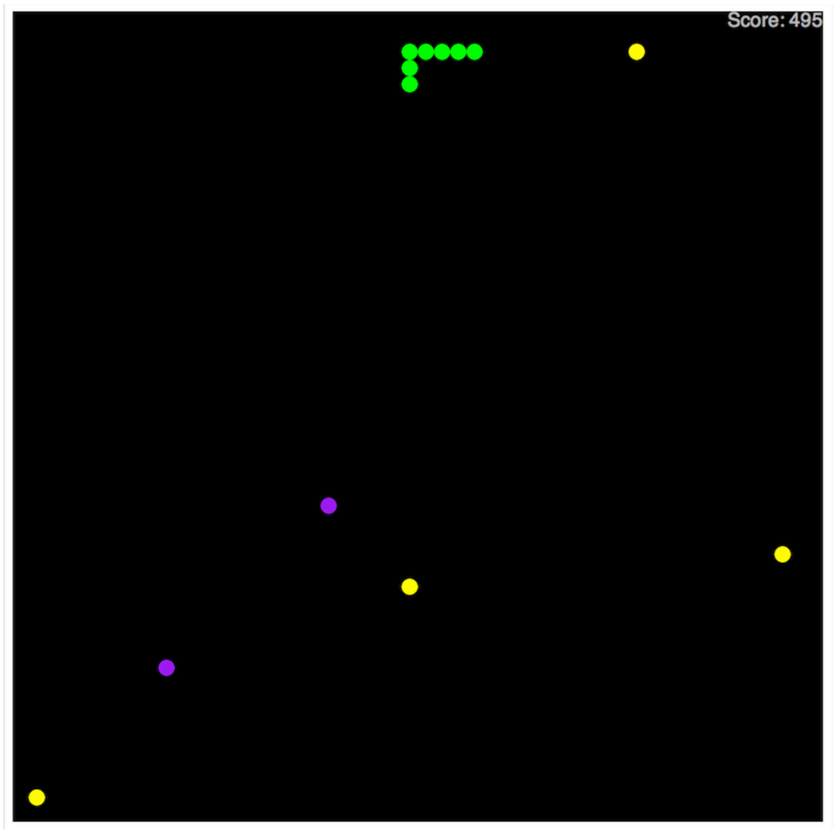
EECS 111 Final Project

due Thursday, March 17th by 11:00am (please note that this is 11AM – in the MORNING)

Language: Intermediate Student with lambda

Your goal is to implement a variation of the game Snake.



In this game, you control a snake that wanders around a square board, eating food (yellow morsels) and avoiding purple obstacles. Each time the snake eats a morsel of food, it gets a little bit longer. If the snake ever runs into itself, runs into the edge of the board or runs into a purple obstacle, the game is over. The snake never stops moving; the player can control only the direction that the snake moves by using the arrow keys.

To begin, save the files snake-lib.rkt and provide.rkt in the folder where you will save the code you write. Do not make ANY changes to these files. Start your code (in a new file called final_project.rkt) with the line

```
(require "snake-lib.rkt")
```

For this project, we will use the following data definitions.

```
; a game is
; (make-game snake food obstacles nat)
; (define-struct game [snake food obstacles ticks])

; a direction is either
; - 'up
; - 'down
; - 'left
; - 'right

; a snake is
; (make-snake direction body)
; (define-struct snake [heading segments])

; a body is either
; - (cons posn empty)
; - (cons posn body)
; x-coordinates increase from 1 to board-length (inclusive) toward the right
; y-coordinates increase from 1 to board-length (inclusive) toward the top
; the default value for board-length is 50.

; a food is either
; - empty
; - (cons posn food)

; obstacles is either
; - empty
; - (cons posn obstacles)
```

If you add these definitions to your code, the define-struct lines must remain commented as they are defined in already in snake-lib.rkt. They are listed here just for your reference.

This **`require`** line also provides two other definitions:

- `board-length`, the length of one side of the board (measured in terms of snake body segments – by default, this is set to 50), and
- `play-game`, a function described below.

Develop the following functions (and any necessary helper functions), which will serve as argument to the `play-game` function (don't forget about check-expects and a signature/purpose for every function):

- `; add-food : game posn -> game`

  This function adds a morsel of food at the specified board position.

- `; change-direction : game direction -> game`

  This function changes the direction in which the snake is traveling.

- `; game-score : game -> nat`

  This function computes the player's score, based on the snake's length and the time (ticks) taken to reach that length. One possibility is to multiply the snake's length by 100 and subtract the elapsed ticks, but you're welcome to experiment with other definitions.

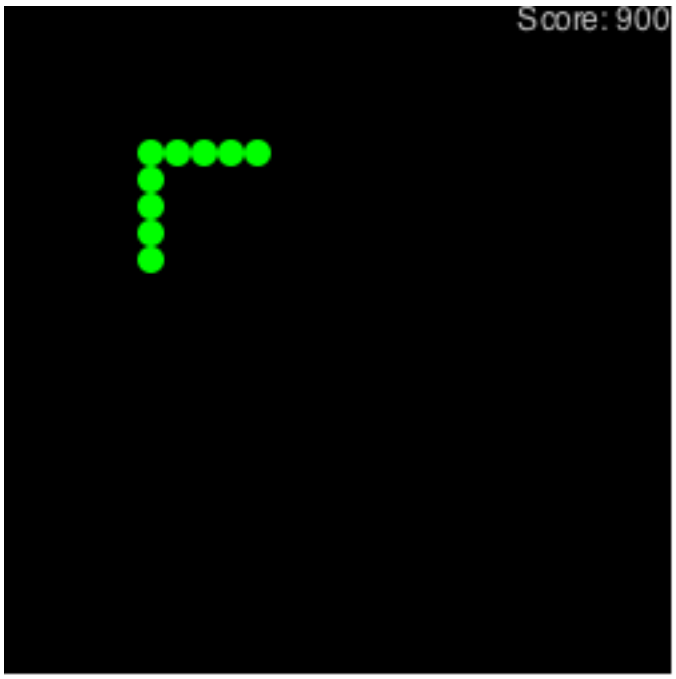
- `; game-over? : game -> boolean`

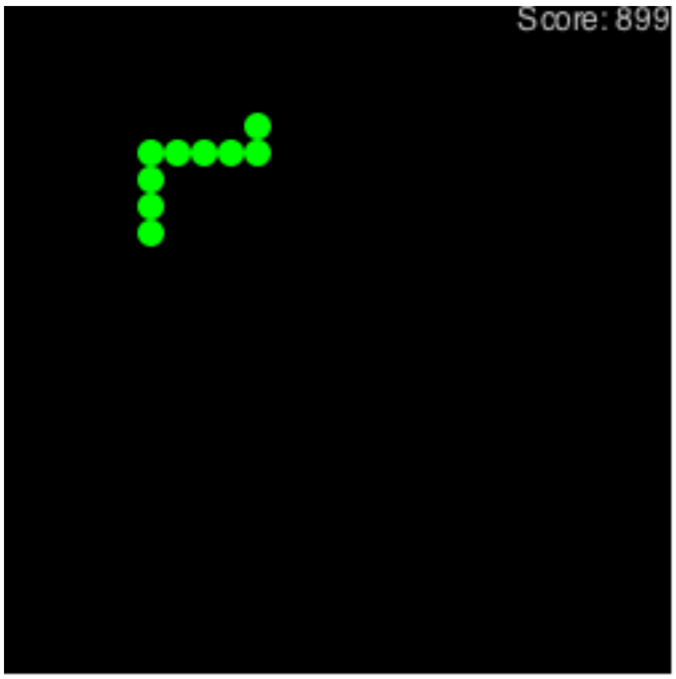
  The game ends when the snake runs into itself, a wall, or an obstacle (purple dot). This function detects when such a collision has occurs. To be clear, if the x and y coordinates of all of the snakes segments fall between 1 and 50 (board-length) inclusive, this means that the snake has NOT hit a wall. If the x or y coordinate of any of the segments becomes 0 or 51, then the snake has hit a wall. Please be very careful when checking the bounds here.

  `; advance-game : game -> game`

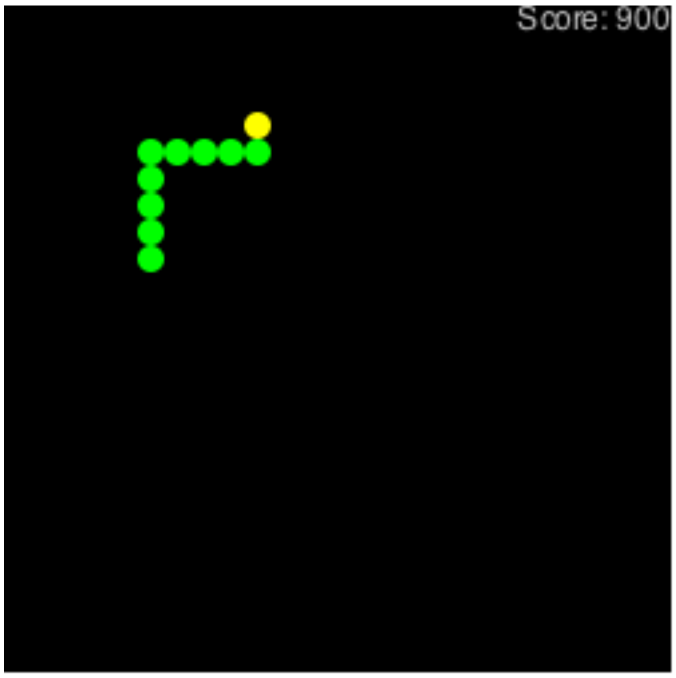
  This function moves the game forward one step. One step increments the game's tick component and moves the snake, possibly causing it to eat and grow. Moving the snake means that the snake both gains and loses a segment (unless it eats). The new segment's coordinates are determined by the segment that was previously at the front of the snake and the direction the snake is heading. The snake loses the oldest segment, namely the one that was previously at the end of the snake.

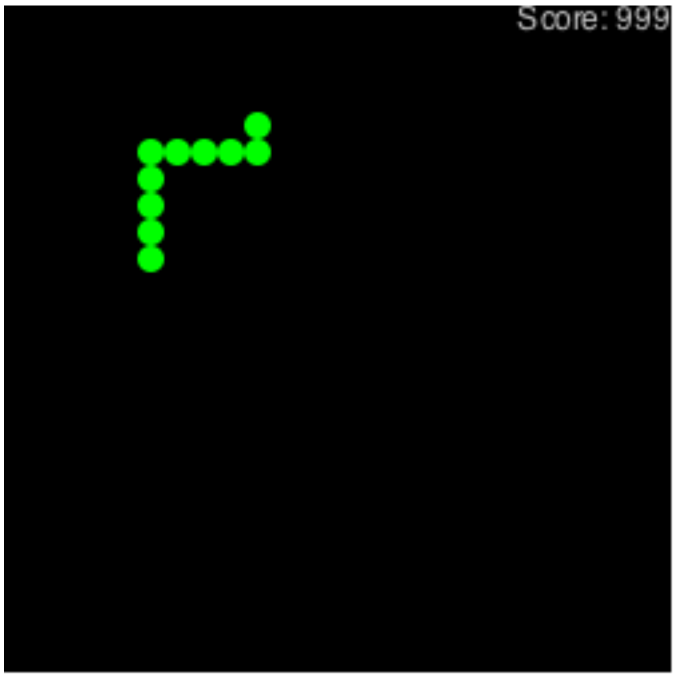
For example, the following shows the snake before and after a single step when the snake is moving upwards without eating.





If that same step were to lead the snake to a morsel of food, then the game would progress as follows.





This function does not replace eaten food; `play-game` (below) handles that task. You **do** need to handle removing the eaten food, however.

After you have written and thoroughly tested these functions, define an initial game value `game-start` and make the following call to play the game.

```
(play-game game-start
           advance-game
           add-food
           change-direction
           game-score
           game-over?)
```

Submit your final_project.rkt file to the handin server in the usual way. This file must include 2 check-expects and a signature/purpose for every function.

Note: As you work on this project, you may consider what would happen if piece of food happens to appear in the same position as an obstacle. Please ignore this issue. That is, don't worry about testing for this situation.