```
;; DO NOT PUT ANYTHING PERSONALLY IDENTIFYING BEYOND YOUR CWL IN THIS FILE.
;; YOUR CWLs WILL BE SUFFICIENT TO IDENTIFY YOU AND, IF YOU HAVE ONE, YOUR
;; PARTNER.
(require 2htdp/universe)
(require 2htdp/image)
(require spd/tags)

(@assignment psets/pset-05);Do not edit or remove this tag

;; If you are:
;;   - A 110 or 107 student replace the first set of '???'s with your cwl.
;;     For problem sets, If you have a partner, please replace the second
;;     set of '???'s with their cwl.  Remember this, it is what you will
;;     do with these @cwl annotations for the whole course.
;;   - A UBC Extended Learning student, replace the first set of ??? with
;;     your email address as confirmed in the email you received from
;;     extended learning.  The handin password is also in that email.
;;     Remember this, it is what you will do with these @cwl annotations
;;     for the whole course.
;;
(@cwl dana28cl ???)

;; Bounce any number of balls around the screen.


;;
;; In this problem set you are given our official solution to problem
;; set 4 (with a few additional things added) as a starting point.
;; We have given you some more constants, a helper function called
;; touch-paddle? which you may use, and a new data defintion called Game.
;; You need to revise the program so that:
;;  - the game includes a paddle that moves back and forth across the
;;    bottom of the screen
;;  - the paddle is controlled by the left and right arrow keys
;;  - when a ball hits the paddle it disappears
;;  - as before the mouse can be used to add balls to the game
;;
;; As stated above, we have given you a new data definition called Game.
;; You MUST revise the program so that it uses Game as the world state.
;; You MUST NOT change the Game data definition in anyway (though you are
;; allowed to add more Game constants).
;;
;; We suggest you work in three distinct phases, making sure your program
;; works correctly at the end of each phase before going on to the next.
;;   - change the program's world state to Game
;;   - provide left/right arrow key control over the paddle
;;   - make it so that when a ball hits the paddle it disappears
;;
;; In each of these phases you should follow the design recipes! Re-work
;; the domain analysis for changing and constant information, update the
;; data definitions, revise the main function, and so on.  Make sure that
;; your tags are correct and that all your tests work correctly before you
;; proceed to the next phase.
;;
;; NOTE: Your on-tick function MUST be designed as a composition of two other
;;       functions called game-with-next-balls and game-with-caught-balls.
;;
;; Note that we are giving you significant help in the starter file.
;; You absolutely MUST USE OUR STARTER AS THE BASIS FOR YOUR WORK.
;;
;; We recommend that you begin by printing this file and planning out what
;; needs to change, what needs to be added, and what will be unchanged.
;;
(@problem 1)
(@htdw ListOfBall)

;; Constants:
(define WIDTH  605)
(define HEIGHT 535)

(define PADDLE-WIDTH 60)
(define PADDLE-THICKNESS 10)
(define PADDLE (rectangle PADDLE-WIDTH PADDLE-THICKNESS "solid" "white"))
(define PADDLE-CTR-Y (- HEIGHT 40))
(define PADDLE-MOVE-PER-KEY 10)

(define BALL-RADIUS 10)

(define TOP             BALL-RADIUS)
(define BOT (- HEIGHT 1 BALL-RADIUS))
(define LEF             BALL-RADIUS)
(define RIG (- WIDTH  1 BALL-RADIUS))

(define BALL (circle BALL-RADIUS "solid" "white"))

(define MTS (rectangle WIDTH HEIGHT "solid" "green"))


;; ========================================================================
```

```racket
;; ============================================================================
;; 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
;; CONSTRAINT: x is in [LEF, RIG]; y is in [TOP, BOT]
(define B1 (make-ball (/ WIDTH 2) (/ HEIGHT 2) 4 -3))

(@dd-template-rules compound)

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

(@htdd ListOfBall)
;; ListOfBall is one of:
;;   - empty
;;   - (cons Ball ListOfBall)
;; interp. a list of balls
(define LOB1 empty)
(define LOB2 (cons B1 empty))

(@dd-template-rules one-of
                    atomic-distinct
                    compound
                    ref
                    self-ref)

(define (fn-for-lob lob)
  (cond [(empty? lob) (...)]
        [else
         (... (fn-for-ball (first lob))
              (fn-for-lob (rest lob)))]))


(@htdd Game)
(define-struct game (balls paddle))
;; Game is (make-game ListOfBall Number)
;; interp. the current state of a game, with all the balls in play,
;;         as well as the x-position of the paddle in screen coordinates
(define G0 (make-game empty (/ WIDTH 2)))
(define G1 (make-game (cons B1 empty) (/ WIDTH 2)))

(@dd-template-rules compound ref)

(define (fn-for-game g)
  (... (fn-for-lob (game-balls g))
       (game-paddle g)))


```

```racket

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

(@htdf main)
(@signature ListOfBall -> ListOfBall)
;; start the game, call with (main LOB1)
;; <no tests for main functions>

(@template-origin htdw-main)

(define (main lob)
  (big-bang lob
    (on-draw   render-balls)   ;ListOfBall -> Image
    (on-tick   next-balls)     ;ListOfBall -> ListOfBall
    (on-key    handle-key)     ;ListOfBall KeyEvent -> ListOfBall
    (on-mouse  handle-mouse))) ;ListOfBall Integer Integer MouseEvent
                               ;   -> ListOfBall

(@htdf render-balls)
(@signature ListOfBall -> Image)
;; render all balls onto MTS
(check-expect (render-balls empty) MTS)
(check-expect (render-balls (cons (make-ball 10 20 3 4)
                                  (cons (make-ball 30 40 1 2)
                                        empty)))
              (place-ball (make-ball 10 20 3 4)
                          (place-ball (make-ball 30 40 1 2)
                                      MTS)))

;(define (render-balls lob) MTS) ;stub

(@template-origin ListOfBall)

(@template
 (define (render-balls lob)
   (cond [(empty? lob) (...)]
         [else
          (... (fn-for-ball (first lob))
               (render-balls (rest lob)))])))

(define (render-balls lob)
  (cond [(empty? lob) MTS]
        [else
         (place-ball (first lob)
                     (render-balls (rest lob)))]))


(@htdf place-ball)
(@signature Ball Image -> Image)
;; place BALL on image at appropriate x, y coordinate
(check-expect (place-ball (make-ball 20 30 3 3) MTS)
              (place-image BALL 20 30 MTS))
```

```
(check-expect (place-ball (make-ball 10 20 -2 -1) empty-image)
              (place-image BALL 10 20 empty-image))
#;
(define (place-ball b img) img)

(@template-origin Ball)

(@template
 (define (place-ball b img)
   (... (ball-x b)
        (ball-y b)
        (ball-dx b)
        (ball-dy b)
        img)))

(define (place-ball b img)
  (place-image BALL (ball-x b) (ball-y b) img))


(@htdf next-balls)
(@signature ListOfBall -> ListOfBall)
;; produce list of balls at their next x, y coordinates
(check-expect (next-balls empty) empty)
(check-expect (next-balls (cons (make-ball (+ LEF 1) TOP 3 -4)
                                (cons (make-ball 200 100 3 4)
                                      empty)))
              (cons (next-ball (make-ball (+ LEF 1) TOP 3 -4))
                    (cons (next-ball (make-ball 200 100 3 4))
                          empty)))

#;
(define (next-balls lob) empty)

(@template-origin ListOfBall)

(@template
 (define (next-balls lob)
   (cond [(empty? lob) (...)]
         [else
          (... (fn-for-ball (first lob))
               (next-balls (rest lob)))])))

(define (next-balls lob)
  (cond [(empty? lob) empty]
        [else
         (cons (next-ball (first lob))
               (next-balls (rest lob)))]))


(@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-origin Number) ; because b is treated as atomic

(@template
 (define (next-ball b)
   (... b)))

(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 ListOfBall Integer Integer MouseEvent -> ListOfBall)
;; adds new ball at x, y to lob
;; NOTE: uses random, so testing has to use check-random
(check-random (handle-mouse empty 100 200 "button-down")
              (cons (make-ball 100 200 (- 5 (random 11)) (- 5 (random 11)))
                    empty))
(check-random (handle-mouse (cons (make-ball 10 20 -3 3) empty)
                            300 100 "button-down")
              (cons (make-ball 300 100 (- 5 (random 11)) (- 5 (random 11)))
                    (cons (make-ball 10 20 -3 3) empty)))
(check-random (handle-mouse empty 100 200 "button-up") empty)
(check-random (handle-mouse (cons (make-ball 10 20 -3 3) empty)
                            100 200 "button-up")
              (cons (make-ball 10 20 -3 3) empty))
#;
(define (handle-mouse lob x y me) empty)

(@template-origin MouseEvent)

(@template
 (define (handle-mouse lob x y me)
   (cond [(mouse=? me "button-down") (... lob x y)]
         [else
          (... lob x y)])))

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

```racket
(@htdf handle-key)
(@signature ListOfBall KeyEvent -> ListOfBall)
;; clear all balls if space key pressed; else do nothing
(check-expect (handle-key (cons (make-ball (/ WIDTH 2) (/ HEIGHT 2) 2 4)
                                (cons (make-ball (+ TOP 2) (+ LEF 5) 3 2)
                                      empty))
                          " ")
              empty)
(check-expect (handle-key (cons (make-ball (/ WIDTH 3) (/ HEIGHT 4) 1 -3)
                                (cons (make-ball (+ TOP 5) (+ LEF 2) 3 -2)
                                      empty))
                          "c")
              (cons (make-ball (/ WIDTH 3) (/ HEIGHT 4) 1 -3)
                    (cons (make-ball (+ TOP 5) (+ LEF 2) 3 -2)
                          empty)))
#;
(define (handle-key lob ke) empty)

(@template-origin KeyEvent)

(@template
 (define (handle-key lob ke)
   (cond [(key=? ke " ") (... lob)]
         [else
          (... lob)])))

(define (handle-key lob ke)
  (cond [(key=? ke " ") empty]
        [else lob]))


(@htdf touch-paddle?)
(@signature Ball Number -> Boolean)
;; produce true if ball's center is inside the paddle
;; NOTE: There are many better and more complex ways to design this function.
;;       This design is fairly primitive (just checks that the center of the
;;       ball is in the paddle), but people playing the game shouldn't see
;;       much difference if the balls are moving quickly.
(check-expect (touch-paddle? (make-ball (- 100 (/ PADDLE-WIDTH 2) 1)
                                        PADDLE-CTR-Y
                                        1 2)
                             100)
              false)
(check-expect (touch-paddle? (make-ball (- 100 (/ PADDLE-WIDTH 2))
                                        PADDLE-CTR-Y
                                        1 2)
                             100)
              true)
(check-expect (touch-paddle? (make-ball (+ 100 (/ PADDLE-WIDTH 2))
                                        PADDLE-CTR-Y
                                        1 2)
                             100)
              true)
(check-expect (touch-paddle? (make-ball (+ 100 (/ PADDLE-WIDTH 2) 1)
                                        PADDLE-CTR-Y
                                        1 2)
                             100)
              false)
(check-expect (touch-paddle?
               (make-ball (+ 100 (/ PADDLE-WIDTH 2))
                          (- PADDLE-CTR-Y (/ PADDLE-THICKNESS 2) 1)
                          1 2)
               100)
              false)
(check-expect (touch-paddle?
               (make-ball (+ 100 (/ PADDLE-WIDTH 2))
                          (- PADDLE-CTR-Y (/ PADDLE-THICKNESS 2))
                          1 2)
               100)
              true)
(check-expect (touch-paddle?
               (make-ball (+ 100 (/ PADDLE-WIDTH 2))
                          (+ PADDLE-CTR-Y (/ PADDLE-THICKNESS 2))
                          1 2)
               100)
              true)
(check-expect (touch-paddle?
               (make-ball (+ 100 (/ PADDLE-WIDTH 2))
                          (+ PADDLE-CTR-Y (/ PADDLE-THICKNESS 2) 1)
                          1 2)
               100)
              false)
(check-expect (touch-paddle? (make-ball (+ 30 (/ PADDLE-WIDTH 2))
                                        PADDLE-CTR-Y
                                        1 2)
                             30)
              true)

(@template-origin Ball)

(@template
 (define (touch-paddle? b p)
   (... (ball-x b)
        (ball-y b)
        (ball-dx b)
        (ball-dy b)
        p)))

(define (touch-paddle? b p)
  (and (<= (- p (/ PADDLE-WIDTH 2))
           (ball-x b)
           (+ p (/ PADDLE-WIDTH 2)))
       (<= (- PADDLE-CTR-Y (/ PADDLE-THICKNESS 2))
           (ball-y b)
           (+ PADDLE-CTR-Y (/ PADDLE-THICKNESS 2)))))


(@htdf touch-top?)
(@signature Ball -> Boolean)
```

```racket
;; true if ball is going up and edge will hit 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-origin Ball)

(@template
 (define (touch-top? b)
   (... (ball-x b)
        (ball-y b)
        (ball-dx b)
        (ball-dy b))))

(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 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)
#;
(define (touch-bottom? b) false)

(@template-origin Ball)

(@template
 (define (touch-bottom? b)
   (... (ball-x b)
        (ball-y b)
        (ball-dx b)
        (ball-dy b))))

(define (touch-bottom? b)
  (>= (+ (ball-y b) (ball-dy b)) BOT))


(@htdf touch-left?)
(@signature Ball -> Boolean)
;; true if ball is going left and edge will hit left  edge of box
(check-expect (touch-left?   (make-ball (+ LEF 6) TOP -5 2)) false)
(check-expect (touch-left?   (make-ball (+ LEF 5) TOP -5 2)) true)
(check-expect (touch-left?   (make-ball (+ LEF 0) TOP -5 2)) true)
(check-expect (touch-left?   (make-ball (+ LEF 0) TOP  3 2)) false)
#;
(define (touch-left? b) false)

(@template-origin Ball)

(@template
 (define (touch-left? b)
   (... (ball-x b)
        (ball-y b)
        (ball-dx b)
        (ball-dy b))))

(define (touch-left? b)
  (<= (+ (ball-x b) (ball-dx b)) LEF))


(@htdf touch-right?)
(@signature Ball -> Boolean)
;; true if ball is going right and edge will hit right edge of box
(check-expect (touch-right?  (make-ball (- RIG 6) TOP  5 2)) false)
(check-expect (touch-right?  (make-ball (- RIG 5) TOP  5 2)) true)
(check-expect (touch-right?  (make-ball (- RIG 0) TOP  5 2)) true)
(check-expect (touch-right?  (make-ball (- RIG 0) TOP -3 2)) false)
#;
(define (touch-right? b) false)

(@template-origin Ball)

(@template
 (define (touch-right? b)
   (... (ball-x b)
        (ball-y b)
        (ball-dx b)
        (ball-dy b))))

(define (touch-right? b)
  (>= (+ (ball-x b) (ball-dx b)) RIG))


(@htdf bounce-top)
(@signature Ball -> Ball)
;; produce a ball with top edge 1 pixel off top of box, moving down
;; CONSTRAINT: assume ball is close to top edge and moving up
(check-expect (bounce-top (make-ball (+ LEF 1) (+ TOP 3) 2 -4))
              (make-ball (+ LEF 1) (+ TOP 1) 2  4))
(check-expect (bounce-top (make-ball (+ LEF 2) (+ TOP 6) 3 -7))
              (make-ball (+ LEF 2) (+ TOP 1) 3 7))
#;
(define (bounce-top b) b)

(@template-origin Ball)

(@template
 (define (bounce-top b)
   (... (ball-x b)
        (ball-y b)
        (ball-dx b)
        (ball-dy b))))

(define (bounce-top b)
  (make-ball (ball-x b) (+ TOP 1) (ball-dx b) (- (ball-dy b))))


(@htdf bounce-bottom)
(@signature Ball -> Ball)
```

```
;; produce a ball with bottom edge 1 pixel off bottom of box,
moving up
;; CONSTRAINT: assume ball is close to bottom edge and moving down
(check-expect (bounce-bottom (make-ball (+ LEF 1) (- BOT 3) 2 4))
              (make-ball (+ LEF 1) (- BOT 1) 2  -4))
(check-expect (bounce-bottom (make-ball (+ LEF 2) (- BOT 6) 3 7))
              (make-ball (+ LEF 2) (- BOT 1) 3 -7))
#;
(define (bounce-bottom b) b)

(@template-origin Ball)

(@template
 (define (bounce-bottom b)
   (... (ball-x b)
        (ball-y b)
        (ball-dx b)
        (ball-dy b))))

(define (bounce-bottom b)
  (make-ball (ball-x b) (- BOT 1) (ball-dx b) (- (ball-dy b))))

(@htdf bounce-left)
(@signature Ball -> Ball)
;; produce a ball with left edge 1 pixel off left of box, moving
right
;; CONSTRAINT: assume ball is close to left edge and moving left
(check-expect (bounce-left (make-ball (+ LEF 3) (+ TOP 2) -4 4))
              (make-ball (+ LEF 1) (+ TOP 2) 4 4))
(check-expect (bounce-left (make-ball (+ LEF 5) (+ TOP 2) -8 4))
              (make-ball (+ LEF 1) (+ TOP 2) 8 4))
#;
(define (bounce-left b) b)

(@template-origin Ball)

(@template
 (define (bounce-left b)
   (... (ball-x b)
        (ball-y b)
        (ball-dx b)
        (ball-dy b))))

(define (bounce-left b)
  (make-ball (+ LEF 1) (ball-y b) (- (ball-dx b)) (ball-dy b) ))


(@htdf bounce-right)
(@signature Ball -> Ball)
;; produce a ball with right edge 1 pixel off right of box, moving
left
;; CONSTRAINT: assume ball is close to right edge and moving right
(check-expect (bounce-right (make-ball (- RIG 3) (+ TOP 1) 4 4))
              (make-ball (- RIG 1) (+ TOP 1) -4 4))
(check-expect (bounce-right (make-ball (- RIG 5) (+ TOP 1) 8 4))
              (make-ball (- RIG 1) (+ TOP 1) -8 4))
#;
(define (bounce-right b) b)

```

```
(@template-origin Ball)

(@template
 (define (bounce-right b)
   (... (ball-x b)
        (ball-y b)
        (ball-dx b)
        (ball-dy b))))

(define (bounce-right b)
  (make-ball (- RIG 1) (ball-y b) (- (ball-dx b)) (ball-dy b)))


(@htdf glide)
(@signature Ball -> Ball)
;; move ball by dx dy
;; CONSTRAINT: ball is not touching or about to touch any edge of
the box
(check-expect (glide (make-ball 100 200 2 3)) (make-ball 102 203 2
3))
(check-expect (glide (make-ball 50 220 -3 -2)) (make-ball 47 218
-3 -2))
#;
(define (glide b) b)

(@template-origin Ball)

(@template
 (define (glide b)
   (... (ball-x b)
        (ball-y b)
        (ball-dx b)
        (ball-dy b))))

(define (glide b)
  (make-ball (+ (ball-x b) (ball-dx b))
             (+ (ball-y b) (ball-dy b))
             (ball-dx b)
             (ball-dy b)))
```