CS 2500, Spring 2013
Problem Set 7

Due date: Tuesday, February 26 @ 11:59pm

Programming Language: Beginning Student Language with List Abbreviations

You must work on and submit this homework with your assigned partner. You will receive no credit for ignoring your partner and working on the homework alone or not working on the homework at all.

Using Blackboard, submit a single Racket file containing all of the code and documentation for this assignment. Place your names and husky email addresses in a comment at the beginning of your file.

Name your file: hw7-lastname1-lastname2.rkt

You must follow the design recipe in your solutions: graders will look for data definitions, contracts, purpose statements, examples/tests, and properly organized function definitions. For the latter, you must follow templates. You do not need to include the templates with your homework, however, unless the question asks for it.

Task:

Develop the Frogger game. Here is a link to a free version of Frogger for those unfamiliar with the game: http://www.happyhopper.org. In this game, the player is a frog who tries to move from the bottom of the screen to the top of the screen. The screen is divided into rows. The player starts in the bottom row, crosses five rows of traffic, and wins if they reach the top row.

In the traffic rows, the player must not collide with any of the vehicles present in these rows. The traffic alternates direction and includes 4 vehicles per road. If the player gets hit by a vehicle, the player loses and the game ends. When a vehicle passes off the edge of the screen, a new entity of the same type is created on the opposite edge, so that there are always the same number of vehicles, and they are all evenly spaced.

The player can move his character in four ways: up, down, left and right. Develop your own data definitions and structures for frogs, and vehicles (and any auxiliary ones you might need, like worlds).

The following data definitions may or may not be useful (and indicative of the kind of operations you might want to develop) for writing this program:

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Data Definitions
;; A Player is a (make-player Number Number Direction)
(define-struct player (x y dir))

;; A Vehicle is a (make-vehicle Number Number Direction)
(define-struct vehicle (x y dir))
```

```
;; A Set of Vehicles (VSet) is one of:
;; - empty
;; - (cons Vehicle VSet)

;; A World is a (make-world Player VSet)
;;The VSet represents the set of vehicles moving across the screen
(define-struct world (player vehicles))
```

You should think about how you will determine whether the player has been hit by a vehicle. You may want to use something like the following functions:

```
;; in-range?: Number, Number, Number -> Boolean
;; is n1 within range of n2?
(define (in-range? n1 n2 range)
 (and (< n1 (+ n2 range))
      (> n1 (- n2 range))))

;; hit?: Player, Vehicle -> Boolean
;; was the player hit by the vehicle?
(define (hit? player vehicle)
 (and (= (player-y player)
         (vehicle-y vehicle))
      (in-range? (player-x player)
                 (vehicle-x vehicle)
                 (+ (/ (image-width PLAYER) 2)
                    (/ (image-width VEHICLE) 2)))))
```

Some advice:

Don't include images in your assignment. You can represent players and vehicles using image functions such as `circle, triangle, square, overlay,` etc.

You may implement other features, if you like (e.g., multiple lives, vehicles of varying lengths, and/or a river). 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.

You will be upgrading the game to full Frogger in a future assignment. So effort expended in making sure that your code is clean and well-written will be rewarded when you have to extend it later – it is quite difficult to modify and extend code that is a snarled-up, confused pile of chaos.

As always, you should use the Design Recipe help get your code written.

Start early. It will take you time to work out the assignment.