Project 3: Snake Revisited

Due: Before lab starts on November 20th

Your goal is to extend and improve your implementation of the Snake game from the first project. You must:

- fix all of the marked problems from your solution to the original Snake lab (see me if you have any questions on what those marks mean),
- use the `testing.ss` teachpack to test your program (even if the annotations say to use `equal?` in your tests, use this instead). See Help Desk or ask me or on the mailing list if you have questions about it. For functions that use `random`, just write out the test case and an approximate expected answer and compare them by hand.
- use the built-in functions: `map`, `filter`, `ormap`, `andmap` and `foldr` (see Help Desk for details of those functions) when possible,
- add two new kinds of food:
  - shrinking food: When the snake eats this kind of food, its size is halved, and
  - foody food: When the snake eats this kind of food, 10 new pieces of food are added to the board. Feel free to use a different number than 10 (but at least two!) if it matches your game better.

  The score should still be based on the number of food particles the snake has eaten (not the current length of the snake).

HINT: First think about how your data definitions must change. This will alert you to what functions must change (and how they must change).

Use a function like this to determine what kind of food to generate:

```
;; generate-food : posn -> food-morsel
;; to generate some food at the posn p
(define (generate-food p)
  (local [(define n (random 10))]
    (cond
      [(= n 0) ...shrinking food...]
      [(= n 1) ...foody food...]
      [else ...regular food...])))
```

That function generates shrinking food 10% of the time, foody food 10% of the time, and regular food 80% of the time. Feel free to adjust the argument to `random` and the case dispatch if different percentages work better for your implementation of Snake.