```
;; =================================================================================
;; Data definitions:

(@htdd Ball)

(define-struct ball (x y dx dy))
;; Ball is (make-ball Number Number Number Number)
;; interp. (make-ball x y dx dy) is ball
;;   - position x, y    in screen coordinates
;;   - velocity dx, dy  in pixels/tick
(define B1 (make-ball (/ WIDTH 2) (/ HEIGHT 2) 4 -3))
                  A
(@dd-template-rules compound)

(define (fn-for-ball b)
  (... (ball-x b)
       (ball-y b)
       (ball-dx b)
       (ball-dy b)))


;; =================================================================================
;; =================================================================================
;; Functions:

(@htdf main)
(@signature Ball -> Ball)
;; start the game, call with (main B1)
;; <no tests for main functions>

(@template htdw-main)

(define (main b)
  (big-bang b
            (on-draw   render-ball)    ;Ball -> Image
            (on-tick   next-ball)      ;Ball -> Ball
            (on-mouse  handle-mouse)));Ball Integer Integer MouseEvent -> Ball


(@htdf render-ball)
(@signature Ball -> Image)
;; place BALL on image at appropriate x, y coordinate
(check-expect (render-ball (make-ball 20 30 3 3))
              (place-image BALL 20 30 MTS))
                                        B
(check-expect (render-ball (make-ball (- WIDTH 4) (- HEIGHT 5) -2 -3))
              (place-image BALL (- WIDTH 4) (- HEIGHT 5) MTS))
#;
(define (render-ball b) MTS)

(@template Ball)
                  C
(define (render-ball b)
  (place-image BALL (ball-x b) (ball-y b) MTS))


(@htdf next-ball)
(@signature Ball -> Ball)
;; produce ball at next x,y; checks bounces off top/right/bottom/left wall
(check-expect (next-ball      (make-ball (+ LEF 1) TOP  3 -4))
              (bounce-top     (make-ball (+ LEF 1) TOP  3 -4)))
(check-expect (next-ball      (make-ball (+ LEF 1) BOT  3  4))
              (bounce-bottom  (make-ball (+ LEF 1) BOT  3  4)))
```

```
(check-expect (next-ball     (make-ball LEF (+ TOP 1) -3 4))
              (bounce-left   (make-ball LEF (+ TOP 1) -3 4)))
(check-expect (next-ball     (make-ball RIG (+ TOP 1)  3 4))
              (bounce-right  (make-ball RIG (+ TOP 1)  3 4)))
(check-expect (next-ball     (make-ball (/ WIDTH 2) (/ HEIGHT 2) 3 4))
              (glide         (make-ball (/ WIDTH 2) (/ HEIGHT 2) 3 4)))

#;
(define (next-ball b) b)

(@template Number) ;(@template Number) because b is treated as atomic

(define (next-ball b)
  (cond [(touch-top?    b) (bounce-top b)]
        [(touch-bottom? b) (bounce-bottom b)]
        [(touch-right?  b) (bounce-right b)]
        [(touch-left?   b) (bounce-left b)]
        [else
         (glide b)]))


(@htdf handle-mouse)
(@signature Ball Integer Integer MouseEvent -> Ball)
;; replace ball with new ball on mouse click
;; NOTE: uses random, so testing has to use check-random
(check-random (handle-mouse (make-ball 1 2 3 4) 100 200 "button-down")
              (make-ball 100 200 (- 5 (random 11)) (- 5 (random 11))))
(check-random (handle-mouse (make-ball 1 2 3 4) 100 200 "button-up")
              (make-ball 1 2 3 4))
#;
(define (handle-mouse b x y me) b)

(@template MouseEvent)

(define (handle-mouse b x y me)
  (cond [(mouse=? me "button-down")
         (make-ball x y (- 5 (random 11)) (- 5 (random 11)))]
        [else b]))

(@htdf touch-top?)
(@signature Ball -> Boolean)
;; true if ball is going up and edge will hit or pass top edge of box
(check-expect (touch-top?    (make-ball LEF (+ TOP  5) 3 -4)) false)
(check-expect (touch-top?    (make-ball LEF (+ TOP  4) 3 -4)) true)
(check-expect (touch-top?    (make-ball LEF (+ TOP  1) 3 -2)) true)
(check-expect (touch-top?    (make-ball LEF (+ TOP  0) 3  2)) false)
#;
(define (touch-top? b) false)

(@template Ball)

(define (touch-top? b)
  (<= (+ (ball-y b) (ball-dy b)) TOP))


(@htdf touch-bottom?)
(@signature Ball -> Boolean)
;; true if ball is going down and edge will hit or pass bottom edge of box
(check-expect (touch-bottom? (make-ball LEF (- BOT 3) 3  2)) false)
(check-expect (touch-bottom? (make-ball LEF (- BOT 2) 3  2)) true)
(check-expect (touch-bottom? (make-ball LEF (- BOT 0) 3  2)) true)
(check-expect (touch-bottom? (make-ball LEF (- BOT 0) 3 -2)) false)
```

D

E

F