CS 2500, Spring 2014
Problem Set 7

Due date: 7pm Tuesday February 25, 2014

Programming Language: BSL with List Abbreviations

Purpose: this problem set concerns the design of functions on self-referential data definitions in the context of interactive programs.

Finger Exercises HtDP/2e: 198, 199, 203, 204, 205

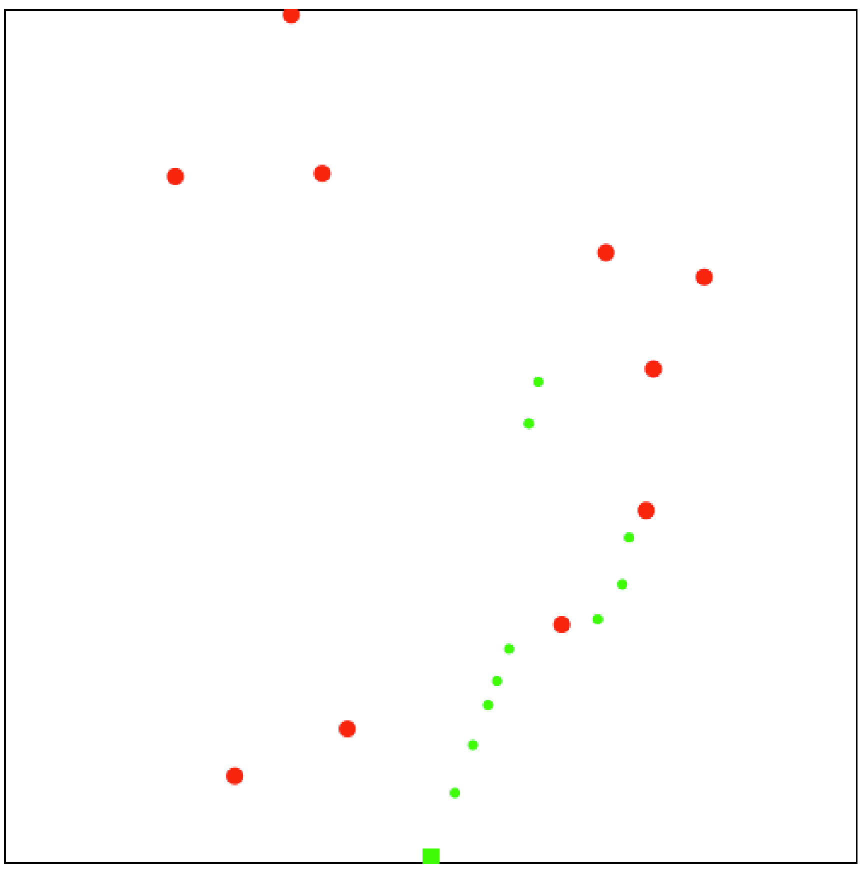
You **must** follow the design recipe. The graders will look for data definitions, signatures, purpose statements, examples/tests, and properly organized function definitions. For the latter, you **must** design templates. You do not need to include the templates however. If you do, make sure to comment them out.

Develop the **Missile Defense** game. In this game you control an artillery gun centered at the bottom of a 500x500 pixel canvas. Clicking the mouse anywhere on the canvas causes one *bullet* to be fired from the gun in that direction. The "enemy" is simultaneously firing *missiles* at you. Unlike bullets, which are always launched under your control from the gun, missiles are launched randomly from invisible silos just above the top of the canvas. There is a silo at every possible horizontal location (i.e. missiles may start out at any in-bounds *x* coordinate), and each missile has a random direction but always aims downwards so that, unless you shoot it down, it will reach the bottom of the canvas without going out of bounds. Once shot, both missiles and bullets travel in straight lines. A missile is shot down when a bullet collides with it; in that case both the bullet and the missile "explode" (i.e. disappear). Both bullets and missiles are represented as circles, though their radii and color should be different to distinguish them.

Note that it should be possible for the missiles to travel "downwards" at an angle, not just straight down. That is, missiles should start at y=0 with any random *in-bounds* x-coordinate (i.e. in the range 0 to 499) and end at y=499 with any random in-bounds x-coordinate, not necessarily the same x-coordinate at which they started.

The goal of the game is to survive as long as possible. You start out with an initial *health* count, which is decremented by one each time a missile reaches it all the way to the bottom of the canvas. The game is over when your health count reaches zero. You are not required to display the health count (though you can if you like), but you are required to make the game end when it reaches zero.

Here is an image of the game in play:



Data Definitions: Use these data definitions unless you *really* have a strong reason for departing (e.g., you want to add extra features like top-score list, animated explosions, smart missiles, etc. -- first see the advice below):

```
 ;;; A Sprite is a (make-sprite Posn Posn Number String)
 ;;; interp. (make-sprite p1 p2 r c) is a sprite with p1
 ;;; as its location, p2 is its velocity, r is the size
 ;;; of the radius of the sprite and c is its color. Location,
 ;;; velocity and size are in computer-graphic/pixel coordinates.
 ;;; A sprite represents either an attacker's missile or a defender's
 ;;; anti-missile bullet.
(define-struct sprite (loc vel size color))

 ;;; A LOS (list of sprites) is one of:
 ;;; - empty
 ;;; - (cons Sprite LOS)

 ;;; A World structure is a (make-world LOS LOS Number)
 ;;; - interp. (make-world m b h) is a world with m missiles
 ;;; (the missiles attacking the player), b bullets
 ;;; (bullets launched by the player) and h health (current
 ;;; health of the player -- game-over if health <= 0)
(define-struct world (missiles bullets health))
```

Top Level Functions: We also strongly suggest you use these top-level functions:

```
;;; move-bullets: World -> World
;;; Move the bullets one time step and remove any that have gone off
screen.

;;; move-missiles: World -> World
;;; Move the missiles one time step.

;;; remove-dead-missiles-and-bullets: World -> World
;;; Remove every missile that is touching some bullet and vice-versa.

;;; detonate-missiles: World -> World
;;; Remove missiles that landed... and decrement the player's health if
any did.

;;; maybe-fire-missile: World -> World
;;; If we haven't maxed out on missiles, launch another one.
```

Note the shared functionality in some of these. For example, both `move-bullets` and `move-missiles` need to advance every `Sprite` in an `LOS`. And both `move-bullets` and `detonate-missiles` need to cull out-of-bounds `Sprite`s. You should implement helper functions for operations like these so that you don't end up writing the same `Sprite` processing code in multiple places.

On-Tick Handler: Given those functions, this should be the organization of your `on-tick` handler:

```
;;; update-world: World -> World
;;; Step the world one tick.
```

1. Move the bullets and remove offscreen ones.
2. Move the missiles.
3. Kill any destroyed missiles.
4. Detonate the missiles that landed.
5. Maybe launch another missile.

Additional Advice:

- Note that two circles "collide" if the distance between their centers is less than the sum of their radii. Remember, the formula for the distance between two points at coordinates (x0, y0) and (x1, y1) is

  ```
  (sqrt (+ (sqr (- x1 x0)) (sqr (- y1 y0))))
  ```

- You may implement more features, if you like (e.g., health display, increasing difficulty, fancier graphics, game pausing). However, extra features won't save you from points taken off if your code has bugs or isn't well written. You will not receive 100% credit simply for having code that works. For full credit, your code must work and be well written. So you should put your effort into writing clean, readable, bug-free code.

- As always, we advise you that the Design Recipe will help you get your code written.
- Start early. It will take you time to work out the assignment.