Part II: Compound Data of Finite Size

Marco T. Morazán

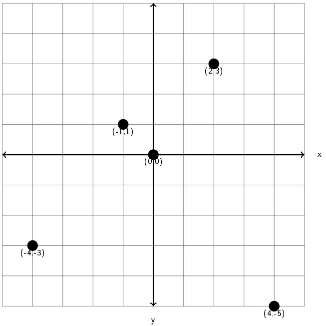
Seton Hall University

Outline

Structures

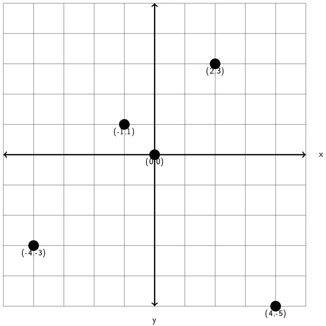
- An important step in problem solving is choosing a data representation
- We have seen how a single value is used to represent the single characteristic that may change

Structures

- An important step in problem solving is choosing a data representation
- We have seen how a single value is used to represent the single characteristic that may change
- 

- Consider integer-based points
- An integer-based point has: an integer x coordinate and an integer y coordinate
- This is an example of *compound data* of finite size

Structures

- An important step in problem solving is choosing a data representation
- We have seen how a single value is used to represent the single characteristic that may change
- 

- Consider integer-based points
- An integer-based point has: an integer x coordinate and an integer y coordinate
- This is an example of *compound data* of finite size
- Compound data of finite size has a constant number of characteristics that may vary from one instance of the data to another

Structures

- How are functions that process compound data of finite size designed?

Structures

- How are functions that process compound data of finite size designed?
- Consider the problem of determining if an integer-based point is in Q1
- How do we represent an integer-point?

Structures

- How are functions that process compound data of finite size designed?
- Consider the problem of determining if an integer-based point is in Q1
- How do we represent an integer-point?
- Need two integers
- OK, but why are we talking about two integers when the problem is about integer-based points?
- Why think about x and y coordinates as two separate values instead of a single value?

Structures

- ```
  ;; Sample expression definitions for in-Q1?
  (define CONSTANT-1 (and (> 3 0) (> 11 0)))
  (define CONSTANT-2 (and (> -5 0) (> 23 0)))
  ```
  ```

Structures

- ```
  ;; integer integer → Boolean
  ;; Purpose: To determine if the given coordinates for a
  ;;          point is in Q1
  (define (in-Q1? x y)
  ```

- ```
  ;; Sample expression definitions for in-Q1?
  (define CONSTANT-1 (and (> 3 0) (> 11 0)))
  (define CONSTANT-2 (and (> -5 0) (> 23 0)))
  ```

Structures

- ```
  ;; integer integer → Boolean
  ;; Purpose: To determine if the given coordinates for a
  ;;          point is in Q1
  (define (in-Q1? x y)
  ```

- ```
  ;; Sample expression definitions for in-Q1?
  (define CONSTANT-1 (and (> 3 0) (> 11 0)))
  (define CONSTANT-2 (and (> -5 0) (> 23 0)))
  ```

- ```
  ;; Tests using sample computations for in-Q1?
  (check-expect (in-Q1? 3 11)  CONSTANT-1)
  (check-expect (in-Q1? -5 23) CONSTANT-2)

  ;; Tests using sample values for in-Q1?
  (check-expect (in-Q1? -3 -7) #false)
  (check-expect (in-Q1? 83 -4) #false)
  ```
  ```

Structures

- ```
  ;; integer integer → Boolean
  ;; Purpose: To determine if the given coordinates for a
  ;;          point is in Q1
  (define (in-Q1? x y)
  ```
- ```
    (and (> x 0) (> y 0)))
  ```
- ```
  ;; Sample expression definitions for in-Q1?
  (define CONSTANT-1 (and (> 3 0) (> 11 0)))
  (define CONSTANT-2 (and (> -5 0) (> 23 0)))
  ```
- ```
  ;; Tests using sample computations for in-Q1?
  (check-expect (in-Q1? 3 11)  CONSTANT-1)
  (check-expect (in-Q1? -5 23) CONSTANT-2)

  ;; Tests using sample values for in-Q1?
  (check-expect (in-Q1? -3 -7) #false)
  (check-expect (in-Q1? 83 -4) #false)
  ```

Structures

- The problem with the solution is that it does not exhibit separation of concerns
- We need a function that takes as input a two-dimensional integer point
- We also need a function to process an integer x coordinate and another function to process an integer y coordinate

Structures

- The problem with the solution is that it does not exhibit separation of concerns
- We need a function that takes as input a two-dimensional integer point
- We also need a function to process an integer x coordinate and another function to process an integer y coordinate
- Our immediate need, therefore, is a mechanism to create a point value out of two coordinates
- With such a mechanism, we may rewrite `in-Q1?` as something similar to:

```
;; 2D-ipoint → Boolean
;; Purpose: To determine if the given 2D-ipoint is in Q1
(define (in-Q1? a-2dipoint)
  (and (x-in-Q1? (x-of a-2dipoint))
       (y-in-Q1? (y-of a-2dipoint))))
```

- Most will argue that the above function, if it were possible to write, is easier to understand than the previous version

Structures

The `posn` Structure

- In `BSL`, compound data of finite size may be represented using a *structure*
- A structure combines a fixed number of values into a single piece of data
- For every structure there is a *constructor*
- There is a *selector* for each characteristic

Structures

The posn Structure

- In BSL, compound data of finite size may be represented using a *structure*
- A structure combines a fixed number of values into a single piece of data
- For every structure there is a *constructor*
- There is a *selector* for each characteristic
- BSL provides the `posn` structure
- A `posn` is intended to represent a position in a Cartesian plane
- Two characteristics (or fields): `x` and `y`
- Constructor: `make-posn`
- Two selector functions: `posn-x` and `posn-y` to retrieve the value of `y`

Structures

The posn Structure

- In BSL, compound data of finite size may be represented using a *structure*
- A structure combines a fixed number of values into a single piece of data
- For every structure there is a *constructor*
- There is a *selector* for each characteristic
- BSL provides the posn structure
- A posn is intended to represent a position in a Cartesian plane
- Two characteristics (or fields): x and y
- Constructor: make-posn
- Two selector functions: posn-x and posn-y to retrieve the value of y
- Sample program:

```
(define A-POSN  (make-posn  3.2 9))
(define A-POSN2 (make-posn -8 9))

(check-expect (posn-x A-POSN)   3.2)
(check-expect (posn-x A-POSN2) -8)
(check-expect (posn-y A-POSN)   9)
(check-expect (posn-y A-POSN2)  9)
(check-expect (= (posn-x A-POSN) (posn-x A-POSN2)) #false)
(check-expect (= (posn-y A-POSN) (posn-y A-POSN2)) #true)
```

Structures

The posn Structure

- Two laws that govern posns:

```
(posn-x (make-posn  x y)) = x
(posn-y (make-posn  x y)) = y
```

Structures

The posn Structure

- Two laws that govern posns:

```
(posn-x (make-posn  x y)) = x
(posn-y (make-posn  x y)) = y
```

- Laws are not enough to properly use posns:

```
(define A-POSN  (make-posn  "Animating" "Programs"))

(check-expect (posn-x A-POSN)   "Animating")
(check-expect (posn-y A-POSN)   "Programs")
(check-expect (string=? (string-append  (posn-x A-POSN)
                                        (posn-y A-POSN))
                        "Animating Programs")
              #true)
```

- A-POSN certainly does not represent a 2D-point
- The constructor does not enforce any type rules for x and y
- How do we make clear what a posn represents? How do we prevent misuse?

Structures

The posn Structure

- To process compound data of fixed size a data definition is needed

Structures

The posn Structure

- To process compound data of fixed size a data definition is needed
-
```
#|;; An x-coordinate (xcoord) is a real number
  ;; A y-coordinate (ycoord) is a real number
  ;; A 2D-point is a structure (make-posn xcoord ycoord)
```

Structures

The posn Structure

- To process compound data of fixed size a data definition is needed
- ```
  #|;; An x-coordinate (xcoord) is a real number
    ;; A y-coordinate (ycoord) is a real number
    ;; A 2D-point is a structure (make-posn xcoord ycoord)
  ```
- ```
    (define XCOORD1   ...) ...   (define YCOORD1   ...) ...
    (define 2D-POINT1 ...) ...
  ```

Structures

The posn Structure

- To process compound data of fixed size a data definition is needed
- ```
  #|;; An x-coordinate (xcoord) is a real number
    ;; A y-coordinate (ycoord) is a real number
    ;; A 2D-point is a structure (make-posn xcoord ycoord)
  ```
- ```
  (define XCOORD1   ...) ...   (define YCOORD1   ...) ...
  (define 2D-POINT1 ...) ...
  ```
- ```
  ;; xcoord ... → ...     Purpose: ...
  (define (f-on-xcoord an-xcoord ...) ... an-xcoord...)
  ;; Sample expressions for f-on-xcoord
  (define XCOORD1-VAL ...) ...
  ;; Tests using sample computations for f-on-xcoord
  (check-expect (f-on-xcoord XCOORD1 ...) XCOORD1-VAL) ...
  ;; Tests using sample values for f-on-xcoord
  (check-expect (f-on-xcoord ...) ...) ...
  ```
  ```

Structures

The posn Structure

- To process compound data of fixed size a data definition is needed
- ```
  #|;; An x-coordinate (xcoord) is a real number
    ;; A y-coordinate (ycoord) is a real number
    ;; A 2D-point is a structure (make-posn xcoord ycoord)
  ```
- ```
  (define XCOORD1   ...) ...   (define YCOORD1   ...) ...
  (define 2D-POINT1 ...) ...
  ```
- ```
  ;; xcoord ... → ...     Purpose: ...
  (define (f-on-xcoord an-xcoord ...) ... an-xcoord...)
  ;; Sample expressions for f-on-xcoord
  (define XCOORD1-VAL ...) ...
  ;; Tests using sample computations for f-on-xcoord
  (check-expect (f-on-xcoord XCOORD1 ...) XCOORD1-VAL) ...
  ;; Tests using sample values for f-on-xcoord
  (check-expect (f-on-xcoord ...) ...) ...
  ```
- ```
  ;; ycoord ... → ...     Purpose: ...
  (define (f-on-ycoord a-ycoord ...) ... a-ycoord...)
  ;; Sample expressions for f-on-ycoord
  (define YCOORD1-VAL ...) ...
  ;; Tests using sample computations for f-on-ycoord
  (check-expect  (f-on-ycoord YCORD1 ...) YCOORD1-VAL) ...
  ;; Tests using sample values for f-on-ycoord
  (check-expect (f-on-ycoord ...) ...) ...
  ```

Structures

The posn Structure

- ```
  ;; 2D-point ... → ...     Purpose: ...
  (define (f-on-2D-point a-2Dpoint ...)
    (f-on-xcoord (posn-x a-2Dpoint))...
    (f-on-ycoord (posn-y a-2Dpoint))...)

  ;; Sample expressions for f-on-2D-point
  (define 2D-POINT1-VAL ...)   ...

  ;; Tests using sample computations for f-on-2D-point
  (check-expect (f-on-2D-point 2D-POINT1 ...) 2D-POINT1-VAL) ...

  ;; Tests using sample values for f-on-2D-2D-point
  (check-expect (f-on-2D-point ...) ...)   ...
  ```
  ```

Structures

The posn Structure

- The 2D-point program refined to:

```
;; Sample instances of xcoord
(define X1 3.2)
(define X2 -8)

;; Sample instances of ycoord
(define Y1 9)
(define Y2 9)

;; Sample instances of 2D-point
(define A-POSN  (make-posn X1 Y1))
(define A-POSN2 (make-posn X2 Y2))

(check-expect (posn-x A-POSN)  X1)
(check-expect (posn-x A-POSN2) X2)
(check-expect (posn-y A-POSN)  Y1)
(check-expect (posn-y A-POSN2) Y2)
(check-expect (= (posn-x A-POSN) (posn-x A-POSN2)) #false)
(check-expect (= (posn-y A-POSN) (posn-y A-POSN2)) #true)
```

Structures

The posn Structure

- Design a program to determine if a `2D-point` is on the graph of `f(x) = x`^{3}

Structures

The posn Structure

- Design a program to determine if a `2D-point` is on the graph of `f(x) = x`^{3}
- Problem analysis: `(x, y)` is on the graph of `f(x) = x`^{3} if `y = x`^{3}

Structures

The posn Structure

- Design a program to determine if a 2D-point is on the graph of `f(x) = x`^{3}
- Problem analysis: `(x, y)` is on the graph of `f(x) = x`^{3} if `y = x`^{3}
- ```
  (define X1 3)   (define X2 -2.5)
  (define Y1 27)  (define Y2 39)
  (define 2D-POINT1 (make-posn X1 Y1))
  (define 2D-POINT2 (make-posn X2 Y2))
  ```
  ```

Structures

The posn Structure

- Design a program to determine if a 2D-point is on the graph of f(x) = x^3
- Problem analysis: (x, y) is on the graph of f(x) = x^3 if y = x^3
- ```
  (define X1 3)   (define X2 -2.5)
  (define Y1 27)  (define Y2 39)
  (define 2D-POINT1 (make-posn X1 Y1))
  (define 2D-POINT2 (make-posn X2 Y2))
  ```
- ```
  ;; Sample expressions for on-x^3?
  (define 2D-POINT1-VAL (= (cube (posn-x 2D-POINT1))
                           (posn-y 2D-POINT1)))
  (define 2D-POINT2-VAL (= (cube (posn-x 2D-POINT2))
                           (posn-y 2D-POINT2)))
  ```

Structures

The posn Structure

- Design a program to determine if a 2D-point is on the graph of `f(x) =` x^3
- Problem analysis: `(x, y)` is on the graph of `f(x) =` x^3 if `y =` x^3
- ```
  (define X1 3)   (define X2 -2.5)
  (define Y1 27)  (define Y2 39)
  (define 2D-POINT1 (make-posn X1 Y1))
  (define 2D-POINT2 (make-posn X2 Y2))
  ```
- ```
  ;; 2D-point ... → Boolean
  ;; Purpose: Determine if the given 2D-point is on the
  ;;          graph of f(x) = x^3
  (define (on-x^3? a-2Dpoint)
  ```
- ```
  ;; Sample expressions for on-x^3?
  (define 2D-POINT1-VAL (= (cube (posn-x 2D-POINT1))
                           (posn-y 2D-POINT1)))
  (define 2D-POINT2-VAL (= (cube (posn-x 2D-POINT2))
                           (posn-y 2D-POINT2)))
  ```
  ```

Structures

The posn Structure

- Design a program to determine if a 2D-point is on the graph of f(x) = x^3
- Problem analysis: (x, y) is on the graph of f(x) = x^3 if y = x^3
-
```
(define X1 3)   (define X2 -2.5)
(define Y1 27)  (define Y2 39)
(define 2D-POINT1 (make-posn X1 Y1))
(define 2D-POINT2 (make-posn X2 Y2))
```
-
```
;; 2D-point ... → Boolean
;; Purpose: Determine if the given 2D-point is on the
;;          graph of f(x) = x^3
(define (on-x^3? a-2Dpoint)
```
-
```
;; Sample expressions for on-x^3?
(define 2D-POINT1-VAL (= (cube (posn-x 2D-POINT1))
                         (posn-y 2D-POINT1)))
(define 2D-POINT2-VAL (= (cube (posn-x 2D-POINT2))
                         (posn-y 2D-POINT2)))
```
-
```
;; Tests using sample computations for on-x^3?
(check-expect (on-x^3? 2D-POINT1) 2D-POINT-VAL1)
(check-expect (on-x^3? 2D-POINT2) 2D-POINT-VAL2)
;; Tests using sample values for on-x^3?
(check-expect (on-x^3? (make-posn 0 0)) #true)
(check-expect (on-x^3? (make-posn -3.3 -35.937)) #true)
(check-expect (on-x^3? (make-posn 5 25)) #false)
```

Structures

The posn Structure

- Design a program to determine if a 2D-point is on the graph of `f(x) = x`^{3}
- Problem analysis: `(x, y)` is on the graph of `f(x) = x`^{3} if `y = x`^{3}
-
```
(define X1 3)   (define X2 -2.5)
(define Y1 27)  (define Y2 39)
(define 2D-POINT1 (make-posn X1 Y1))
(define 2D-POINT2 (make-posn X2 Y2))
```
-
```
;; 2D-point ... → Boolean
;; Purpose: Determine if the given 2D-point is on the
;;          graph of f(x) = x^3
(define (on-x^3? a-2Dpoint)
```
-
```
  (= (cube (posn-x a2D-point)) (posn-y a-2Dpoint)))
```
-
```
;; Sample expressions for on-x^3?
(define 2D-POINT1-VAL (= (cube (posn-x 2D-POINT1))
                         (posn-y 2D-POINT1)))
(define 2D-POINT2-VAL (= (cube (posn-x 2D-POINT2))
                         (posn-y 2D-POINT2)))
```
-
```
;; Tests using sample computations for on-x^3?
(check-expect (on-x^3? 2D-POINT1) 2D-POINT-VAL1)
(check-expect (on-x^3? 2D-POINT2) 2D-POINT-VAL2)
;; Tests using sample values for on-x^3?
(check-expect (on-x^3? (make-posn 0 0)) #true)
(check-expect (on-x^3? (make-posn -3.3 -35.937)) #true)
(check-expect (on-x^3? (make-posn 5 25)) #false)
```

Structures

The posn Structure

- The function cube is not part of BSL

Structures

The posn Structure

- The function cube is not part of BSL
-
```
;; xcoord → xcoord
;; Purpose: To cube the given xcoord
(define (cube an-xcoord) (expt an-xcoord 3))

;; Sample expressions for f-on-xcoord
(define XCOORD-VAL1 (expt -2 3))
(define XCOORD-VAL2 (expt 8 3))
(define XCOORD-VAL3 (expt 3 3))

;; Tests using sample computations for f-on-xcoord
(check-expect (cube -2)  XCOORD-VAL1)
(check-expect (cube  8)  XCOORD-VAL2)
(check-expect (cube  X1) XCOORD-VAL3)

;; Tests using sample values for f-on-xcoord
(check-expect (cube 0) 0)
(check-within (cube X2) -15.62 0.01)
```

Structures

The posn Structure

- Refined Design Recipe

  Data Design:

  1. Outline data representation and create data definitions.
  2. Create sample instances for each data definition.
  3. Create a function template for each data definition.

  For a needed function:

  4. Outline the computation.
  5. Define constants for the value of sample expressions.
  6. Identify and name the differences among the sample expressions.
  7. Write the function's signature and purpose.
  8. Write the function's header.
  9. Write tests.
  10. Write the function's body.
  11. Run the tests and, if necessary, redesign.

Structures

Revisiting `in-Q1?`

- Redevelop predicate to determine if an integer-based point is in Q1

Structures

Revisiting in-Q1?

- Redevelop predicate to determine if an integer-based point is in Q1
- ```
  #| ;; An x-coordinate (ix) and a y-coordinate (iy) are integers
     ;; A 2D-ipoint is a structure (make-posn ix iy)
  ```
  ```

Structures

Revisiting `in-Q1?`

- Redevelop predicate to determine if an integer-based point is in Q1
- ```
  #| ;; An x-coordinate (ix) and a y-coordinate (iy) are integers
     ;; A 2D-ipoint is a structure (make-posn ix iy)
  ```
- ```
  ;; (define IX1 ...) ...  (define IY1 ...)
  ;; (define 2D-IPOINT1 (make-posn ... ...))...
  ;; ix ... → ...     Purpose: ...
  (define (f-on-ix an-ix ...) ... an-ix...)
  ;; Sample expressions for f-on-ix
  (define IX-VAL ...)
  ;; Tests using sample computations for f-on-ix
  (check-expect (f-on-ix ...) IX-VAL)   ...
  ;; Tests using sample values for f-on-ix
  (check-expect (f-on-ix ...) ...)   ...
  ```

Structures

Revisiting in-Q1?

- Redevelop predicate to determine if an integer-based point is in Q1
- ```
  #| ;; An x-coordinate (ix) and a y-coordinate (iy) are integers
     ;; A 2D-ipoint is a structure (make-posn ix iy)
  ```
- ```
  ;; (define IX1 ...) ...  (define IY1 ...)
  ;; (define 2D-IPOINT1 (make-posn ... ...))...
  ;; ix ... → ...     Purpose: ...
  (define (f-on-ix an-ix ...) ... an-ix...)
  ;; Sample expressions for f-on-ix
  (define IX-VAL ...)
  ;; Tests using sample computations for f-on-ix
  (check-expect (f-on-ix ...) IX-VAL)   ...
  ;; Tests using sample values for f-on-ix
  (check-expect (f-on-ix ...) ...)   ...
  ```
- ```
  ;; iy ... → ...     Purpose: ...
  (define (f-on-iy a-iy ...) ... a-iy...)
  ;; Sample expressions for f-on-iy
  (define IY-VAL ...)   ...
  ;; Tests using sample computations for f-on-iy
  (check-expect (f-on-iy ...) IY-VAL)   ...
  ;; Tests using sample values for f-on-iy
  (check-expect (f-on-iy ...) ...)   ...
  ```
  ```

Structures

The posn Structure

- ```
  ;; 2D-ipoint ... → ...     Purpose: ...
  (define (f-on-2D-ipoint a-2Dipoint ...)
     ...(f-on-ix (posn-x a-2Dipoint))...
     ...(f-on-iy (posn-y a-2Dipoint))...)

  ;; Sample expressions for f-on-2D-ipoint
  (define 2D-IPOINT-VAL ...)   ...

  ;; Tests using sample computations for f-on-2D-ipoint
  (check-expect (f-on-2D-ipoint ...) 2D-IPOINT-VAL)   ...

  ;; Tests using sample values for f-on-2D-ipoint
  (check-expect (f-on-2D-ipoint ...) ...) ...                    |#
  ```
  ```

Structures

The posn Structure

- ```
  ;; Sample instances of ix, iy, and 2D-ipoint
  (define IX1 3)       (define IX2 -5)
  (define IY1 11)      (define IY2 23)
  (define 2D-IPOINT1 (make-posn IX1 IY1))
  (define 2D-IPOINT2 (make-posn IX2 IY2))
  ```
  ```

Structures

The posn Structure

- ```
  ;; Sample instances of ix, iy, and 2D-ipoint
  (define IX1 3)       (define IX2 -5)
  (define IY1 11)      (define IY2 23)
  (define 2D-IPOINT1 (make-posn IX1 IY1))
  (define 2D-IPOINT2 (make-posn IX2 IY2))
  ```

- ```
  ;; Sample expressions for in-Q1?
  (define 2D-IPOINT1-VAL (and (ix-in-Q1? (posn-x 2D-IPOINT1))
                              (iy-in-Q1? (posn-y 2D-IPOINT1))))
  (define 2D-IPOINT2-VAL (and (ix-in-Q1? (posn-x 2D-IPOINT2))
                              (iy-in-Q1? (posn-y 2D-IPOINT2))))
  ```

Structures

The posn Structure

- ```
  ;; Sample instances of ix, iy, and 2D-ipoint
  (define IX1 3)       (define IX2 -5)
  (define IY1 11)      (define IY2 23)
  (define 2D-IPOINT1 (make-posn IX1 IY1))
  (define 2D-IPOINT2 (make-posn IX2 IY2))
  ```
- ```
  ;; 2D-ipoint → Boolean
  ;; Purpose: Determine if 2D-ipoint is in Q1
  (define (in-Q1? a-2Dipoint)
  ```
- ```
  ;; Sample expressions for in-Q1?
  (define 2D-IPOINT1-VAL (and (ix-in-Q1? (posn-x 2D-IPOINT1))
                              (iy-in-Q1? (posn-y 2D-IPOINT1))))
  (define 2D-IPOINT2-VAL (and (ix-in-Q1? (posn-x 2D-IPOINT2))
                              (iy-in-Q1? (posn-y 2D-IPOINT2))))
  ```
  ```

Structures

The posn Structure

- ```
  ;; Sample instances of ix, iy, and 2D-ipoint
  (define IX1 3)        (define IX2 -5)
  (define IY1 11)       (define IY2 23)
  (define 2D-IPOINT1 (make-posn IX1 IY1))
  (define 2D-IPOINT2 (make-posn IX2 IY2))
  ```
- ```
  ;; 2D-ipoint → Boolean
  ;; Purpose: Determine if 2D-ipoint is in Q1
  (define (in-Q1? a-2Dipoint)
  ```
- ```
  ;; Sample expressions for in-Q1?
  (define 2D-IPOINT1-VAL (and (ix-in-Q1? (posn-x 2D-IPOINT1))
                              (iy-in-Q1? (posn-y 2D-IPOINT1))))
  (define 2D-IPOINT2-VAL (and (ix-in-Q1? (posn-x 2D-IPOINT2))
                              (iy-in-Q1? (posn-y 2D-IPOINT2))))
  ```
- ```
  ;; Tests using sample computations for in-Q1?
  (check-expect (in-Q1? 2D-IPOINT1) 2D-IPOINT1-VAL)
  (check-expect (in-Q1? 2D-IPOINT2) 2D-IPOINT2-VAL)
  ;; Tests using sample values for in-Q1?
  (check-expect (in-Q1? (make-posn -3 -7))   #false)
  (check-expect (in-Q1? (make-posn 83 -4))   #false)
  (check-expect (in-Q1? (make-posn 100 864)) #true)
  ```

Structures

The posn Structure

- ```
  ;; Sample instances of ix, iy, and 2D-ipoint
  (define IX1 3)       (define IX2 -5)
  (define IY1 11)      (define IY2 23)
  (define 2D-IPOINT1 (make-posn IX1 IY1))
  (define 2D-IPOINT2 (make-posn IX2 IY2))
  ```
- ```
  ;; 2D-ipoint → Boolean
  ;; Purpose: Determine if 2D-ipoint is in Q1
  (define (in-Q1? a-2Dipoint)
  ```
- ```
    (and (ix-in-Q1? (posn-x a-2Dipoint))
         (iy-in-Q1? (posn-y a-2Dipoint))))
  ```
- ```
  ;; Sample expressions for in-Q1?
  (define 2D-IPOINT1-VAL (and (ix-in-Q1? (posn-x 2D-IPOINT1))
                              (iy-in-Q1? (posn-y 2D-IPOINT1))))
  (define 2D-IPOINT2-VAL (and (ix-in-Q1? (posn-x 2D-IPOINT2))
                              (iy-in-Q1? (posn-y 2D-IPOINT2))))
  ```
- ```
  ;; Tests using sample computations for in-Q1?
  (check-expect (in-Q1? 2D-IPOINT1) 2D-IPOINT1-VAL)
  (check-expect (in-Q1? 2D-IPOINT2) 2D-IPOINT2-VAL)
  ;; Tests using sample values for in-Q1?
  (check-expect (in-Q1? (make-posn -3 -7))   #false)
  (check-expect (in-Q1? (make-posn 83 -4))   #false)
  (check-expect (in-Q1? (make-posn 100 864)) #true)
  ```
  ```

Structures

The posn Structure

- ```
  ;; ix → Boolean
  ;; Purpose: To determine if the given ix is in the Q1 range
  (define (ix-in-Q1? an-ix)
    (> an-ix 0))

  ;; Sample expressions for ix-in-Q1?
  (define IX1-VAL (> IX1 0))
  (define IX2-VAL (> IX2 0))

  ;; Tests using sample computations for ix-in-Q1?
  (check-expect (ix-in-Q1? IX1) IX1-VAL)
  (check-expect (ix-in-Q1? IX2) IX2-VAL)

  ;; Tests using sample values for ix-in-Q1?
  (check-expect (ix-in-Q1? 0) #false)
  (check-expect (ix-in-Q1? 19380) #true)
  ```
  ```

Structures

The posn Structure

- ```
  ;; iy → Boolean
  ;; Purpose: To determine if the given iy is in the Q1 range
  (define (iy-in-Q1? an-iy)
    (> an-iy 0))

  ;; Sample expressions for iy-in-Q1?
  (define IY1-VAL (> IY1 0))
  (define IY2-VAL (> IY2 0))

  ;; Tests using sample computations for iy-in-Q1?
  (check-expect (iy-in-Q1? IY1) IY1-VAL)
  (check-expect (iy-in-Q1? IY2) IY2-VAL)

  ;; Tests using sample values for iy-in-Q1?
  (check-expect (iy-in-Q1? 0) #false)
  (check-expect (iy-in-Q1? 19380) #true)
  ```
  ```

Structures

Homework

- Problems: 79–82

Defining Structures

- It turns out that rarely is the input to a function, for example, a single string or number
- It is far more common for there to be several pieces of related data that need to be processed like a `2D-point`
- What needs to be done if data has more than two (but finite) varying characteristics?

Defining Structures

- It turns out that rarely is the input to a function, for example, a single string or number
- It is far more common for there to be several pieces of related data that need to be processed like a `2D-point`
- What needs to be done if data has more than two (but finite) varying characteristics?
- Consider representing a student that has a first name, a middle name, a last name and a grade point average
- How can a student be represented?

Defining Structures

- It turns out that rarely is the input to a function, for example, a single string or number
- It is far more common for there to be several pieces of related data that need to be processed like a `2D-point`
- What needs to be done if data has more than two (but finite) varying characteristics?
- Consider representing a student that has a first name, a middle name, a last name and a grade point average
- How can a student be represented?
- Need a structure that has four fields
- `BSL` gives programmers the ability to define their own structures
- A programmer can define finite compound data of any size and provide custom names to the structure and its fields

Defining Structures

Structure Definitions

- BSL grammar:

  program ::= {expr | test | def | defs}*

  defs ::= (define-struct <structure name> (<field name>*))

Defining Structures

Structure Definitions

- BSL grammar:

  `program` ::= {`expr` | `test` | `def` | `defs`}*

  `defs` ::= (define-struct <structure name> (<field name>*))

- A structure definition creates much more than just a structure
- Also creates the constructor and selector functions for the structure

Defining Structures

Structure Definitions

- BSL grammar:

  `program` ::= {`expr` | `test` | `def` | `defs`}*

  `defs` ::= (define-struct <structure name> (<field name>*))

- A structure definition creates much more than just a structure
- Also creates the constructor and selector functions for the structure
- Naming conventions:

  `make-<structure name>`   `<structure name>-<field name>`

Defining Structures

Structure Definitions

- BSL grammar:

```
program ::= {expr | test | def | defs}*
   defs ::= (define-struct <structure name> (<field name>*))
```

- A structure definition creates much more than just a structure
- Also creates the constructor and selector functions for the structure
- Naming conventions:

```
make-<structure name>    <structure name>-<field name>
```

- We can now define a structure for a 2D-point as follows:

```
(define-struct 2Dpoint (xval yval))
```

Defining Structures

Structure Definitions

- BSL grammar:

```
program ::= {expr | test | def | defs}*
   defs ::= (define-struct <structure name> (<field name>*))
```

- A structure definition creates much more than just a structure
- Also creates the constructor and selector functions for the structure
- Naming conventions:

```
make-<structure name>    <structure name>-<field name>
```

- We can now define a structure for a 2D-point as follows:

```
(define-struct 2Dpoint (xval yval))
```

- Creates the following functions to manipulate 2D-points:

```
;; X Y → 2Dpoint Purpose: Create a 2Dpoint
(define (make-2Dpoint an-X an-Y) ...)

;; 2Dpoint → X  Purpose: Return xval of given 2Dpoint
(define (2Dpoint-xval a-2Dpoint) ...)

;; 2Dpoint → Y  Purpose: Return yval of given 2Dpoint
(define (2Dpoint-yval a-2Dpoint) ...)

;; Any → Boolean  Purpose: Determine if a 2Dpoint
(define (2Dpoint? any-value) ...)
```

- Do not worry about how these functions are implemented in BSL
- Important to understand the structure API provided by BSL

Defining Structures

Structure Definitions

- ```
  ;; X Y → 2Dpoint
  ;; Purpose: To create a 2Dpoint with the given values
  (define (make-2Dpoint an-X an-Y) ...)
  ```
- The sigatures refer to indeterminate types called `X` and `Y`
- `X` and `Y` are *type variables*
- Represent a defined type that exists in `BSL` or that has been defined by a programmer
  ```

Defining Structures

Structure Definitions

- ```
  ;; X Y → 2Dpoint
  ;; Purpose: To create a 2Dpoint with the given values
  (define (make-2Dpoint an-X an-Y) ...)
  ```
- The sigatures refer to indeterminate types called `X` and `Y`
- `X` and `Y` are *type variables*
- Represent a defined type that exists in `BSL` or that has been defined by a programmer
- Type variables indicate that a structure is *generic*
- This means that it works equally well for many different data types
- Must be used with care:

  ```
  (define A (make-2Dpoint 34 90))

  (define B (make-2Dpoint "Hello" "World!"))

  (define C (make-2Dpoint #true 'CS))
  ```
  ```

Defining Structures

Structure Definitions

- How does a programmer know the correct use of the constructor?

Defining Structures

Structure Definitions

- How does a programmer know the correct use of the constructor?
- Without a data definition it is impossible to know, because `X` and `Y` may represent any type

Defining Structures

Structure Definitions

- How does a programmer know the correct use of the constructor?
- Without a data definition it is impossible to know, because `X` and `Y` may represent any type
- `2D-point`:

  `A 2Dpoint is a structure (make-2Dpoint` $\mathbb{R}$ $\mathbb{R}$`)`

Defining Structures

Structure Definitions

- How does a programmer know the correct use of the constructor?
- Without a data definition it is impossible to know, because `X` and `Y` may represent any type
- `2D-point`:

  A 2Dpoint is a structure (make-2Dpoint $\mathbb{R}$ $\mathbb{R}$)
- The template for `2D-point` is:

```
 ;; Sample Instances of 2Dpoint
(define P1 (make-2Dpoint ...))

 ;; 2Dpoint ... → ...
 ;; Purpose: ...
(define (f-on-2Dpoint a-2dpoint ...)
  ...(2Dpoint-xval a-2dpoint)...(2Dpoint-yval a-2dpoint)...)

;; Sample expressions for f-on-2Dpoint
(define 2DP-VAL1 ...(2Dpoint-xval P1)...(2Dpoint-xval P1)...)
     ...
;; Tests using sample computations for f-on-2Dpoint
(check-within (f-on-2Dpoint P1 ...) 2DP-VAL1)
     ...
;; Tests using sample values for f-on-2Dpoint
(check-within (f-on-2Dpoint ...) ...)
     ...
```

Defining Structures

Structure Definitions

- Write a function to compute the distance of a `2D-point` from the origin

Defining Structures

Structure Definitions

- Write a function to compute the distance of a `2D-point` from the origin
- The distance formula may be simplified as follows:

$$
\begin{aligned}
\text{distance}((x_1, y_1), (x_2, y_2)) &= \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \\
\text{distance}((0, 0), (x_2, y_2)) &= \sqrt{(x_2 - 0)^2 + (y_2 - 0)^2} \\
&= \sqrt{x_2^2 + y_2^2}
\end{aligned}
$$

Defining Structures

Structure Definitions

- ```
  ;; A 2Dpoint is a structure (make-2Dpoint ℝ ℝ)
  (define-struct 2Dpoint (xval yval))
  ```
  ```

Defining Structures

Structure Definitions

- ```
  ;; A 2Dpoint is a structure (make-2Dpoint ℝ ℝ)
  (define-struct 2Dpoint (xval yval))
  ```
- ```
  ;; Sample instances of 2Dpoint
  (define P1 (make-2Dpoint 5 10))
  (define P2 (make-2Dpoint 25 11))
  ```

Defining Structures

Structure Definitions

- ```
  ;; A 2Dpoint is a structure (make-2Dpoint ℝ ℝ)
  (define-struct 2Dpoint (xval yval))
  ```
- ```
  ;; Sample instances of 2Dpoint
  (define P1 (make-2Dpoint 5 10))
  (define P2 (make-2Dpoint 25 11))
  ```
- ```
  ;; Sample expressions for distance-to-origin
  (define 2DP-VAL1 (sqrt (+ (sqr (2Dpoint-xval P1))
                            (sqr (2Dpoint-yval P1)))))Textbook typo
  (define 2DP-VAL2 (sqrt (+ (sqr (2Dpoint-xval P2))
                            (sqr (2Dpoint-yval P2)))))Textbook typo
  ```
  ```

Defining Structures

Structure Definitions

- ```
  ;; A 2Dpoint is a structure (make-2Dpoint ℝ ℝ)
  (define-struct 2Dpoint (xval yval))
  ```
- ```
  ;; Sample instances of 2Dpoint
  (define P1 (make-2Dpoint 5 10))
  (define P2 (make-2Dpoint 25 11))
  ```
- ```
  ;; 2Dpoint → real-number
  ;; Purpose: To compute the distance to the origin for the
  ;;          given 2Dpoint
  (define (distance-to-origin a-2dpoint)
  ```
- ```
  ;; Sample expressions for distance-to-origin
  (define 2DP-VAL1 (sqrt (+ (sqr (2Dpoint-xval P1))
                            (sqr (2Dpoint-yval P1))))))Textbook typo
  (define 2DP-VAL2 (sqrt (+ (sqr (2Dpoint-xval P2))
                            (sqr (2Dpoint-yval P2))))))Textbook typo
  ```

Defining Structures

Structure Definitions

- ```
  ;; A 2Dpoint is a structure (make-2Dpoint ℝ ℝ)
  (define-struct 2Dpoint (xval yval))
  ```
- ```
  ;; Sample instances of 2Dpoint
  (define P1 (make-2Dpoint 5 10))
  (define P2 (make-2Dpoint 25 11))
  ```
- ```
  ;; 2Dpoint → real-number
  ;; Purpose: To compute the distance to the origin for the
  ;;          given 2Dpoint
  (define (distance-to-origin a-2dpoint)
  ```
- ```
  ;; Sample expressions for distance-to-origin
  (define 2DP-VAL1 (sqrt (+ (sqr (2Dpoint-xval P1))
                            (sqr (2Dpoint-yval P1)))))Textbook typo
  (define 2DP-VAL2 (sqrt (+ (sqr (2Dpoint-xval P2))
                            (sqr (2Dpoint-yval P2)))))Textbook typo
  ```
- ```
  ;; Tests using sample computations for distance-to-origin
  (check-within (distance-to-origin P1) 2DP-VAL1 0.01)
  (check-within (distance-to-origin P2) 2DP-VAL2 0.01)
  ;; Tests using sample values for distance-to-origin
  (check-within (distance-to-origin (make-2Dpoint 0 0)) 0 0.01)
  (check-within (distance-to-origin (make-2Dpoint -3 -4)) 5 0.01)
  ```
  ```

Defining Structures

Structure Definitions

- ```
  ;; A 2Dpoint is a structure (make-2Dpoint ℝ ℝ)
  (define-struct 2Dpoint (xval yval))
  ```
- ```
  ;; Sample instances of 2Dpoint
  (define P1 (make-2Dpoint 5 10))
  (define P2 (make-2Dpoint 25 11))
  ```
- ```
  ;; 2Dpoint → real-number
  ;; Purpose: To compute the distance to the origin for the
  ;;          given 2Dpoint
  (define (distance-to-origin a-2dpoint)
  ```
- ```
    (sqrt (+ (sqr (2Dpoint-xval a-2dpoint))
             (sqr (2Dpoint-yval a-2dpoint)))))
  ```
- ```
  ;; Sample expressions for distance-to-origin
  (define 2DP-VAL1 (sqrt (+ (sqr (2Dpoint-xval P1))
                            (sqr (2Dpoint-yval P1)))))Textbook typo
  (define 2DP-VAL2 (sqrt (+ (sqr (2Dpoint-xval P2))
                            (sqr (2Dpoint-yval P2)))))Textbook typo
  ```
- ```
  ;; Tests using sample computations for distance-to-origin
  (check-within (distance-to-origin P1) 2DP-VAL1 0.01)
  (check-within (distance-to-origin P2) 2DP-VAL2 0.01)
  ;; Tests using sample values for distance-to-origin
  (check-within (distance-to-origin (make-2Dpoint 0 0)) 0 0.01)
  (check-within (distance-to-origin (make-2Dpoint -3 -4)) 5 0.01)
  ```

Defining Structures

Computing Structures

- Just like numbers, Booleans, strings, and `posn`s, the structures a programmer defines are *first class*
- First class means that they may be passed as input to functions and may be returned as a function value
- Functions may compute instances of a structure

Defining Structures

Computing Structures

- Just like numbers, Booleans, strings, and `posn`s, the structures a programmer defines are *first class*
- First class means that they may be passed as input to functions and may be returned as a function value
- Functions may compute instances of a structure
- Consider the problem of updating a student's grade point average
- A student has a first name, a middle name, a last name and a grade point average

Defining Structures

Computing Structures

- ```
  #| DATA DEFINITIONS
     A GPA is a real number in [0..4].

     ;; Sample instances for GPA
     (define GPA1 ...)
          ...
  ```
  ```

Defining Structures

Computing Structures

- ```
  #| DATA DEFINITIONS
     A GPA is a real number in [0..4].

     ;; Sample instances for GPA
     (define GPA1 ...)
          ...
  ```

- ```
     A student is a structure
         (make-student string string string GPA) Typo: Real --> GPA
     that contains a first name, a middle name, a last name,
     and a grade point average.

     ;; Sample instances for student
     (define STUD1 (make-student ... ... ... ...)
          ...
  |#

  (define-struct student (fn mn ln gpa))
  ```

Defining Structures

Computing Structures

- ```
  #| FUNCTION TEMPLATES
  ;; GPA ... → ...  Purpose: ...
  (define (f-on-GPA a-gpa ...) ...)
  ;; Sample expressions for f-on-GPA
  (define GPA1-VAL ...) ...
  ;; Tests using sample computations for f-on-GPA
  (check-expect (f-on-GPA GPA1 ...) GPA1-VAL) ...
  ;; Tests using sample values for f-on-GPA
  (check-expect (f-on-student ...) ...) ...
  ```
  ```

Defining Structures

Computing Structures

- ```
  #| FUNCTION TEMPLATES
  ;; GPA ... → ...  Purpose: ...
  (define (f-on-GPA a-gpa ...) ...)
  ;; Sample expressions for f-on-GPA
  (define GPA1-VAL ...) ...
  ;; Tests using sample computations for f-on-GPA
  (check-expect (f-on-GPA GPA1 ...) GPA1-VAL) ...
  ;; Tests using sample values for f-on-GPA
  (check-expect (f-on-student ...) ...) ...
  ```

- ```
  ;; student ... → ...  Purpose: ...
  (define (f-on-student a-student ...)
    ...(f-on-string (student-fn  a-student))...
    ...(f-on-string (student-mn  a-student))...
    ...(f-on-string (student-ln  a-student))...
    ...(f-on-GPA(student-gpa a-student))...)
  ;; Sample expressions for f-on-student
  (define STUD1-VAL ...) ...
  ;; Tests using sample computations for f-on-student
  (check-expect (f-on-student STUD1 ...) STUD1-VAL) ...
  ;; Tests using sample values for f-on-student
  (check-expect (f-on-student (make-student ... ... ... ...))...)  ...|#
  ```

Defining Structures

Computing Structures

- To start, define several instances of GPA and of student:

```
(define OLDGPA1 3.8)
(define OLDGPA2 3.7)
(define OLDGPA3 3.58)

(define NEWGPA1 3.93)
(define NEWGPA1 3.9)
(define NEWGPA1 3.8)

(define STUD1 (make-student "Barbara" "" "Mucha" OLDGPA1))
(define STUD2 (make-student "Joan"   "Elizabeth"
                            "Feeney" OLDGPA2))
(define STUD3 (make-student "Christopher" "Michael"
                            "Dutra"       OLDGPA3))
```

Defining Structures

Computing Structures

- ```
  ;; Sample expressions for update-student-gpa
  (define STUD1-VAL (make-student (student-fn STUD1)
                                  (student-mn STUD1)
                                  (student-ln STUD1)
                                  NEWGPA1))

  (define STUD2-VAL (make-student (student-fn STUD2)
                                  (student-mn STUD2)
                                  (student-ln STUD2)
                                  NEWGPA2))

  (define STUD3-VAL (make-student (student-fn STUD3)
                                  (student-mn STUD3)
                                  (student-ln STUD3)
                                  NEWGPA3))
  ```
  ```

Defining Structures

Computing Structures

- ```
  ;; student GPA → student  Typo: Real --> GPA
  ;; Purpose: Update the given student's gpa
  (define (update-student-gpa a-student a-gpa)
  ```

- ```
  ;; Sample expressions for update-student-gpa
  (define STUD1-VAL (make-student (student-fn STUD1)
                                  (student-mn STUD1)
                                  (student-ln STUD1)
                                  NEWGPA1))

  (define STUD2-VAL (make-student (student-fn STUD2)
                                  (student-mn STUD2)
                                  (student-ln STUD2)
                                  NEWGPA2))

  (define STUD3-VAL (make-student (student-fn STUD3)
                                  (student-mn STUD3)
                                  (student-ln STUD3)
                                  NEWGPA3))
  ```

Defining Structures

Computing Structures

- ```
  ;; student GPA → student  Typo: Real --> GPA
  ;; Purpose: Update the given student's gpa
  (define (update-student-gpa a-student a-gpa)
  ```
- ```
    (make-student (student-fn a-student)
                  (student-mn a-student)
                  (student-ln a-student)
                  a-gpa))
  ```
- ```
  ;; Sample expressions for update-student-gpa
  (define STUD1-VAL (make-student (student-fn STUD1)
                                  (student-mn STUD1)
                                  (student-ln STUD1)
                                  NEWGPA1))

  (define STUD2-VAL (make-student (student-fn STUD2)
                                  (student-mn STUD2)
                                  (student-ln STUD2)
                                  NEWGPA2))

  (define STUD3-VAL (make-student (student-fn STUD3)
                                  (student-mn STUD3)
                                  (student-ln STUD3)
                                  NEWGPA3))
  ```
  ```

Defining Structures

Structure Definitions

- ```
  ;; Tests using sample computations for update-student-gpa
  (check-expect (update-student-gpa STUD1 NEWGPA1)
                STUD1-VAL)
  (check-expect (update-student-gpa STUD2 NEWGPA2)
                STUD2-VAL)
  (check-expect (update-student-gpa STUD3 NEWGPA3)
                STUD3-VAL)

  ;; Tests using sample values for update-student-gpa
  (check-expect (update-student-gpa
                 (make-student "Sandy" "" "Marinakys" 3.1)
                 3.4)
                (make-student "Sandy" "" "Marinakys" 3.4))
  (check-expect (update-student-gpa
                 (make-student "Lidia" "Carolina" "Vazquez" 3.5)
                 3.5)
                (make-student "Lidia" "Carolina" "Vazquez" 3.5))
  (check-expect (update-student-gpa
                 (make-student "Luis" "Manuel" "Diaz" 3.7)
                 3.6)
                (make-student "Luis" "Manuel" "Diaz" 3.6))
  ```
  ```

Defining Structures

Structures for the Masses

- BSL is exceptionally well-equipped for structures, most people are not
- Everyone, nonetheless, may benefit from structures

Defining Structures

Structures for the Masses

- BSL is exceptionally well-equipped for structures, most people are not
- Everyone, nonetheless, may benefit from structures
- Challenge: transform data to something understood by the masses

Defining Structures

Structures for the Masses

- BSL is exceptionally well-equipped for structures, most people are not
- Everyone, nonetheless, may benefit from structures
- Challenge: transform data to something understood by the masses
- Consider:

```
(make-student "Barbara" "" "Mucha" 3.8)

(make-student "Joan" "Elizabeth" "Feeney" 3.7)
```

- Problem: most are not aware of our data definition of `student`

Defining Structures

Structures for the Masses

- Common technique: convert data to a string
- ```
  > (student2string STUD1)
  "Barbara Mucha has a 3.8 grade point average."
  > (student2string STUD2)
  "Joan E. Feeney has a 3.7 grade point average."
  ```
  ```

Defining Structures

Structures for the Masses

- Common technique: convert data to a string
- ```
  > (student2string STUD1)
  "Barbara Mucha has a 3.8 grade point average."
  > (student2string STUD2)
  "Joan E. Feeney has a 3.7 grade point average."
  ```
- The returned strings have the following components:
  1. The student's first name
  2. The student's middle name abbreviation if any
  3. The string " has a "
  4. The student's grade point average
  5. The string " grade point average."
  ```

Defining Structures

Structures for the Masses

- ```
  ;; Sample expressions for student2string
  (define STUD1-STR (string-append
                       (student-fn  STUD1)
                       (middle-name-abbrev (student-mn  STUD1))
                       (student-ln  STUD1)
                       " has a "
                       (gpa->string (student-gpa STUD1))
                       " grade point average."))
  (define STUD2-STR (string-append
                       (student-fn  STUD2)
                       (middle-name-abbrev (student-mn  STUD2))
                       (student-ln  STUD2)
                       " has a "
                       (gpa->string (student-gpa STUD2))
                       " grade point average."))
  ```
  ```

Defining Structures

Structures for the Masses

- ```
  ;; student → string  Purpose: Transform student to a string
  (define (student2string a-student)
  ```

- ```
  ;; Sample expressions for student2string
  (define STUD1-STR (string-append
                      (student-fn  STUD1)
                      (middle-name-abbrev (student-mn  STUD1))
                      (student-ln  STUD1)
                      " has a "
                      (gpa->string (student-gpa STUD1))
                      " grade point average."))
  (define STUD2-STR (string-append
                      (student-fn  STUD2)
                      (middle-name-abbrev (student-mn  STUD2))
                      (student-ln  STUD2)
                      " has a "
                      (gpa->string (student-gpa STUD2))
                      " grade point average."))
  ```

Defining Structures

Structures for the Masses

- ```
  ;; student → string  Purpose: Transform student to a string
  (define (student2string a-student)
  ```
- ```
    (string-append
      (student-fn  a-student)
      (middle-name-abbrev (student-mn  a-student))
      (student-ln  a-student)
      " has a "
      (gpa->string (student-gpa a-student))
      " grade point average."))
  ```
- ```
  ;; Sample expressions for student2string
  (define STUD1-STR (string-append
                      (student-fn  STUD1)
                      (middle-name-abbrev (student-mn  STUD1))
                      (student-ln  STUD1)
                      " has a "
                      (gpa->string (student-gpa STUD1))
                      " grade point average."))
  (define STUD2-STR (string-append
                      (student-fn  STUD2)
                      (middle-name-abbrev (student-mn  STUD2))
                      (student-ln  STUD2)
                      " has a "
                      (gpa->string (student-gpa STUD2))
                      " grade point average."))
  ```
  ```

Defining Structures

Structures for the Masses

- ```
  ;; Tests using sample computations for student2string
  (check-expect (student2string STUD1) STUD1-STR)
  (check-expect (student2string STUD2) STUD2-STR)

  ;; Tests using sample values for student2string
  (check-expect
    (student2string (make-student "Mercedes" "G." "Merayo" 3.97))
    "Mercedes G. Merayo has a 3.97 grade point average.")
  (check-expect
    (student2string (make-student "Manuel" "" "Núñez" 3.89))
    "Manuel Núñez has a 3.89 grade point average.")
  ```
  ```

Defining Structures

Structures for the Masses

- Let us continue with the task of designing the auxiliary function `middle-name-abbrev`
- Not everyone has a middle name

Defining Structures

Structures for the Masses

- Let us continue with the task of designing the auxiliary function `middle-name-abbrev`
- Not everyone has a middle name
-
```
    A middle name (mn) is either:
      1. ""
      2. not ""
```

Defining Structures

Structures for the Masses

- Let us continue with the task of designing the auxiliary function `middle-name-abbrev`
- Not everyone has a middle name
- ```
  A middle name (mn) is either:
    1. ""
    2. not ""
  ```
- Template

```
#|  ;; Sample instances for mn
    (define MN1 "")
    (define MN2 ...)   ...
    ;; mn ... --> ...   Purpose: ...
    (define (f-on-nm an-nm ...)
      (if (string=? an-nm "")
          ...
          ...))
    ;; Sample expressions for f-on-mn
    (define MN1-VAL ...)
    (define MN2-VAL ...) ...
    ;; Tests using sample computations for f-on-mn
    (check-expect (f-on-mn MN1 ...) MN1-VAL)
    (check-expect (f-on-mn MN2 ...) MN2-VAL) ...
    ;; Tests using sample values for f-on-mn
    (check-expect (f-on-mn ...) ...) ...   |#
```
```

Defining Structures

Structures for the Masses

- Must update student data definition:

  A student is a structure, (make-student string mn string GPA), that contains a first name, a middle name, a last name, and a grade point average.

```
;; student ... → ...
;; Purpose: ...
(define (f-on-student a-student ...)
  ...(f-on-string (student-fn  a-student))...
  ...(f-on-mn (student-mn  a-student))...
  ...(f-on-string (student-ln  a-student))...
  ...(f-on-GPA (student-gpa a-student))...)
```

Defining Structures

Structures for the Masses

- Proceed with the design of `middle-name-abbrev`:

```
;; Sample instances of mn
(define MN1 "")  (define MN2 "Jose")  (define MN3 "Francisco")
```

Defining Structures

Structures for the Masses

- Proceed with the design of middle-name-abbrev:

```
 ;; Sample instances of mn
(define MN1 "")  (define MN2 "Jose")  (define MN3 "Francisco")
```

- ```
  ;; Sample expressions for middle-name-abbrev
  (define MN1-VAL " ")
  (define MN2-VAL (string-append " " (substring MN2 0 1) ". "))
  (define MN3-VAL (string-append " " (substring MN3 0 1) ". "))
  ```
  ```

Defining Structures

Structures for the Masses

- Proceed with the design of middle-name-abbrev:

```
 ;; Sample instances of mn
(define MN1 "")  (define MN2 "Jose")  (define MN3 "Francisco")
```

- ```
  ;; mn → string   Purpose: Abbreviate given name
  (define (middle-name-abbrev a-mn)
  ```

- ```
  ;; Sample expressions for middle-name-abbrev
  (define MN1-VAL " ")
  (define MN2-VAL (string-append " " (substring MN2 0 1) ". "))
  (define MN3-VAL (string-append " " (substring MN3 0 1) ". "))
  ```

Defining Structures

Structures for the Masses

- Proceed with the design of middle-name-abbrev:

```
 ;; Sample instances of mn
(define MN1 "")  (define MN2 "Jose")  (define MN3 "Francisco")
```

- ```
  ;; mn → string   Purpose: Abbreviate given name
  (define (middle-name-abbrev a-mn)
  ```

- ```
  ;; Sample expressions for middle-name-abbrev
  (define MN1-VAL " ")
  (define MN2-VAL (string-append " " (substring MN2 0 1) ". "))
  (define MN3-VAL (string-append " " (substring MN3 0 1) ". "))
  ```

- ```
  ;; Tests using sample computations for middle-name-abbrev
  (check-expect (middle-name-abbrev MN1) MN1-VAL)
  (check-expect (middle-name-abbrev MN2) MN2-VAL)
  (check-expect (middle-name-abbrev MN3) MN3-VAL)
  ;; Tests using sample values for middle-name-abbrev
  (check-expect (middle-name-abbrev "Kaliman") " K. ")
  ```
  ```

Defining Structures

Structures for the Masses

- Proceed with the design of middle-name-abbrev:

```
 ;; Sample instances of mn
(define MN1 "")  (define MN2 "Jose")  (define MN3 "Francisco")
```

- ```
  ;; mn → string   Purpose: Abbreviate given name
  (define (middle-name-abbrev a-mn)
  ```

- ```
    (if (string=? a-mn "") Typo in textbook.
        " "
        (string-append " " (substring a-mn 0 1) ". ")))
  ```

- ```
  ;; Sample expressions for middle-name-abbrev
  (define MN1-VAL " ")
  (define MN2-VAL (string-append " " (substring MN2 0 1) ". "))
  (define MN3-VAL (string-append " " (substring MN3 0 1) ". "))
  ```

- ```
  ;; Tests using sample computations for middle-name-abbrev
  (check-expect (middle-name-abbrev MN1) MN1-VAL)
  (check-expect (middle-name-abbrev MN2) MN2-VAL)
  (check-expect (middle-name-abbrev MN3) MN3-VAL)
  ;; Tests using sample values for middle-name-abbrev
  (check-expect (middle-name-abbrev "Kaliman") " K. ")
  ```

Defining Structures

Structures for the Masses

- To compute a string from a number `number->string` may be used:

Defining Structures

Structures for the Masses

- To compute a string from a number `number->string` may be used:
-
```
;; GPA → string
;; Purpose: Transform the given GPA to a string
(define (gpa->string a-gpa)
  (number->string a-gpa))

;; Sample expressions for gpa->string
(define GPA1-VAL "3.8")
(define GPA2-VAL "3.7")

;; Tests using sample computations for gpa->string
(check-expect (gpa->string OLDGPA1) GPA1-VAL)
(check-expect (gpa->string OLDGPA2) GPA2-VAL)

;; Tests using sample values for gpa->string
(check-expect (gpa->string 4.0) "4.0")
(check-expect (gpa->string 2.3) "2.3")
```

Defining Structures

Structures for the Masses

- To compute a string from a number `number->string` may be used:
- ```
  ;; GPA → string
  ;; Purpose: Transform the given GPA to a string
  (define (gpa->string a-gpa)
    (number->string a-gpa))

  ;; Sample expressions for gpa->string
  (define GPA1-VAL "3.8")
  (define GPA2-VAL "3.7")

  ;; Tests using sample computations for gpa->string
  (check-expect (gpa->string OLDGPA1) GPA1-VAL)
  (check-expect (gpa->string OLDGPA2) GPA2-VAL)

  ;; Tests using sample values for gpa->string
  (check-expect (gpa->string 4.0) "4.0")
  (check-expect (gpa->string 2.3) "2.3")
  ```
- Run the tests
  ```

Defining Structures

Structures for the Masses

- Alas! Six tests fail:

```
Actual value "19/5" differs from "3.8", the expected value.

Actual value "37/10" differs from "3.7", the expected value.

Actual value "4" differs from "4.0", the expected value.

Actual value "23/10" differs from "2.3", the expected value.

Actual value
 "Mercedes G. Merayo has a 397/100 grade point average."
differs from
 "Mercedes G. Merayo has a 3.97 grade point average.",
the expected value.

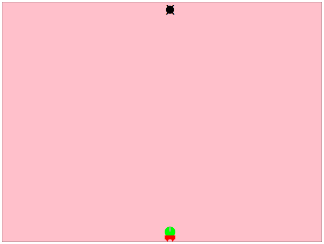
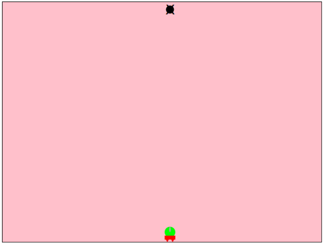
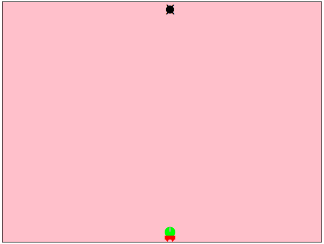
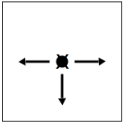
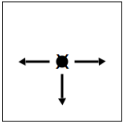
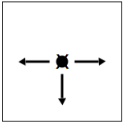
Actual value
 "Manuel Núñez has a 389/100 grade point average."
differs from
 "Manuel Núñez has a 3.89 grade point average.",
the expected value.
```

Defining Structures

Structures for the Masses

- Alas! Six tests fail:

```
Actual value "19/5" differs from "3.8", the expected value.

Actual value "37/10" differs from "3.7", the expected value.

Actual value "4" differs from "4.0", the expected value.

Actual value "23/10" differs from "2.3", the expected value.

Actual value
 "Mercedes G. Merayo has a 397/100 grade point average."
differs from
 "Mercedes G. Merayo has a 3.97 grade point average.",
the expected value.

Actual value
 "Manuel Núñez has a 389/100 grade point average."
differs from
 "Manuel Núñez has a 3.89 grade point average.",
the expected value.
```

- The first four failures are from tests involving `gpa->string` and the last two failures are from tests using sample values for `student2string`
- What happened? Where did those fractions come from?

Defining Structures

Structures for the Masses

- We need to better understand how BSL stores numbers

Defining Structures

Structures for the Masses

- We need to better understand how `BSL` stores numbers
- Whenever possible, `BSL` stores a numerical value as exact (instead of inexact) number
- A real number that may be exactly represented is stored as an integer or a fraction.
- `4.0` is stored as the integer `4`
- `3.8` is stored as the fraction $\frac{19}{5}$
- This explains the test failures for `gpa->string`.

Defining Structures

Structures for the Masses

- Start the debugging process with `gpa->string`
- The problem is that this function needs to return a string representing an inexact number (not an exact number)

Defining Structures

Structures for the Masses

- Start the debugging process with `gpa->string`
- The problem is that this function needs to return a string representing an inexact number (not an exact number)
- A function to transform an exact number to an inexact number is needed

Defining Structures

Structures for the Masses

- Start the debugging process with `gpa->string`
- The problem is that this function needs to return a string representing an inexact number (not an exact number)
- A function to transform an exact number to an inexact number is needed
- Exploring the `BSL` page in the `Help Desk` reveals the following function:

```
number → number
Purpose: Converts an exact number to an inexact one.
(define (exact->inexact x) ...)
```

Defining Structures

Structures for the Masses

- Start the debugging process with gpa->string
- The problem is that this function needs to return a string representing an inexact number (not an exact number)
- A function to transform an exact number to an inexact number is needed
- Exploring the BSL page in the Help Desk reveals the following function:

```
number → number
Purpose: Converts an exact number to an inexact one.
(define (exact->inexact x) ...)
```

- Refined gpa->string is:

```
;; GPA → string
;; Purpose: Transform the given GPA to a string
(define (gpa->string a-gpa)
  (number->string (exact->inexact a-gpa)))
```

- Running the program reveals that all the tests pass
- This completes the design of student2string.

Defining Structures

Homework

- Problems: 84–85, 87–88
- Quiz: Problem 86

Aliens Attack Version 2

- We have the power to refine Aliens Attack version 1:



- Add an alien

Aliens Attack Version 2

- We have the power to refine Aliens Attack version 1:



- Add an alien
- World changes: contains, at least, a `rocket` and an `alien`
- Needed: a data definition for an `alien` and a new `world` data definition

Aliens Attack Version 2

- We have the power to refine Aliens Attack version 1:



- Add an alien
- World changes: contains, at least, a `rocket` and an `alien`
- Needed: a data definition for an `alien` and a new `world` data definition
- A refinement does not mean that we start writing code from scratch
- Well-designed code with separation of concerns helps us avoid reinventing the wheel

Aliens Attack Version 2

Data Definitions

- 
- Think carefully about what changes when the alien is moved right, left, or down

Aliens Attack Version 2

Data Definitions

- 
- Think carefully about what changes when the alien is moved right, left, or down
- the `image-x` of the alien changes
- `image-y` changes
- How can alien may be represented?

Aliens Attack Version 2

Data Definitions

- 
- Think carefully about what changes when the alien is moved right, left, or down
- the `image-x` of the alien changes
- `image-y` changes
- How can alien may be represented?
- `#| An alien is a posn: (make-posn image-x image-y). |#`

Aliens Attack Version 2

Data Definitions

- How does the `world` change?

Aliens Attack Version 2

Data Definitions

- How does the `world` change?
- `rocket` and `alien` change
- Does anything else change?

Aliens Attack Version 2

Data Definitions

- How does the `world` change?
- `rocket` and `alien` change
- Does anything else change?
- The direction of the alien changes
- How do we define the direction?

Aliens Attack Version 2

Data Definitions

- How does the `world` change?
- `rocket` and `alien` change
- Does anything else change?
- The direction of the alien changes
- How do we define the direction?
- ```
  #|     A direction (dir) is either:
          1. 'right
          2. 'left
          3. 'down                      |#
  ```
- How do we define the world?
  ```

Aliens Attack Version 2

Data Definitions

- How does the world change?
- rocket and alien change
- Does anything else change?
- The direction of the alien changes
- How do we define the direction?
- ```
  #|     A direction (dir) is either:
          1. 'right
          2. 'left
          3. 'down                      |#
  ```
- How do we define the world?
- ```
  #| A world is a structure: (make-world rocket alien dir). |#
  (define-struct world (rocket alien dir))
  ```

Aliens Attack Version 2

Function Templates

- ```
  #|   ;; alien ... → ...
       ;; Purpose: ...
       (define (f-on-alien an-alien ...)
         ...(f-on-image-x (posn-x an-alien)...)
         ...(f-on-image-y (posn-y an-alien)...))

       ;; Sample instances
       (define ALIEN1 (make-posn ... ...))

       ;; Sample expressions for f-on-alien
       (define ALIEN-VAL1 ...)  ...

       ;; Tests using sample computations for f-on-alien
       (check-expect (f-on-alien ...) ALIEN-VAL1) ...

       ;; Tests using sample values for f-on-alien
       (check-expect (f-on-alien ...) ...)     ...   |#

       ;; Sample instances of alien
       (define INIT-ALIEN  (make-posn AN-IMG-X 0))
       (define INIT-ALIEN2 (make-posn 3 MAX-IMG-Y))
  ```
  ```

Aliens Attack Version 2

Function Templates

- ```
  #|  ;; dir ... → ...   Purpose: ...
      (define (f-on-dir a-dir ...)
        (cond [(eq? a-dir 'right) ...]
              [(eq? a-dir 'left) ...]
              [else ...]))
      ;; Sample instances of dir
      (define DIR1 ...)  (define DIR2 ...)  (define DIR3 ...)

      ;; Sample expressions for f-on-dir
      (define DIR-VAL1 ...)
      (define DIR-VAL2 ...)
      (define DIR-VAL3 ...) ...
      ;; Tests using sample computations for f-on-dir
      (check-expect (f-on-dir ...) DIR-VAL1)
      (check-expect (f-on-dir ...) DIR-VAL2)
      (check-expect (f-on-dir ...) DIR-VAL3) ...
      ;; Tests using sample values for f-on-dir
      (check-expect (f-on-dir ...) ...)   ...       |#

      ;; Sample instances of dir
      (define INIT-DIR  'right)
      (define INIT-DIR2 'left)
      (define INIT-DIR3 'down)
  ```
  ```

Aliens Attack Version 2

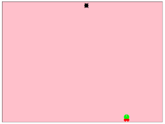
Function Templates

- ```
  ;; world ... → ...
  ;; Purpose: ...
  (define (f-on-world a-world ...)
     ...(f-on-rocket (world-rocket a-world) ...)
     ...(f-on-alien  (world-alien  a-world) ...)
     ...(f-on-dir    (world-dir a-world)    ...))

  ;; Sample expressions for f-on-world
  (define WORLD-VAL ...)

  ;; Tests using sample computations for f-on-world
  (check-expect (f-on-world ...) WORLD-VAL)
       ...

  ;; Tests using sample values for f-on-world
  (check-expect (f-on-world ...) ...)
  ;; Sample instances of world
  (define INIT-WORLD  (make-world INIT-ROCKET
                                  INIT-ALIEN
                                  INIT-DIR))
  (define INIT-WORLD2 (make-world INIT-ROCKET2
                                  INIT-ALIEN2
                                  INIT-DIR2))
  ```
  ```

Aliens Attack Version 2

The `run` Function

- The alien must move every time the clock ticks
- A clock tick handler is needed

Aliens Attack Version 2

The `run` Function

- The alien must move every time the clock ticks
- A clock tick handler is needed
- Game ends when the alien reaches earth
- A handler to detect the end of the game is needed

Aliens Attack Version 2

The run Function

- The alien must move every time the clock ticks
- A clock tick handler is needed
- Game ends when the alien reaches earth
- A handler to detect the end of the game is needed
- ```
  (define TICK-RATE 1/4)

  ; string → world
  ; Purpose: To run the game
  (define (run a-name)
    (big-bang INIT-WORLD
              [on-draw draw-world]
              [name a-name]
              [on-key process-key]
              [on-tick process-tick TICK-RATE]
              [stop-when game-over?]))
  ```
- Limiting the number of clock ticks per second limits alien speed
- Here: the clock ticks every $\frac{1}{4}$ seconds
- You may, of course, adjust the value of `TICK-RATE` to your liking to make the alien move faster or slower.
  ```

Aliens Attack Version 2

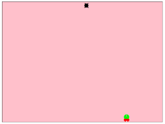
Drawing the world

- ```
  ;; Sample expressions for draw-world
  (define WORLD-SCN1
     (draw-alien (world-alien INIT-WORLD)
                 (draw-rocket (world-rocket INIT-WORLD) E-SCENE)))
  (define WORLD-SCN2
     (draw-alien (world-alien INIT-WORLD2)
                 (draw-rocket (world-rocket INIT-WORLD2) E-SCENE)))
  ```
  ```

Aliens Attack Version 2

Drawing the world

- ```
  ;; world → scene  Purpose: To draw the world in E-SCENE
  (define (draw-world a-world)
  ```

- ```
  ;; Sample expressions for draw-world
  (define WORLD-SCN1
     (draw-alien (world-alien INIT-WORLD)
                 (draw-rocket (world-rocket INIT-WORLD) E-SCENE)))
  (define WORLD-SCN2
     (draw-alien (world-alien INIT-WORLD2)
                 (draw-rocket (world-rocket INIT-WORLD2) E-SCENE)))
  ```

Aliens Attack Version 2

Drawing the world

- ```
  ;; world → scene  Purpose: To draw the world in E-SCENE
  (define (draw-world a-world)
  ```

- ```
  ;; Sample expressions for draw-world
  (define WORLD-SCN1
     (draw-alien (world-alien INIT-WORLD)
                 (draw-rocket (world-rocket INIT-WORLD) E-SCENE)))
  (define WORLD-SCN2
     (draw-alien (world-alien INIT-WORLD2)
                 (draw-rocket (world-rocket INIT-WORLD2) E-SCENE)))
  ```

- ```
  ;; Tests using sample computations for draw-world
  (check-expect (draw-world INIT-WORLD)  WORLD-SCN1)
  (check-expect (draw-world INIT-WORLD2) WORLD-SCN2)
  ;; Tests using sample values
  (check-expect
    (draw-world (make-world INIT-ROCKET2 INIT-ALIEN INIT-DIR3))
  ```



)
  ```

Aliens Attack Version 2

Drawing the world

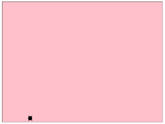
- ```
  ;; world → scene  Purpose: To draw the world in E-SCENE
  (define (draw-world a-world)
  ```
- ```
    (draw-alien (world-alien a-world)
                (draw-rocket (world-rocket a-world) E-SCENE)))
  ```
- ```
  ;; Sample expressions for draw-world
  (define WORLD-SCN1
     (draw-alien (world-alien INIT-WORLD)
                 (draw-rocket (world-rocket INIT-WORLD) E-SCENE)))
  (define WORLD-SCN2
     (draw-alien (world-alien INIT-WORLD2)
                 (draw-rocket (world-rocket INIT-WORLD2) E-SCENE)))
  ```
- ```
  ;; Tests using sample computations for draw-world
  (check-expect (draw-world INIT-WORLD)  WORLD-SCN1)
  (check-expect (draw-world INIT-WORLD2) WORLD-SCN2)
  ;; Tests using sample values
  (check-expect
    (draw-world (make-world INIT-ROCKET2 INIT-ALIEN INIT-DIR3))
  ```

  ```
  )
  ```

Aliens Attack Version 2

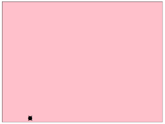
Drawing Aliens

- ```
  ;; Sample expressions for draw-alien
  (define ALIEN-VAL1 (draw-ci ALIEN-IMG
                              (posn-x INIT-ALIEN)
                              (posn-y INIT-ALIEN)
                              E-SCENE))
  (define ALIEN-VAL2 (draw-ci ALIEN-IMG
                              (posn-x INIT-ALIEN2)
                              (posn-y INIT-ALIEN2)
                              E-SCENE2))
  ```
  ```

Aliens Attack Version 2

Drawing Aliens

- ```
  ;; alien scene → scene Purpose: Draw alien in given scene
  (define (draw-alien an-alien scn)
  ```

- ```
  ;; Sample expressions for draw-alien
  (define ALIEN-VAL1 (draw-ci ALIEN-IMG
                              (posn-x INIT-ALIEN)
                              (posn-y INIT-ALIEN)
                              E-SCENE))
  (define ALIEN-VAL2 (draw-ci ALIEN-IMG
                              (posn-x INIT-ALIEN2)
                              (posn-y INIT-ALIEN2)
                              E-SCENE2))
  ```

Aliens Attack Version 2

Drawing Aliens

- ```
  ;; alien scene → scene Purpose: Draw alien in given scene
  (define (draw-alien an-alien scn)
  ```

- ```
  ;; Sample expressions for draw-alien
  (define ALIEN-VAL1 (draw-ci ALIEN-IMG
                              (posn-x INIT-ALIEN)
                              (posn-y INIT-ALIEN)
                              E-SCENE))
  (define ALIEN-VAL2 (draw-ci ALIEN-IMG
                              (posn-x INIT-ALIEN2)
                              (posn-y INIT-ALIEN2)
                              E-SCENE2))
  ```

- ```
  ;; Tests using sample computations for draw-alien
  (check-expect (draw-alien INIT-ALIEN  E-SCENE)  ALIEN-VAL1)
  (check-expect (draw-alien INIT-ALIEN2 E-SCENE2) ALIEN-VAL2)
  ;; Tests using sample values for draw-alien
  (check-expect (draw-alien INIT-ALIEN2 E-SCENE) alien off scene
  ```

  

)
  ```

Aliens Attack Version 2

Drawing Aliens

- ```
  ;; alien scene → scene Purpose: Draw alien in given scene
  (define (draw-alien an-alien scn)
  ```
- ```
    (draw-ci ALIEN-IMG (posn-x an-alien) (posn-y an-alien) scn))
  ```
- ```
  ;; Sample expressions for draw-alien
  (define ALIEN-VAL1 (draw-ci ALIEN-IMG
                              (posn-x INIT-ALIEN)
                              (posn-y INIT-ALIEN)
                              E-SCENE))
  (define ALIEN-VAL2 (draw-ci ALIEN-IMG
                              (posn-x INIT-ALIEN2)
                              (posn-y INIT-ALIEN2)
                              E-SCENE2))
  ```
- ```
  ;; Tests using sample computations for draw-alien
  (check-expect (draw-alien INIT-ALIEN  E-SCENE)  ALIEN-VAL1)
  (check-expect (draw-alien INIT-ALIEN2 E-SCENE2) ALIEN-VAL2)
  ;; Tests using sample values for draw-alien
  (check-expect (draw-alien INIT-ALIEN2 E-SCENE) alien off scene
  ```



)

Aliens Attack Version 2

The process-key Refinement

- ```
  ;; Sample expressions for process-key
  (define KEY-RVAL (make-world (move-rckt-right (world-rocket INIT-WORLD))
                               (world-alien INIT-WORLD)
                               (world-dir INIT-WORLD)))
  (define KEY-LVAL (make-world (move-rckt-left (world-rocket INIT-WORLD))
                               (world-alien INIT-WORLD)
                               (world-dir INIT-WORLD)))
  (define KEY-OVAL INIT-WORLD2)
  ```
  ```

Aliens Attack Version 2

The process-key Refinement

- ```
  ;; world key → world
  ;; Purpose: Process a key event to return next world
  (define (process-key a-world a-key)
  ```

- ```
  ;; Sample expressions for process-key
  (define KEY-RVAL (make-world (move-rckt-right (world-rocket INIT-WORLD))
                               (world-alien INIT-WORLD)
                               (world-dir INIT-WORLD)))
  (define KEY-LVAL (make-world (move-rckt-left (world-rocket INIT-WORLD))
                               (world-alien INIT-WORLD)
                               (world-dir INIT-WORLD)))
  (define KEY-OVAL INIT-WORLD2)
  ```

Aliens Attack Version 2

The process-key Refinement

- ```
  ;; world key → world
  ;; Purpose: Process a key event to return next world
  (define (process-key a-world a-key)
  ```

- ```
  ;; Sample expressions for process-key
  (define KEY-RVAL (make-world (move-rckt-right (world-rocket INIT-WORLD))
                               (world-alien INIT-WORLD)
                               (world-dir INIT-WORLD)))
  (define KEY-LVAL (make-world (move-rckt-left (world-rocket INIT-WORLD))
                               (world-alien INIT-WORLD)
                               (world-dir INIT-WORLD)))
  (define KEY-OVAL INIT-WORLD2)
  ```

- ```
  ;; Tests using sample computations for process-key
  (check-expect (process-key INIT-WORLD  "right") KEY-RVAL)
  (check-expect (process-key INIT-WORLD  "left")  KEY-LVAL)
  (check-expect (process-key INIT-WORLD2 "m")     KEY-OVAL)
  ;; Tests using sample values for process-key
     ...
  (check-expect (process-key (make-world 0 INIT-ALIEN 'left)
                             "left")
                (make-world 0 INIT-ALIEN 'left))
  (check-expect (process-key (make-world 0 INIT-ALIEN 'left)
                             "o")
                (make-world 0 INIT-ALIEN 'left))
  ```
  ```

Aliens Attack Version 2

The process-key Refinement

- ```
  ;; world key → world
  ;; Purpose: Process a key event to return next world
  (define (process-key a-world a-key)
  ```
- ```
   (cond [(key=? a-key "right")
          (make-world (move-rckt-right (world-rocket a-world))
                      (world-alien a-world)
                      (world-dir a-world))]
         [(key=? a-key "left")
          (make-world (move-rckt-left (world-rocket a-world))
                      (world-alien a-world)
                      (world-dir a-world))]
         [else a-world]))
  ```
- ```
  ;; Sample expressions for process-key
  (define KEY-RVAL (make-world (move-rckt-right (world-rocket INIT-WORLD))
                               (world-alien INIT-WORLD)
                               (world-dir INIT-WORLD)))
  (define KEY-LVAL (make-world (move-rckt-left (world-rocket INIT-WORLD))
                               (world-alien INIT-WORLD)
                               (world-dir INIT-WORLD)))
  (define KEY-OVAL INIT-WORLD2)
  ```
- ```
  ;; Tests using sample computations for process-key
  (check-expect (process-key INIT-WORLD  "right") KEY-RVAL)
  (check-expect (process-key INIT-WORLD  "left")  KEY-LVAL)
  (check-expect (process-key INIT-WORLD2 "m")     KEY-OVAL)
  ;; Tests using sample values for process-key
     ...
  (check-expect (process-key (make-world 0 INIT-ALIEN 'left)
                             "left")
                (make-world 0 INIT-ALIEN 'left))
  (check-expect (process-key (make-world 0 INIT-ALIEN 'left)
                             "o")
                (make-world 0 INIT-ALIEN 'left))
  ```

Aliens Attack Version 2

Processing Ticks

- How does the game evolve every time the clock ticks? What changes? What does not change?

Aliens Attack Version 2

Processing Ticks

- How does the game evolve every time the clock ticks? What changes? What does not change?
- The rocket is not affected by clock ticks
- The alien must move
- The direction of the moved alien may be different
- When does the direction change?

Aliens Attack Version 2

Processing Ticks

- How does the game evolve every time the clock ticks? What changes? What does not change?
- The rocket is not affected by clock ticks
- The alien must move
- The direction of the moved alien may be different
- When does the direction change?
- Only changes when the alien is at either the left or right edge of the scene:
  - New alien at right edge created by moving right means the new direction is down
  - New alien at left edge created by moving left means the new direction is down
  - New alien at right edge created by moving down means the new direction is left
  - New alien at left edge created by moving down means the new direction is right
  - Otherwise, the direction does not change.
- Data required: the new alien and the direction

Aliens Attack Version 2

The process-tick Handler

- ```
  ;; Sample expressions for process-tick
  (define AFTER-TICK-WORLD1
          (make-world (world-rocket INIT-WORLD)
                      (move-alien (world-alien INIT-WORLD)
                                  (world-dir INIT-WORLD))
                      (new-dir-after-tick
                        (move-alien (world-alien INIT-WORLD)
                                    (world-dir INIT-WORLD))
                                    (world-dir INIT-WORLD))))

  (define AFTER-TICK-WORLD2
          (make-world (world-rocket INIT-WORLD2)
                      (move-alien (world-alien INIT-WORLD2)
                                  (world-dir INIT-WORLD2))
                      (new-dir-after-tick
                        (move-alien (world-alien INIT-WORLD2)
                                    (world-dir INIT-WORLD2))
                                    (world-dir INIT-WORLD2))))
  ```
  ```

Aliens Attack Version 2

The process-tick Handler

- ```
  ;; world → world
  ;; Purpose: Create a new world after a clock tick
  (define (process-tick a-world)
    (make-world
      (world-rocket a-world)
      (move-alien (world-alien a-world) (world-dir a-world))
      (new-dir-after-tick (move-alien (world-alien a-world)
                                      (world-dir a-world))
                          (world-dir a-world))))
  ```
- ```
  ;; Sample expressions for process-tick
  (define AFTER-TICK-WORLD1
          (make-world (world-rocket INIT-WORLD)
                      (move-alien (world-alien INIT-WORLD)
                                  (world-dir INIT-WORLD))
                      (new-dir-after-tick
                        (move-alien (world-alien INIT-WORLD)
                                    (world-dir INIT-WORLD))
                                    (world-dir INIT-WORLD))))

  (define AFTER-TICK-WORLD2
          (make-world (world-rocket INIT-WORLD2)
                      (move-alien (world-alien INIT-WORLD2)
                                  (world-dir INIT-WORLD2))
                      (new-dir-after-tick
                        (move-alien (world-alien INIT-WORLD2)
                                    (world-dir INIT-WORLD2))
                                    (world-dir INIT-WORLD2))))
  ```

Aliens Attack Version 2

The process-tick Handler

- ```
  ;; Tests using sample computations for process-tick
  (check-expect (process-tick INIT-WORLD)  AFTER-TICK-WORLD1)
  (check-expect (process-tick INIT-WORLD2) AFTER-TICK-WORLD2)

  ;; Tests using sample values for process-tick
  (check-expect (process-tick (make-world INIT-ROCKET
                                          (make-posn 1 5)
                                          'left))
                (make-world INIT-ROCKET
                            (make-posn MIN-IMG-X 5)
                            'down))

  (check-expect (process-tick
                  (make-world
                    INIT-ROCKET2
                    (make-posn (- MAX-CHARS-HORIZONTAL 2) 10)
                    'right))
                (make-world INIT-ROCKET2
                            (make-posn MAX-IMG-X 10)
                            'down))

  (check-expect (process-tick (make-world INIT-ROCKET2
                                          (make-posn MAX-IMG-X 2)
                                          'down))
  ```
  ```

Aliens Attack Version 2

The Design of `new-dir-after-tick`

- `new-dir-after-tick` takes as input two different types of data
- Should this function be designed by specializing the template for a function on an alien or a function on a direction?

Aliens Attack Version 2

The Design of `new-dir-after-tick`

- `new-dir-after-tick` takes as input two different types of data
- Should this function be designed by specializing the template for a function on an alien or a function on a direction?
- Every time the clock ticks the direction of the alien may change:
  1. If the direction is right then the new direction may be left or down.
  2. If the direction is left then the new direction may be down or left.
  3. If the direction is down then the new direction may be right or left.
- Suggests that the function ought to be designed around the given direction

Aliens Attack Version 2

The Design of `new-dir-after-tick`

- `new-dir-after-tick` takes as input two different types of data
- Should this function be designed by specializing the template for a function on an alien or a function on a direction?
- Every time the clock ticks the direction of the alien may change:
  1. If the direction is right then the new direction may be left or down.
  2. If the direction is left then the new direction may be down or left.
  3. If the direction is down then the new direction may be right or left.
- Suggests that the function ought to be designed around the given direction
- For testing aliens on the left and right edges:

```
(define LEFT-EDGE-ALIEN  (make-posn MIN-IMG-X 10))
(define RIGHT-EDGE-ALIEN (make-posn MAX-IMG-X 6))
```

Aliens Attack Version 2

The Design of `new-dir-after-tick`

- `new-dir-after-tick` takes as input two different types of data
- Should this function be designed by specializing the template for a function on an alien or a function on a direction?
- Every time the clock ticks the direction of the alien may change:
  1. If the direction is right then the new direction may be left or down.
  2. If the direction is left then the new direction may be down or left.
  3. If the direction is down then the new direction may be right or left.
- Suggests that the function ought to be designed around the given direction
- For testing aliens on the left and right edges:

```
(define LEFT-EDGE-ALIEN  (make-posn MIN-IMG-X 10))
(define RIGHT-EDGE-ALIEN (make-posn MAX-IMG-X 6))
```

- Computation of new direction done by different auxiliary functions:
  1. `new-dir-after-down` used when the direction is down.
  2. `new-dir-after-left` used when the direction is left.
  3. `new-dir-after-right`used when the direction is right.

Aliens Attack Version 2

The Design of new-dir-after-tick

- ```
  ;; Sample expressions for new-dir-after-tick
  (define NEW-DIR-LEDGE-ALIEN-DOWN
            (new-dir-after-down LEFT-EDGE-ALIEN))
  (define NEW-DIR-REDGE-ALIEN-DOWN
            (new-dir-after-down RIGHT-EDGE-ALIEN))
  (define NEW-DIR-INIT-ALIEN-LEFT
            (new-dir-after-left INIT-ALIEN))
  (define NEW-DIR-LEDGE-ALIEN-LEFT
            (new-dir-after-left LEFT-EDGE-ALIEN))
  (define NEW-DIR-INIT-ALIEN-RIGHT
            (new-dir-after-right INIT-ALIEN))
  (define NEW-DIR-REDGE-ALIEN-RIGHT
            (new-dir-after-right RIGHT-EDGE-ALIEN))
  ```
  ```

Aliens Attack Version 2

The Design of new-dir-after-tick

- ```
  ;; alien dir → dir
  ;; Purpose: Return new alien direction
  (define (new-dir-after-tick an-alien old-dir)
    (cond [(eq? old-dir 'right)
           (new-dir-after-right an-alien)]
          [(eq? old-dir 'left)
           (new-dir-after-left an-alien)]
          [else (new-dir-after-down an-alien)]))
  ```
- ```
    ;; Sample expressions for new-dir-after-tick
    (define NEW-DIR-LEDGE-ALIEN-DOWN
              (new-dir-after-down LEFT-EDGE-ALIEN))
    (define NEW-DIR-REDGE-ALIEN-DOWN
              (new-dir-after-down RIGHT-EDGE-ALIEN))
    (define NEW-DIR-INIT-ALIEN-LEFT
              (new-dir-after-left INIT-ALIEN))
    (define NEW-DIR-LEDGE-ALIEN-LEFT
              (new-dir-after-left LEFT-EDGE-ALIEN))
    (define NEW-DIR-INIT-ALIEN-RIGHT
              (new-dir-after-right INIT-ALIEN))
    (define NEW-DIR-REDGE-ALIEN-RIGHT
              (new-dir-after-right RIGHT-EDGE-ALIEN))
  ```

Aliens Attack Version 2

The Design of new-dir-after-tick

- ```
  ;; Tests using sample computations for new-dir-after-tick
  (check-expect (new-dir-after-tick LEFT-EDGE-ALIEN 'down)
                NEW-DIR-LEDGE-ALIEN-DOWN)
  (check-expect (new-dir-after-tick RIGHT-EDGE-ALIEN 'down)
                NEW-DIR-REDGE-ALIEN-DOWN)
  (check-expect (new-dir-after-tick INIT-ALIEN 'left)
                NEW-DIR-INIT-ALIEN-LEFT)
  (check-expect (new-dir-after-tick LEFT-EDGE-ALIEN 'left)
                NEW-DIR-LEDGE-ALIEN-LEFT)
  (check-expect (new-dir-after-tick INIT-ALIEN 'right)
                NEW-DIR-INIT-ALIEN-RIGHT)
  (check-expect (new-dir-after-tick RIGHT-EDGE-ALIEN 'right)
                NEW-DIR-REDGE-ALIEN-RIGHT)
  ;; Tests using sample values for new-dir-after-tick
  (check-expect (new-dir-after-tick (make-posn MIN-IMG-X 10) 'down)
                'right)
  (check-expect (new-dir-after-tick (make-posn MAX-IMG-X 12) 'down)
                'left)
  (check-expect (new-dir-after-tick (make-posn 10 10) 'left)
                'left)
  (check-expect (new-dir-after-tick (make-posn MIN-IMG-X 15) 'left)
                'down)
  (check-expect (new-dir-after-tick (make-posn 10 14) 'right)
                'right)
  ```
  ```

Aliens Attack Version 2

Design of new-dir-after-down

- ```
  ;; Sample expressions for new-dir-after-down
  (define AT-LEDGE-DOWN 'right)
  (define AT-REDGE-DOWN 'left)
  ```
  ```

Aliens Attack Version 2

Design of new-dir-after-down

- ```
  ;; alien → direction
  ;; Purpose: Compute the direction of the given alien
  ;;          when previous direction is down
  (define (new-dir-after-down an-alien)
  ```

- ```
  ;; Sample expressions for new-dir-after-down
  (define AT-LEDGE-DOWN 'right)
  (define AT-REDGE-DOWN 'left)
  ```

Aliens Attack Version 2

Design of new-dir-after-down

- ```
  ;; alien → direction
  ;; Purpose: Compute the direction of the given alien
  ;;          when previous direction is down
  (define (new-dir-after-down an-alien)
  ```

- ```
  ;; Sample expressions for new-dir-after-down
  (define AT-LEDGE-DOWN 'right)
  (define AT-REDGE-DOWN 'left)
  ```

- ```
  ;; Tests using sample computations for new-dir-after-down
  (check-expect (new-dir-after-down LEFT-EDGE-ALIEN)
                AT-LEDGE-DOWN)
  (check-expect (new-dir-after-down RIGHT-EDGE-ALIEN)
                AT-REDGE-DOWN)

  ;; Tests using sample values for new-dir-after-down
  (check-expect (new-dir-after-down (make-posn MIN-IMG-X 4))
                'right)
  (check-expect (new-dir-after-down (make-posn MAX-IMG-X 9))
                'left)
  ```
  ```

Aliens Attack Version 2

Design of new-dir-after-down

- ```
  ;; alien → direction
  ;; Purpose: Compute the direction of the given alien
  ;;          when previous direction is down
  (define (new-dir-after-down an-alien)
  ```
- ```
    (if (alien-at-left-edge? an-alien)
        'right
        'left))
  ```
- ```
  ;; Sample expressions for new-dir-after-down
  (define AT-LEDGE-DOWN 'right)
  (define AT-REDGE-DOWN 'left)
  ```
- ```
  ;; Tests using sample computations for new-dir-after-down
  (check-expect (new-dir-after-down LEFT-EDGE-ALIEN)
                AT-LEDGE-DOWN)
  (check-expect (new-dir-after-down RIGHT-EDGE-ALIEN)
                AT-REDGE-DOWN)

  ;; Tests using sample values for new-dir-after-down
  (check-expect (new-dir-after-down (make-posn MIN-IMG-X 4))
                'right)
  (check-expect (new-dir-after-down (make-posn MAX-IMG-X 9))
                'left)
  ```

Aliens Attack Version 2

Design of new-dir-after-left

- ```
  ;; Sample expressions for new-dir-after-left
  (define AT-LEDGE     'down)
  (define NOT-AT-LEDGE 'left)
  ```
  ```

Aliens Attack Version 2

Design of new-dir-after-left

- ```
  ;; alien → direction
  ;; Purpose: Compute the direction of the given alien
  ;;          when previous direction is left
  (define (new-dir-after-left an-alien)
  ```

- ```
  ;; Sample expressions for new-dir-after-left
  (define AT-LEDGE     'down)
  (define NOT-AT-LEDGE 'left)
  ```

Aliens Attack Version 2

Design of new-dir-after-left

- ```
  ;; alien → direction
  ;; Purpose: Compute the direction of the given alien
  ;;          when previous direction is left
  (define (new-dir-after-left an-alien)
  ```

- ```
  ;; Sample expressions for new-dir-after-left
  (define AT-LEDGE     'down)
  (define NOT-AT-LEDGE 'left)
  ```

- ```
  ;; Tests using sample computations for new-dir-after-left
  (check-expect (new-dir-after-left LEFT-EDGE-ALIEN)  'down)
  (check-expect (new-dir-after-left INIT-ALIEN)       'left)

  ;; Tests using sample values for new-dir-after-left
  (check-expect (new-dir-after-left RIGHT-EDGE-ALIEN) 'left)
  ```
  ```

Aliens Attack Version 2

Design of new-dir-after-left

- ```
  ;; alien → direction
  ;; Purpose: Compute the direction of the given alien
  ;;          when previous direction is left
  (define (new-dir-after-left an-alien)
  ```
- ```
    (if (alien-at-left-edge? an-alien)
        'down
        'left))
  ```
- ```
  ;; Sample expressions for new-dir-after-left
  (define AT-LEDGE     'down)
  (define NOT-AT-LEDGE 'left)
  ```
- ```
  ;; Tests using sample computations for new-dir-after-left
  (check-expect (new-dir-after-left LEFT-EDGE-ALIEN)  'down)
  (check-expect (new-dir-after-left INIT-ALIEN)       'left)

  ;; Tests using sample values for new-dir-after-left
  (check-expect (new-dir-after-left RIGHT-EDGE-ALIEN) 'left)
  ```

Aliens Attack Version 2

Design of new-dir-after-right

- ```
  ;; Sample expressions for new-dir-after-right
  (define AT-REDGE     'down)
  (define NOT-AT-REDGE 'right)
  ```
  ```

Aliens Attack Version 2

Design of new-dir-after-right

- ```
  ;; alien → direction
  ;; Purpose: Compute the direction of the given alien
  ;;          when previous direction is right
  (define (new-dir-after-right an-alien)
  ```

- ```
  ;; Sample expressions for new-dir-after-right
  (define AT-REDGE     'down)
  (define NOT-AT-REDGE 'right)
  ```

Aliens Attack Version 2

Design of new-dir-after-right

- ```
  ;; alien → direction
  ;; Purpose: Compute the direction of the given alien
  ;;          when previous direction is right
  (define (new-dir-after-right an-alien)
  ```
- ```
  ;; Sample expressions for new-dir-after-right
  (define AT-REDGE     'down)
  (define NOT-AT-REDGE 'right)
  ```
- ```
  ;; Tests using sample computations for new-dir-after-right
  (check-expect (new-dir-after-right RIGHT-EDGE-ALIEN)
                AT-REDGE)
  (check-expect (new-dir-after-right INIT-ALIEN)
                NOT-AT-REDGE)

  ;; Tests using sample values for new-dir-after-right
  (check-expect (new-dir-after-right LEFT-EDGE-ALIEN)
                'right)
  ```
  ```

Aliens Attack Version 2

Design of new-dir-after-right

- ```
  ;; alien → direction
  ;; Purpose: Compute the direction of the given alien
  ;;          when previous direction is right
  (define (new-dir-after-right an-alien)
  ```
- ```
    (if (alien-at-right-edge? an-alien)
        'down
        'right))
  ```
- ```
  ;; Sample expressions for new-dir-after-right
  (define AT-REDGE     'down)
  (define NOT-AT-REDGE 'right)
  ```
- ```
  ;; Tests using sample computations for new-dir-after-right
  (check-expect (new-dir-after-right RIGHT-EDGE-ALIEN)
                AT-REDGE)
  (check-expect (new-dir-after-right INIT-ALIEN)
                NOT-AT-REDGE)

  ;; Tests using sample values for new-dir-after-right
  (check-expect (new-dir-after-right LEFT-EDGE-ALIEN)
                'right)
  ```

Aliens Attack Version 2

Design of alien-at-left-edge?

- ```
  ;; Sample expressions for alien-at-left-edge?
  (define LEDGE-VAL1 (= (posn-x INIT-ALIEN)      MIN-IMG-X))
  (define LEDGE-VAL2 (= (posn-x LEFT-EDGE-ALIEN) MIN-IMG-X))
  ```
  ```

Aliens Attack Version 2

Design of alien-at-left-edge?

- ```
  ;; alien → Boolean
  ;; Purpose: Determine if he given alien is at the left edge
  (define (alien-at-left-edge? an-alien)
  ```

- ```
  ;; Sample expressions for alien-at-left-edge?
  (define LEDGE-VAL1 (= (posn-x INIT-ALIEN)      MIN-IMG-X))
  (define LEDGE-VAL2 (= (posn-x LEFT-EDGE-ALIEN) MIN-IMG-X))
  ```

Aliens Attack Version 2

Design of alien-at-left-edge?

- ```
  ;; alien → Boolean
  ;; Purpose: Determine if he given alien is at the left edge
  (define (alien-at-left-edge? an-alien)
  ```
- ```
  ;; Sample expressions for alien-at-left-edge?
  (define LEDGE-VAL1 (= (posn-x INIT-ALIEN)      MIN-IMG-X))
  (define LEDGE-VAL2 (= (posn-x LEFT-EDGE-ALIEN) MIN-IMG-X))
  ```
- ```
  ;; Tests using sample computations for alien-at-left-edge?
  (check-expect (alien-at-left-edge? INIT-ALIEN) LEDGE-VAL1)
  (check-expect (alien-at-left-edge? LEFT-EDGE-ALIEN) LEDGE-VAL2)

  ;; Tests using sample values for alien-at-left-edge?
  (check-expect (alien-at-left-edge? (make-posn 3 2)) #false)
  (check-expect (alien-at-left-edge? (make-posn MIN-IMG-X 8))
                #true)
  ```
  ```

Aliens Attack Version 2

Design of alien-at-left-edge?

- ```
  ;; alien → Boolean
  ;; Purpose: Determine if he given alien is at the left edge
  (define (alien-at-left-edge? an-alien)
  ```
- ```
    (= (posn-x an-alien) MIN-IMG-X))
  ```
- ```
  ;; Sample expressions for alien-at-left-edge?
  (define LEDGE-VAL1 (= (posn-x INIT-ALIEN)      MIN-IMG-X))
  (define LEDGE-VAL2 (= (posn-x LEFT-EDGE-ALIEN) MIN-IMG-X))
  ```
- ```
  ;; Tests using sample computations for alien-at-left-edge?
  (check-expect (alien-at-left-edge? INIT-ALIEN) LEDGE-VAL1)
  (check-expect (alien-at-left-edge? LEFT-EDGE-ALIEN) LEDGE-VAL2)

  ;; Tests using sample values for alien-at-left-edge?
  (check-expect (alien-at-left-edge? (make-posn 3 2)) #false)
  (check-expect (alien-at-left-edge? (make-posn MIN-IMG-X 8))
                #true)
  ```

Aliens Attack Version 2

Design of alien-at-right-edge?

- ```
  ;; Sample expressions for alien-at-right-edge?
  (define REDGE-VAL1 (= (posn-x INIT-ALIEN) MAX-IMG-X))
  (define REDGE-VAL2 (= (posn-x (make-posn MAX-IMG-X 8))
                        MAX-IMG-X))
  ```
  ```

Aliens Attack Version 2

Design of alien-at-right-edge?

- ```
  ;; alien → Boolean
  ;; Purpose: Determine if the given alien is at the
  ;;          right edge
  (define (alien-at-right-edge? an-alien)
  ```

- ```
  ;; Sample expressions for alien-at-right-edge?
  (define REDGE-VAL1 (= (posn-x INIT-ALIEN) MAX-IMG-X))
  (define REDGE-VAL2 (= (posn-x (make-posn MAX-IMG-X 8))
                        MAX-IMG-X))
  ```

Aliens Attack Version 2

Design of alien-at-right-edge?

- ```
  ;; alien → Boolean
  ;; Purpose: Determine if the given alien is at the
  ;;          right edge
  (define (alien-at-right-edge? an-alien)
  ```

- ```
  ;; Sample expressions for alien-at-right-edge?
  (define REDGE-VAL1 (= (posn-x INIT-ALIEN) MAX-IMG-X))
  (define REDGE-VAL2 (= (posn-x (make-posn MAX-IMG-X 8))
                        MAX-IMG-X))
  ```

- ```
  ;; Tests using sample computations for alien-at-right-edge?
  (check-expect (alien-at-right-edge? INIT-ALIEN)
                REDGE-VAL1)
  (check-expect (alien-at-right-edge? RIGHT-EDGE-ALIEN)
                REDGE-VAL2)

  ;; Tests using sample values for alien-at-right-edge?
  (check-expect (alien-at-right-edge? (make-posn 1 1))  #false)
  (check-expect (alien-at-right-edge? RIGHT-EDGE-ALIEN) #true)
  ```
  ```

Aliens Attack Version 2

Design of alien-at-right-edge?

- ```
  ;; alien → Boolean
  ;; Purpose: Determine if the given alien is at the
  ;;          right edge
  (define (alien-at-right-edge? an-alien)
  ```
- ```
    (= (posn-x an-alien) MAX-IMG-X))
  ```
- ```
  ;; Sample expressions for alien-at-right-edge?
  (define REDGE-VAL1 (= (posn-x INIT-ALIEN) MAX-IMG-X))
  (define REDGE-VAL2 (= (posn-x (make-posn MAX-IMG-X 8))
                        MAX-IMG-X))
  ```
- ```
  ;; Tests using sample computations for alien-at-right-edge?
  (check-expect (alien-at-right-edge? INIT-ALIEN)
                REDGE-VAL1)
  (check-expect (alien-at-right-edge? RIGHT-EDGE-ALIEN)
                REDGE-VAL2)

  ;; Tests using sample values for alien-at-right-edge?
  (check-expect (alien-at-right-edge? (make-posn 1 1))  #false)
  (check-expect (alien-at-right-edge? RIGHT-EDGE-ALIEN) #true)
  ```

Aliens Attack Version 2

Homework

- Problem: 92 Typo: `alien-at-left-edge?` and `alien-at-right-edge?`

Aliens Attack Version 2

The Design of `move-alien`

- Must decide how to design it, input: an `alien` and a `dir`

Aliens Attack Version 2

The Design of `move-alien`

- Must decide how to design it, input: an `alien` and a `dir`
- Specializing the template for functions on a `dir` means that a new alien is created by computing either a new `image-x` value or a new `image-y` value depending on the given `dir`
- A conditional is needed to determine which is computed

Aliens Attack Version 2

The Design of `move-alien`

- Must decide how to design it, input: an `alien` and a `dir`
- Specializing the template for functions on a `dir` means that a new alien is created by computing either a new `image-x` value or a new `image-y` value depending on the given `dir`
- A conditional is needed to determine which is computed
- Specializing the template for functions on a `alien` means that a new alien is constructed by computing an `image-x` value using the given alien's `image-x` coordinate and the given direction and by computing an `image-y` value using the given alien's `image-y` coordinate and the given direction
- The functions to compute the new `image-x` and `image-y` coordinates must have a conditional

Aliens Attack Version 2

The Design of `move-alien`

- Must decide how to design it, input: an `alien` and a `dir`
- Specializing the template for functions on a `dir` means that a new alien is created by computing either a new `image-x` value or a new `image-y` value depending on the given `dir`
- A conditional is needed to determine which is computed
- Specializing the template for functions on a `alien` means that a new alien is constructed by computing an `image-x` value using the given alien's `image-x` coordinate and the given direction and by computing an `image-y` value using the given alien's `image-y` coordinate and the given direction
- The functions to compute the new `image-x` and `image-y` coordinates must have a conditional
- `move-alien` may be designed by specializing either template
- Which design seems easier?

Aliens Attack Version 2

The Design of move-alien

- ```
  ;; Sample expressions for move-alien
  (define MALIEN-VAL1-1
          (make-posn (move-right-image-x (posn-x INIT-ALIEN)) (posn-y INIT-ALIEN)))
  (define MALIEN-VAL1-2
          (make-posn (move-right-image-x (posn-x INIT-ALIEN2)) (posn-y INIT-ALIEN2)))
  (define MALIEN-VAL2-1
          (make-posn (move-left-image-x (posn-x INIT-ALIEN)) (posn-y INIT-ALIEN)))
  (define MALIEN-VAL2-2
          (make-posn (move-left-image-x (posn-x INIT-ALIEN2)) (posn-y INIT-ALIEN2)))
  (define MALIEN-VAL3-1
          (make-posn (posn-x INIT-ALIEN) (move-down-image-y (posn-y INIT-ALIEN))))
  (define MALIEN-VAL3-2
          (make-posn (posn-x INIT-ALIEN2) (move-down-image-y (posn-y INIT-ALIEN2))))
  ```
  ```

Aliens Attack Version 2

The Design of move-alien

- ```
  ;; alien dir → alien   Purpose: Move alien in given direction
  (define (move-alien an-alien a-dir)
  ```

- ```
  ;; Sample expressions for move-alien
  (define MALIEN-VAL1-1
          (make-posn (move-right-image-x (posn-x INIT-ALIEN)) (posn-y INIT-ALIEN)))
  (define MALIEN-VAL1-2
          (make-posn (move-right-image-x (posn-x INIT-ALIEN2)) (posn-y INIT-ALIEN2)))
  (define MALIEN-VAL2-1
          (make-posn (move-left-image-x (posn-x INIT-ALIEN)) (posn-y INIT-ALIEN)))
  (define MALIEN-VAL2-2
          (make-posn (move-left-image-x (posn-x INIT-ALIEN2)) (posn-y INIT-ALIEN2)))
  (define MALIEN-VAL3-1
          (make-posn (posn-x INIT-ALIEN) (move-down-image-y (posn-y INIT-ALIEN))))
  (define MALIEN-VAL3-2
          (make-posn (posn-x INIT-ALIEN2) (move-down-image-y (posn-y INIT-ALIEN2))))
  ```

Aliens Attack Version 2

The Design of move-alien

- ```
  ;; alien dir → alien   Purpose: Move alien in given direction
  (define (move-alien an-alien a-dir)
  ```

- ```
  ;; Sample expressions for move-alien
  (define MALIEN-VAL1-1
          (make-posn (move-right-image-x (posn-x INIT-ALIEN)) (posn-y INIT-ALIEN)))
  (define MALIEN-VAL1-2
          (make-posn (move-right-image-x (posn-x INIT-ALIEN2)) (posn-y INIT-ALIEN2)))
  (define MALIEN-VAL2-1
          (make-posn (move-left-image-x (posn-x INIT-ALIEN)) (posn-y INIT-ALIEN)))
  (define MALIEN-VAL2-2
          (make-posn (move-left-image-x (posn-x INIT-ALIEN2)) (posn-y INIT-ALIEN2)))
  (define MALIEN-VAL3-1
          (make-posn (posn-x INIT-ALIEN) (move-down-image-y (posn-y INIT-ALIEN))))
  (define MALIEN-VAL3-2
          (make-posn (posn-x INIT-ALIEN2) (move-down-image-y (posn-y INIT-ALIEN2))))
  ```

- ```
  ;; Tests using sample computations for move-alien
  (check-expect (move-alien INIT-ALIEN  'right) MALIEN-VAL1-1)
  (check-expect (move-alien INIT-ALIEN2 'right) MALIEN-VAL1-2)
  (check-expect (move-alien INIT-ALIEN  'left)  MALIEN-VAL2-1)
  (check-expect (move-alien INIT-ALIEN2 'left)  MALIEN-VAL2-2)
  (check-expect (move-alien INIT-ALIEN  'down)  MALIEN-VAL3-1)
  (check-expect (move-alien INIT-ALIEN2 'down)  MALIEN-VAL3-2)
  ;; Tests using sample values for move-alien
  (check-expect (move-alien (make-posn MAX-IMG-X 3) 'down) (make-posn MAX-IMG-X 4))
  (check-expect (move-alien (make-posn MAX-IMG-X 3) 'left) (make-posn (sub1 MAX-IMG-X) 3))
  (check-expect (move-alien (make-posn 0 5) 'right) (make-posn 1 5))
  ```
  ```

Aliens Attack Version 2

The Design of move-alien

```
;; alien dir → alien   Purpose: Move alien in given direction
(define (move-alien an-alien a-dir)
```

```
  (cond [(eq? a-dir 'right)
         (make-posn (move-right-image-x (posn-x an-alien)) (posn-y an-alien))]
        [(eq? a-dir 'left)
         (make-posn (move-left-image-x (posn-x an-alien)) (posn-y an-alien))]
        [else (make-posn (posn-x an-alien) (move-down-image-y (posn-y an-alien)))]))
```

```
;; Sample expressions for move-alien
(define MALIEN-VAL1-1
        (make-posn (move-right-image-x (posn-x INIT-ALIEN)) (posn-y INIT-ALIEN)))
(define MALIEN-VAL1-2
        (make-posn (move-right-image-x (posn-x INIT-ALIEN2)) (posn-y INIT-ALIEN2)))
(define MALIEN-VAL2-1
        (make-posn (move-left-image-x (posn-x INIT-ALIEN)) (posn-y INIT-ALIEN)))
(define MALIEN-VAL2-2
        (make-posn (move-left-image-x (posn-x INIT-ALIEN2)) (posn-y INIT-ALIEN2)))
(define MALIEN-VAL3-1
        (make-posn (posn-x INIT-ALIEN) (move-down-image-y (posn-y INIT-ALIEN))))
(define MALIEN-VAL3-2
        (make-posn (posn-x INIT-ALIEN2) (move-down-image-y (posn-y INIT-ALIEN2))))
```

```
;; Tests using sample computations for move-alien
(check-expect (move-alien INIT-ALIEN  'right) MALIEN-VAL1-1)
(check-expect (move-alien INIT-ALIEN2 'right) MALIEN-VAL1-2)
(check-expect (move-alien INIT-ALIEN  'left)  MALIEN-VAL2-1)
(check-expect (move-alien INIT-ALIEN2 'left)  MALIEN-VAL2-2)
(check-expect (move-alien INIT-ALIEN  'down)  MALIEN-VAL3-1)
(check-expect (move-alien INIT-ALIEN2 'down)  MALIEN-VAL3-2)
;; Tests using sample values for move-alien
(check-expect (move-alien (make-posn MAX-IMG-X 3) 'down) (make-posn MAX-IMG-X 4))
(check-expect (move-alien (make-posn MAX-IMG-X 3) 'left) (make-posn (sub1 MAX-IMG-X) 3))
(check-expect (move-alien (make-posn 0 5) 'right) (make-posn 1 5))
```

Aliens Attack Version 2

Subtyping

- The development of the auxiliary functions needed by `move-alien` is paused to address a bug in our design
- Have you picked up on it?

Aliens Attack Version 2

Subtyping

- The development of the auxiliary functions needed by `move-alien` is paused to address a bug in our design
- Have you picked up on it?
- Consider carefully the following test:

```
(check-expect (move-alien INIT-ALIEN2 'down) MALIEN-VAL3-2)
```

- Does this test make sense?

Aliens Attack Version 2

Subtyping

- The development of the auxiliary functions needed by `move-alien` is paused to address a bug in our design
- Have you picked up on it?
- Consider carefully the following test:

```
(check-expect (move-alien INIT-ALIEN2 'down) MALIEN-VAL3-2)
```

- Does this test make sense?
- Recall that `INIT-ALIEN2`'s `image-y` coordinate is `MAX-IMG-Y`
- Can this alien be moved down?

Aliens Attack Version 2

Subtyping

- The development of the auxiliary functions needed by `move-alien` is paused to address a bug in our design
- Have you picked up on it?
- Consider carefully the following test:

```
(check-expect (move-alien INIT-ALIEN2 'down) MALIEN-VAL3-2)
```

- Does this test make sense?
- Recall that `INIT-ALIEN2`'s `image-y` coordinate is `MAX-IMG-Y`
- Can this alien be moved down?
- We are facing a situation where the test is suggesting that a function given a valid input value returns an invalid output value

Aliens Attack Version 2

Subtyping

- The development of the auxiliary functions needed by `move-alien` is paused to address a bug in our design
- Have you picked up on it?
- Consider carefully the following test:

  ```
  (check-expect (move-alien INIT-ALIEN2 'down) MALIEN-VAL3-2)
  ```

- Does this test make sense?
- Recall that `INIT-ALIEN2`'s `image-y` coordinate is `MAX-IMG-Y`
- Can this alien be moved down?
- We are facing a situation where the test is suggesting that a function given a valid input value returns an invalid output value
- The problem is that the auxiliary functions should not process an `image-x` value or an `image-y` value
- They need to process a subset of the values defined by these data types
- Need to define `image-x` and `image-y` *subtypes*

Aliens Attack Version 2

Subtyping

- The development of the auxiliary functions needed by `move-alien` is paused to address a bug in our design
- Have you picked up on it?
- Consider carefully the following test:

```
(check-expect (move-alien INIT-ALIEN2 'down) MALIEN-VAL3-2)
```

- Does this test make sense?
- Recall that `INIT-ALIEN2`'s `image-y` coordinate is `MAX-IMG-Y`
- Can this alien be moved down?
- We are facing a situation where the test is suggesting that a function given a valid input value returns an invalid output value
- The problem is that the auxiliary functions should not process an `image-x` value or an `image-y` value
- They need to process a subset of the values defined by these data types
- Need to define `image-x` and `image-y` *subtypes*
- A subtype defines a proper subset of the values of an existing type

Aliens Attack Version 2

Subtyping

- ```
  #|
  An image-x>min is an image-x in [(add1 MIN-IMG-X)..MAX-IMG-X]
  ```
  ```

Aliens Attack Version 2

Subtyping

- ```
  #|
  An image-x>min is an image-x in [(add1 MIN-IMG-X)..MAX-IMG-X]
  ```
- ```
  ;; image-x>min ... → ...
  ;; Purpose: ...
  (define (f-on-image-x an-img-x>min ...) ... an-img-x...)

  ;; Sample instances of image-x>min
  (define IMG-X>MIN1 ...) ...

  ;; Sample expressions for f-on-image-x>min
  (define IMGX>MIN-VAL1 ...) ...

  ;; Sample tests using sample computations for f-on-image-x>min
  (check-expect (f-on-image-x>min ...) IMGX>MIN-VAL1) ...

  ;; Sample tests using sample computations for f-on-image-x>min
  (check-expect (f-on-image-x>min ...) ...)   ...          |#
  ```

Aliens Attack Version 2

Subtyping

- `move-left-image-x` may be refined as follows:

```
;; image-x>min → image-x
;; Purpose: Move the given image-x>min left
(define (move-left-image-x an-img-x>min)
  (sub1 an-img-x>min))

;; Sample expressions for move-left-image-x
(define IMGX>MIN-VALL1 (sub1 AN-IMG-X))
(define IMGX>MIN-VALL2 (sub1 MAX-IMG-X))

;; Tests using sample computations for move-left-image-x
(check-expect (move-left-image-x AN-IMG-X)  IMGX>MIN-VALL1)
(check-expect (move-left-image-x MAX-IMG-X) IMGX>MIN-VALL2)

;; Tests using sample values for move-left-image-x
(check-expect (move-left-image-x 9) 8)
```

- `move-left-image-x` always returns an `image-x`

Aliens Attack Version 2

Subtyping

- ```
  #|
  An image-x<max, is an image-x in [MIN-IMG-X..(sub1 MAX-IMG-X)]

  ;; image-x<max ... → ...   Purpose: ...
  (define (f-on-image-x<max an-img-x<max ...)
    ... an-img-x...)

  ;; Sample instances of image-x<max
  (define IMG-X<MAX1 ...) ...

  ;; Sample expressions for f-on-image-x<max
  (define IMGX<MAX-VAL1 ...) ...

  ;; Tests using sample computations for f-on-image-x<max
  (check-expect (f-on-image-x<max ...) IMGX<MAX-VAL1) ...

  ;; Tests using sample computations for f-on-image-x<max
  (check-expect (f-on-image-x<max ...) ...) ...
  ```
  ```

Aliens Attack Version 2

Subtyping

- An image-y<max is an image-y in [MIN-IMG-Y..(sub1 MAX-IMG-Y)]

```
;; image-y<max ... → ...
;; Purpose: ...
(define (f-on-image-y<max an-img-y<max ...)
  ...~an-img-y<max...)

;; Sample instances of image-y<max
(define IMG-Y<MAX1 ...) ...

;; Sample expressions for f-on-image-y<max
(define IMGY<MAX-VAL1 ...) ...

;; Tests using sample computations for f-on-image-y<max
(check-expect (f-on-image-y<max ...) IMGY<MAX-VAL1) ...

;; Tests using sample values for f-on-image-y<max
(check-expect (f-on-image-y<max ...) ...)  ...    |#
```

Aliens Attack Version 2

Subtyping

- Creating refined data definitions for subtypes feels like a lot of extra work
- Time well spent
- It makes solutions to problems easier to understand

Aliens Attack Version 2

Subtyping

- Creating refined data definitions for subtypes feels like a lot of extra work
- Time well spent
- It makes solutions to problems easier to understand
- Important, because it makes it easier to refine and for others to work with your code
- If you had to maintain or refine another person's code, would you not prefer to have code that is easier to understand?
- This is a matter of ethics
- We need to always strive to develop the easiest code to understand and refine.

Aliens Attack Version 2

Checking Errors

- There are also cases where the programmer is unable to guarantee correct input (e.g., when the input to a function depends on the user of a program)
- What do you do in such cases?

Aliens Attack Version 2

Checking Errors

- There are also cases where the programmer is unable to guarantee correct input (e.g., when the input to a function depends on the user of a program)
- What do you do in such cases?
- One option is write *guarded* functions
- A guarded function uses a conditional expression to first check if the input is valid

Aliens Attack Version 2

Checking Errors

- There are also cases where the programmer is unable to guarantee correct input (e.g., when the input to a function depends on the user of a program)
- What do you do in such cases?
- One option is write *guarded* functions
- A guarded function uses a conditional expression to first check if the input is valid
- `BSL` provides the necessary syntax to generate and check errors
- An error always returns a string that ought to help understand error:

```
(error expr)
```

- Use `error` whenever an error must be thrown
- The `expr` must evaluate to a string

Aliens Attack Version 2

Checking Errors

- There are also cases where the programmer is unable to guarantee correct input (e.g., when the input to a function depends on the user of a program)
- What do you do in such cases?
- One option is write *guarded* functions
- A guarded function uses a conditional expression to first check if the input is valid
- `BSL` provides the necessary syntax to generate and check errors
- An error always returns a string that ought to help understand error:

```
(error expr)
```

- Use `error` whenever an error must be thrown
- The `expr` must evaluate to a string
- The syntax to test for an error is similar to `check-expect`:

```
(check-error expr expr)
```

Aliens Attack Version 2

Checking Errors

- BSL provides the `format` function to generate strings with customized information:

  `(format string expr*)`

- The given string may contain zero or more times the special character, ~s
- Each is substituted with the value of one of the expressions provided after the string
- The ith ~s is substituted with the value of the ith expression

Aliens Attack Version 2

Checking Errors

- ```
  ;; Sample expressions for move-right-image-x
  (define IMGX-VALR1 (add1 MIN-IMG-X))
  (define IMGX-VALR2 (add1 11))
  (define IMGX-VALRE (format "move-right-image-x: The character
                              at x=~s cannot move right."
                             MAX-IMG-X))
  ```
  ```

Aliens Attack Version 2

Checking Errors

- ```
  ;; image-x → image-x throws error
  ;; Purpose: Move the given image-x right
  (define (move-right-image-x an-img-x)
  ```

- ```
  ;; Sample expressions for move-right-image-x
  (define IMGX-VALR1 (add1 MIN-IMG-X))
  (define IMGX-VALR2 (add1 11))
  (define IMGX-VALRE (format "move-right-image-x: The character
                               at x=~s cannot move right."
                              MAX-IMG-X))
  ```

Aliens Attack Version 2

Checking Errors

- ```
  ;; image-x → image-x throws error
  ;; Purpose: Move the given image-x right
  (define (move-right-image-x an-img-x)
  ```

- ```
  ;; Sample expressions for move-right-image-x
  (define IMGX-VALR1 (add1 MIN-IMG-X))
  (define IMGX-VALR2 (add1 11))
  (define IMGX-VALRE (format "move-right-image-x: The character
                              at x=~s cannot move right."
                             MAX-IMG-X))
  ```

- ```
  ;; Tests using sample computations for move-right-image-x
  (check-expect (move-right-image-x MIN-IMG-X) IMGX-VALR1)
  (check-expect (move-right-image-x 11)        IMGX-VALR2)
  (check-error  (move-right-image-x MAX-IMG-X) IMGX-VALRE)

  ;; Tests using sample values for move-right-image-x
  (check-expect (move-right-image-x 12) 13)
  ```
  ```

Aliens Attack Version 2

Checking Errors

- ```
  ;; image-x → image-x throws error
  ;; Purpose: Move the given image-x right
  (define (move-right-image-x an-img-x)
  ```
- ```
    (if (< an-img-x MAX-IMG-X)
        (add1 an-img-x)
        (error (format "move-right-image-x: The character
                        at x=~s cannot move right."
                       MAX-IMG-X))))
  ```
- ```
  ;; Sample expressions for move-right-image-x
  (define IMGX-VALR1 (add1 MIN-IMG-X))
  (define IMGX-VALR2 (add1 11))
  (define IMGX-VALRE (format "move-right-image-x: The character
                              at x=~s cannot move right."
                             MAX-IMG-X))
  ```
- ```
  ;; Tests using sample computations for move-right-image-x
  (check-expect (move-right-image-x MIN-IMG-X) IMGX-VALR1)
  (check-expect (move-right-image-x 11)        IMGX-VALR2)
  (check-error  (move-right-image-x MAX-IMG-X) IMGX-VALRE)

  ;; Tests using sample values for move-right-image-x
  (check-expect (move-right-image-x 12) 13)
  ```

Aliens Attack Version 2

Checking Errors

- `move-alien` itself does not include code to raise an error but may still happen
- It happens when one of its auxiliary functions raises an error

Aliens Attack Version 2

Checking Errors

- `move-alien` itself does not include code to raise an error but may still happen
- It happens when one of its auxiliary functions raises an error
- This must be reflected in the signature for `move-alien`
- Sample expressions and tests must be more carefully designed
- Sample expressions may not use values, like `INIT-ALIEN2`, that generate an error
- Tests must check expected error messages from auxiliary functions using values that force throwing an error (like `INIT-ALIEN2`)

Aliens Attack Version 2

Checking Errors

```
;; alien dir → alien throws error Purpose: Move alien ...
(define (move-alien an-alien a-dir)
  (cond [(eq? a-dir 'right)
         (make-posn (move-right-image-x (posn-x an-alien)) (posn-y an-alien))]
        [(eq? a-dir 'left)
         (make-posn (move-left-image-x (posn-x an-alien)) (posn-y an-alien))]
        [else (make-posn (posn-x an-alien)
                         (move-down-image-y (posn-y an-alien)))]))
;; Sample expressions for move-alien
     ...
(define MALIEN-VAL3-2
        (make-posn (posn-x (make-posn 1 8))
                   (move-down-image-y (posn-y (make-posn 1 8)))))
;; Tests using sample computations for move-alien
     ...
(check-expect (move-alien (make-posn 1 8) 'down) MALIEN-VAL3-2)
;; Tests using sample values for move-alien
     ...
(check-error
 (move-alien INIT-ALIEN2 'down)
 (format "move-down-image-y: The character at y=~s cannot
          move down."
         MAX-IMG-Y))
(check-error
 (move-alien (make-posn 0 5) 'left)
 (format "move-left-image-x: The character at x=~s cannot
          move left."
         MIN-IMG-X))
(check-error
 (move-alien (make-posn MAX-IMG-X 15) 'right)
 (format "move-right-image-x: The character at x=~s cannot move right." MAX-IMG-X))
```

Aliens Attack Version 2

The `game-over?` Handler

- When does the game comes to an end?

Aliens Attack Version 2

The `game-over?` Handler

- When does the game comes to an end?
- The alien reaches earth
- Need to solve a problem about an alien
- How can an alien reaching earth be determined?
- Think it terms of the `alien` data definition.

Aliens Attack Version 2

The `game-over?` Handler

- When does the game comes to an end?
- The alien reaches earth
- Need to solve a problem about an alien
- How can an alien reaching earth be determined?
- Think it terms of the `alien` data definition.
- The alien's y coordinate is `MAX-IMG-Y`

Aliens Attack Version 2

The game-over? Handler

- ```
  ;; Sample expressions for game-over?
  (define GAME-OVER     (alien-reached-earth?
                          (world-alien INIT-WORLD2)))
  (define GAME-NOT-OVER (alien-reached-earth?
                          (world-alien INIT-WORLD)))
  ```
  ```

Aliens Attack Version 2

The game-over? Handler

- ```
  ;; world → Boolean
  ;; Purpose: Detect if the game is over
  (define (game-over? a-world)
  ```

- ```
  ;; Sample expressions for game-over?
  (define GAME-OVER     (alien-reached-earth?
                          (world-alien INIT-WORLD2)))
  (define GAME-NOT-OVER (alien-reached-earth?
                          (world-alien INIT-WORLD)))
  ```

Aliens Attack Version 2

The game-over? Handler

- ```
  ;; world → Boolean
  ;; Purpose: Detect if the game is over
  (define (game-over? a-world)
  ```

- ```
  ;; Sample expressions for game-over?
  (define GAME-OVER     (alien-reached-earth?
                          (world-alien INIT-WORLD2)))
  (define GAME-NOT-OVER (alien-reached-earth?
                          (world-alien INIT-WORLD)))
  ```

- ```
  ;; Tests using sample computations for game-over?
  (check-expect (game-over? INIT-WORLD2) GAME-OVER)
  (check-expect (game-over? INIT-WORLD)  GAME-NOT-OVER)

  ;; Tests using sample values for game-over?
  (check-expect (game-over? (make-world 8
                                        (make-posn 0 3)
                                        'right))
                #false)
  (check-expect (game-over? (make-world 8
                                        (make-posn 0 MAX-IMG-Y)
                                        'right))
               #true)
  ```
  ```

Aliens Attack Version 2

The game-over? Handler

- ```
  ;; world → Boolean
  ;; Purpose: Detect if the game is over
  (define (game-over? a-world)
  ```
- ```
    (alien-reached-earth? (world-alien a-world)))
  ```
- ```
  ;; Sample expressions for game-over?
  (define GAME-OVER     (alien-reached-earth?
                          (world-alien INIT-WORLD2)))
  (define GAME-NOT-OVER (alien-reached-earth?
                          (world-alien INIT-WORLD)))
  ```
- ```
  ;; Tests using sample computations for game-over?
  (check-expect (game-over? INIT-WORLD2) GAME-OVER)
  (check-expect (game-over? INIT-WORLD)  GAME-NOT-OVER)

  ;; Tests using sample values for game-over?
  (check-expect (game-over? (make-world 8
                                        (make-posn 0 3)
                                        'right))
                #false)
  (check-expect (game-over? (make-world 8
                                        (make-posn 0 MAX-IMG-Y)
                                        'right))
               #true)
  ```

Aliens Attack Version 2

The game-over? Handler

- ```
  ;; Sample expressions for alien-reached-earth?
  (define ALIEN-EARTH1 (= (posn-y INIT-ALIEN)  MAX-IMG-Y))
  (define ALIEN-EARTH2 (= (posn-y INIT-ALIEN2) MAX-IMG-Y))
  ```
  ```

Aliens Attack Version 2

The game-over? Handler

- ```
  ;; alien → Boolean
  ;; Purpose: Determine if the given alien reached earth
  (define (alien-reached-earth? an-alien)
  ```

- ```
  ;; Sample expressions for alien-reached-earth?
  (define ALIEN-EARTH1 (= (posn-y INIT-ALIEN)  MAX-IMG-Y))
  (define ALIEN-EARTH2 (= (posn-y INIT-ALIEN2) MAX-IMG-Y))
  ```

Aliens Attack Version 2

The game-over? Handler

- ```
  ;; alien → Boolean
  ;; Purpose: Determine if the given alien reached earth
  (define (alien-reached-earth? an-alien)
  ```

- ```
  ;; Sample expressions for alien-reached-earth?
  (define ALIEN-EARTH1 (= (posn-y INIT-ALIEN)  MAX-IMG-Y))
  (define ALIEN-EARTH2 (= (posn-y INIT-ALIEN2) MAX-IMG-Y))
  ```

- ```
  ;; Tests using sample computations for alien-reached-earth?
  (check-expect (alien-reached-earth? INIT-ALIEN)  ALIEN-EARTH1)
  (check-expect (alien-reached-earth? INIT-ALIEN2) ALIEN-EARTH2)

  ;; Tests using sample values for alien-reached-earth?
  (check-expect (alien-reached-earth? (make-posn 14 0)) #false)
  (check-expect (alien-reached-earth? (make-posn 9 MAX-IMG-Y))
                #true)
  ```
  ```

Aliens Attack Version 2

The game-over? Handler

- ```
  ;; alien → Boolean
  ;; Purpose: Determine if the given alien reached earth
  (define (alien-reached-earth? an-alien)
  ```
- ```
    (= (posn-y an-alien) MAX-IMG-Y))
  ```
- ```
  ;; Sample expressions for alien-reached-earth?
  (define ALIEN-EARTH1 (= (posn-y INIT-ALIEN)  MAX-IMG-Y))
  (define ALIEN-EARTH2 (= (posn-y INIT-ALIEN2) MAX-IMG-Y))
  ```
- ```
  ;; Tests using sample computations for alien-reached-earth?
  (check-expect (alien-reached-earth? INIT-ALIEN)  ALIEN-EARTH1)
  (check-expect (alien-reached-earth? INIT-ALIEN2) ALIEN-EARTH2)

  ;; Tests using sample values for alien-reached-earth?
  (check-expect (alien-reached-earth? (make-posn 14 0)) #false)
  (check-expect (alien-reached-earth? (make-posn 9 MAX-IMG-Y))
                #true)
  ```

Aliens Attack Version 2

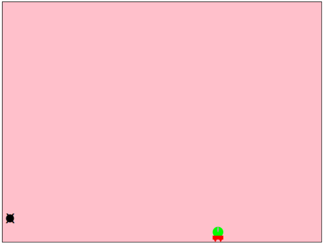
Computing the Last Scene

- Running the code reveals that all the tests pass. Hooray!

Aliens Attack Version 2

Computing the Last Scene

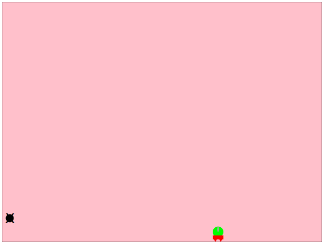
- Running the code reveals that all the tests pass. Hooray!
- Allowing the game to run until the alien reaches earth:



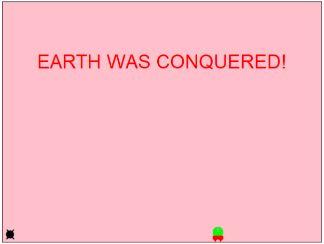
Aliens Attack Version 2

Computing the Last Scene

- Running the code reveals that all the tests pass. Hooray!
- Allowing the game to run until the alien reaches earth:



- Would like:



Aliens Attack Version 2

Computing the Last Scene

- The `universe` teachpack stops drawing the world when `game-over?` evaluates to `#true`

Aliens Attack Version 2

Computing the Last Scene

- The universe teachpack stops drawing the world when game-over? evaluates to #true
- To remedy this situation the universe API allows the programmer to specify a function to draw the last world
- This function is specified in the stop-when stanza of the big-bang expression:

```
; string → world
; Purpose: To run the game
(define (run a-name)
  (big-bang INIT-WORLD
            [on-draw draw-world]
            [name a-name]
            [on-key process-key]
            [on-tick process-tick TICK-RATE]
            [stop-when game-over? draw-last-world]))
```

Aliens Attack Version 2

Computing the Last Scene

- ```
  ;; world → scene throws error
  ;; Purpose: To draw the game's final scene
  (define (draw-last-world a-world)
   (if (= (posn-y (world-alien a-world)) MAX-IMG-Y)
       (place-image (text "EARTH WAS CONQUERED!" 36 'red)
                    (/ E-SCENE-W 2)
                    (/ E-SCENE-H 4)
                    (draw-world a-world))
       (error
        (format "draw-last-world: Invalid world with ~s as the
                alien's y coordinate. The alien's y coordinate
                must be in [~s..~s]."
               (posn-y (world-alien a-world))
               MIN-IMG-Y
               MAX-IMG-Y))))

  ;; Sample Instance of (final) world
  (define FWORLD1 (make-world 13 (make-posn  0 14) 'right))
  (define FWORLD2 (make-world  7 (make-posn 19 14) 'left))

  ;; Sample expressions for draw-last-world
  (define FWORLD1-VAL (place-image
                       (text "EARTH WAS CONQUERED!" 36 'red)
                       (/ E-SCENE-W 2)
                       (/ E-SCENE-H 4)
                       (draw-world FWORLD1)))
  (define FWORLD2-VAL (place-image
                       (text "EARTH WAS CONQUERED!" 36 'red)
                       (/ E-SCENE-W 2)
                       (/ E-SCENE-H 4)
                       (draw-world FWORLD2)))
  ```
  ```

Aliens Attack Version 2

Computing the Last Scene

- ```
  ;; Tests using sample computations for draw-last-world
  (check-expect (draw-last-world FWORLD1) FWORLD1-VAL)
  (check-expect (draw-last-world FWORLD2) FWORLD2-VAL)

  ;; Tests using sample values for draw-last-world
  (check-error
   (draw-last-world (make-world 10 (make-posn 7 20) 'right))
   "draw-last-world: Invalid world with 20 as the alien's y
    coordinate. The alien's y coordinate must be in [0..14].")
  ```
  ```

Aliens Attack Version 2

Homework

- Problem: 98

Structures and Variety

- Consider designing a function that processes motorized vehicles like cars, motorcycles, and personal transporters
- Consider designing a function that processes geometric shapes like squares, rectangles, triangles, and ellipses

Structures and Variety

- Consider designing a function that processes motorized vehicles like cars, motorcycles, and personal transporters
- Consider designing a function that processes geometric shapes like squares, rectangles, triangles, and ellipses
- The input may be an instance of a subtype

Structures and Variety

- Consider designing a function that processes motorized vehicles like cars, motorcycles, and personal transporters
- Consider designing a function that processes geometric shapes like squares, rectangles, triangles, and ellipses
- The input may be an instance of a subtype
- We shall explore the design a function to process motor vehicles
- The characteristics of the different motor vehicles are:

  Car Has a tank capacity in gallons, a miles per gallon, a maximum speed, and a mode. The mode indicates if the car is running in economic mode. When the car is running in economic mode it may travel 20% further than in normal mode, but the accelerator is less responsive.

  Motorcycle Has a tank capacity in gallons, a miles per gallon, and a maximum speed.

  Personal Transporter Has a maximum miles per hour and maximum hours per charge.

Structures and Variety

A Bottom-Up Design

- A bottom-up design starts with defining the different motor vehicles:

```
#|A car, carr, is a structure
  (make-carr integer>=0 integer>=0 integer>=0 integer>=0 Boolean)
containing tank capacity in gallons, miles per gallon, maximum
speed, and economy mode flag.
```

Structures and Variety

A Bottom-Up Design

- A bottom-up design starts with defining the different motor vehicles:

```
#|A car, carr, is a structure
  (make-carr integer>=0 integer>=0 integer>=0 integer>=0 Boolean)
containing tank capacity in gallons, miles per gallon, maximum
speed, and economy mode flag.
```

- ```
  ;; Sample instances of carr
  (define CARR1 ...)
  ;; carr ... → ...  Purpose: ...
  (define (f-on-carr a-carr ...)
    ...(carr-gallons a-carr)...(carr-mpg a-carr)
    ...(carr-maxspeed a-carr)...(carr-mode a-carr))
  ;; Sample expressions for f-on-carr
  (define CARR1-VAL ... CARR1 ...)  ...
  ;; Tests using sample computations for f-on-carr
  (check-expect (f-on-carr CARR1 ...) CARR1-VAL)  ...
  ;; Tests using sample values for f-on-carr
  (check-expect (f-on-carr ...) ...) ...
  ```
  ```

Structures and Variety

A Bottom-Up Design

- A bottom-up design starts with defining the different motor vehicles:

```
#|A car, carr, is a structure
  (make-carr integer>=0 integer>=0 integer>=0 integer>=0 Boolean)
containing tank capacity in gallons, miles per gallon, maximum
speed, and economy mode flag.
```

- ```
  ;; Sample instances of carr
  (define CARR1 ...)
  ;; carr ... → ...  Purpose: ...
  (define (f-on-carr a-carr ...)
    ...(carr-gallons a-carr)...(carr-mpg a-carr)
    ...(carr-maxspeed a-carr)...(carr-mode a-carr))
  ;; Sample expressions for f-on-carr
  (define CARR1-VAL ... CARR1 ...)  ...
  ;; Tests using sample computations for f-on-carr
  (check-expect (f-on-carr CARR1 ...) CARR1-VAL)  ...
  ;; Tests using sample values for f-on-carr
  (check-expect (f-on-carr ...) ...) ...
  ```

- ```
  ;; Structure Definition
  (define-struct carr (gallons mpg maxspeed mode))
  ;; Sample Instances of carr
  (define CARR1 (make-carr 20 35 140 #true))
  (define CARR2 (make-carr 22 15 160 #false))
  ```

Structures and Variety

A Bottom-Up Design

- ```
  #|   A motorcycle, mc, is a structure
         (make-mc integer>=0 integer>=0 integer>=0)
       with a tank capacity in gallons, miles per gallon, and a
       maximum speed.
  ```
  ```

Structures and Variety

A Bottom-Up Design

- ```
  #|   A motorcycle, mc, is a structure
         (make-mc integer>=0 integer>=0 integer>=0)
       with a tank capacity in gallons, miles per gallon, and a
       maximum speed.
  ```
- ```
       ;; Sample instances of mc
       (define MC1 ...)
       ;; mc ... → ...   Purpose: ...
       (define (f-on-mc an-mc ...)
         ...(mc-gallons an-mc)...(mc-mpg an-mc)
         ...(mc-maxspeed an-mc)...)
       ;; Sample expressions for f-on-mc
       (define MC1-VAL ... MC1 ...) ...
       ;; Tests using sample computations for f-on-mc
       (check-expect (f-on-mc MC1 ...) MC1-VAL)  ...
       ;; Tests using sample values for f-on-mc
       (check-expect (f-on-mc ...) ...) ...   |#
  ```

Structures and Variety

A Bottom-Up Design

- ```
  #|   A motorcycle, mc, is a structure
         (make-mc integer>=0 integer>=0 integer>=0)
       with a tank capacity in gallons, miles per gallon, and a
       maximum speed.
  ```
- ```
  ;; Sample instances of mc
  (define MC1 ...)
  ;; mc ... → ...   Purpose: ...
  (define (f-on-mc an-mc ...)
    ...(mc-gallons an-mc)...(mc-mpg an-mc)
    ...(mc-maxspeed an-mc)...)
  ;; Sample expressions for f-on-mc
  (define MC1-VAL ... MC1 ...) ...
  ;; Tests using sample computations for f-on-mc
  (check-expect (f-on-mc MC1 ...) MC1-VAL)  ...
  ;; Tests using sample values for f-on-mc
  (check-expect (f-on-mc ...) ...) ...   |#
  ```
- ```
  ;; Structure Definition
  (define-struct mc (gallons mpg maxspeed))
  ;; Sample instances of mc
  (define MC1   (make-mc   8  22 135))
  (define MC2   (make-mc   10 20 145))
  ```
  ```

Structures and Variety

A Bottom-Up Design

- ```
  #|   A personal transporter, pt, is a structure
        (make-pt integer>=0 integer>=0)
       with a maximum miles per hour and a maximum hours per
       charge.
  ```
  ```

Structures and Variety

A Bottom-Up Design

- ```
  #|   A personal transporter, pt, is a structure
        (make-pt integer>=0 integer>=0)
       with a maximum miles per hour and a maximum hours per
       charge.
  ```
- ```
  ;; Sample instances of pt
  (define PT1 ...)
  ;; pt ... → ...   Purpose: ...
  (define (f-on-pt a-pt ...)
    ...(pt-mph a-pt)...(pt-hpc a-pt))
  ;; Sample expressions for f-on-pt
  (define PT1-VAL ... PT1 ...)  ...
  ;; Tests using sample computations for f-on-pt
  (check-expect (f-on-pt PT1 ...) PT1-VAL)  ...
  ;; Tests using sample values for f-on-pt
  (check-expect (f-on-pt ...) ...)  ...   |#
  ```

Structures and Variety

A Bottom-Up Design

- ```
  #|   A personal transporter, pt, is a structure
        (make-pt integer>=0 integer>=0)
       with a maximum miles per hour and a maximum hours per
       charge.
  ```
- ```
  ;; Sample instances of pt
  (define PT1 ...)
  ;; pt ... → ...   Purpose: ...
  (define (f-on-pt a-pt ...)
    ...(pt-mph a-pt)...(pt-hpc a-pt))
  ;; Sample expressions for f-on-pt
  (define PT1-VAL ... PT1 ...)  ...
  ;; Tests using sample computations for f-on-pt
  (check-expect (f-on-pt PT1 ...) PT1-VAL)  ...
  ;; Tests using sample values for f-on-pt
  (check-expect (f-on-pt ...) ...)  ...   |#
  ```
- ```
  ;; Structure Definition
  (define-struct pt (mph hpc))
  ;; Sample instances of pt
  (define PT1   (make-pt 7  7))
  (define PT2   (make-pt 6 10))
  ```
  ```

Structures and Variety

A Bottom-Up Design

- Once subtypes are defined, the next step is to define a *union type*
- A union type enumerates the type varieties that may be used to build instances

Structures and Variety

A Bottom-Up Design

- Once subtypes are defined, the next step is to define a *union type*
- A union type enumerates the type varieties that may be used to build instances
- If `A`, `B`, and `C` are types then `D` may be defined as the union of `A`, `B`, and `C`
- `A`, `B`, and `C` are subtypes of `D`

Structures and Variety

A Bottom-Up Design

- Once subtypes are defined, the next step is to define a *union type*
- A union type enumerates the type varieties that may be used to build instances
- If `A`, `B`, and `C` are types then `D` may be defined as the union of `A`, `B`, and `C`
- `A`, `B`, and `C` are subtypes of `D`
- We say that `D` is `A`'s, `B`'s, and `C`'s *supertype*
- Functions written for a supertype must be able to process instances of each of its subtypes

Structures and Variety

A Bottom-Up Design

- Once subtypes are defined, the next step is to define a *union type*
- A union type enumerates the type varieties that may be used to build instances
- If `A`, `B`, and `C` are types then `D` may be defined as the union of `A`, `B`, and `C`
- `A`, `B`, and `C` are subtypes of `D`
- We say that `D` is `A`'s, `B`'s, and `C`'s *supertype*
- Functions written for a supertype must be able to process instances of each of its subtypes
- A function on a supertype must be *polymorphic*
- Polymorphic means that a function can process different types of data.

Structures and Variety

A Bottom-Up Design

- The supertype mv:

```
#|   A motor vehicle, mv, is either:
       1. carr  2. mc  3. pt
```

Structures and Variety

A Bottom-Up Design

- The supertype mv:

```
#|   A motor vehicle, mv, is either:
        1. carr  2. mc  3. pt
```

- ```
;; Sample instances of mv
(define MVCARR1 ...) (define MVMC1 ...) (define MVPT1 ...)
;; mv ... → ...   Purpose: ...
(define (f-on-mv an-mv ...)
  (cond [(carr? an-mv) (f-on-car an-mv ...)]
        [(mc? an-mv) (f-on-mc an-mv ...)]
        [else (f-on-mv an-mv ...)]))
;; Sample expressions for f-on-mv
(define MVCARR1-VAL ... MVCARR1 ...)
(define MVMC1-VAL ... MVMC1 ...)
(define MVPT1-VAL ... MVPT1 ...) ...
;; Tests using sample computations for f-on-mv
(check-expect (f-on-mv MVCARR1 ...) MVCARR1-VAL)
(check-expect (f-on-mv MVMC1 ...) MVMC1-VAL)
(check-expect (f-on-mv MVPT1 ...) MVPT1-VAL) ...
;; Tests using sample values for f-on-mv
(check-expect (f-on-mv ...) ...) ... |#
```
```

Structures and Variety

A Bottom-Up Design

- The supertype mv:

```
#|   A motor vehicle, mv, is either:
        1. carr  2. mc  3. pt
```

- ```
  ;; Sample instances of mv
  (define MVCARR1 ...) (define MVMC1  ...) (define MVPT1  ...)
  ;; mv ... → ...   Purpose: ...
  (define (f-on-mv an-mv ...)
    (cond [(carr? an-mv) (f-on-car an-mv ...)]
          [(mc? an-mv) (f-on-mc  an-mv ...)]
          [else (f-on-mv an-mv ...)]))
  ;; Sample expressions for f-on-mv
  (define MVCARR1-VAL ... MVCARR1 ...)
  (define MVMC1-VAL  ... MVMC1 ...)
  (define MVPT1-VAL  ... MVPT1 ...)  ...
  ;; Tests using sample computations for f-on-mv
  (check-expect (f-on-mv MVCARR1 ...) MVCARR1-VAL)
  (check-expect (f-on-mv MVMC1 ...)  MVMC1-VAL)
  (check-expect (f-on-mv MVPT1 ...)  MVPT1-VAL)  ...
  ;; Tests using sample values for f-on-mv
  (check-expect (f-on-mv ...) ...) ...       |#
  ```

- ```
  ;; Sample instances of mv
  (define MVCARR1 CARR1) (define MVCARR2 CARR2) (define MVMC1 MC1)
  (define MVMC2   MC2)   (define MVPT1   PT1)   (define MVPT2 PT2)
  ```

Structures and Variety

A Bottom-Up Design

- Consider writing a function to compute the maximum distance a motor vehicle may travel

Structures and Variety

A Bottom-Up Design

- Consider writing a function to compute the maximum distance a motor vehicle may travel
- Bottom-up approach: first functions to compute the maximum distance for each subtype

Structures and Variety

A Bottom-Up Design

- ```
  ;; mc → integer>=0
  ;; Purpose: Return max distance the  given mc may travel on full
  (define (mc-maxdist an-mc)
    (* (mc-gallons an-mc) (mc-mpg an-mc)))

  ;; Sample expressions for mc-maxdist
  (define MC1-VAL (* (mc-gallons  MC1) (mc-mpg  MC1)))
  (define MC2-VAL (* (mc-gallons  MC2) (mc-mpg  MC2)))

  ;; Tests using sample computations for mc-maxdist
  (check-expect (mc-maxdist MC1) MC1-VAL)
  (check-expect (mc-maxdist MC1) MC1-VAL)

  ;; Tests using sample values for mc-maxdist
  (check-expect (mc-maxdist (make-mc 10 10 120)) 100)
  (check-expect (mc-maxdist (make-mc 8  15 105)) 120)
  ```
  ```

Structures and Variety

A Bottom-Up Design

- ```
  ;; pt → integer>=0
  ;; Purpose: Return max distance the given pt may travel on full
  (define (pt-maxdist a-pt)
    (* (pt-mph a-pt) (pt-hpc a-pt)))

  ;; Sample expressions for f-on-pt
  (define PT1-VAL (* (pt-mph PT1) (pt-hpc PT1)))
  (define PT2-VAL (* (pt-mph PT2) (pt-hpc PT2)))

  ;; Tests using sample computations for pt-maxdist
  (check-expect (pt-maxdist PT1) PT1-VAL)
  (check-expect (pt-maxdist PT2) PT2-VAL)

  ;; Tests using sample values for pt-maxdist
  (check-expect (pt-maxdist (make-pt 4 5)) 20)
  ```
  ```

Structures and Variety

A Bottom-Up Design

- ```
  ;; carr → number>=0
  ;; Purpose: Return max distance the given carr may travel on full
  (define (carr-maxdist a-carr)
    (if (carr-mode a-carr)
        (* 1.2 (carr-gallons a-carr) (carr-mpg a-carr))
        (* (carr-gallons a-carr) (carr-mpg a-carr))))
  ;; Sample expressions for f-on-carr
  (define CARR1-VAL (* 1.2 (carr-gallons CARR1) (carr-mpg CARR1)))
  (define CARR2-VAL (* (carr-gallons  CARR2) (carr-mpg  CARR2)))

  ;; Tests using sample computations
  ;; for carr-maxdist
  (check-expect (carr-maxdist CARR1) CARR1-VAL)
  (check-expect (carr-maxdist CARR2) CARR2-VAL)
  ;; Tests using sample values
  ;; for carr-maxdist
  (check-expect (carr-maxdist (make-carr 9 21 10 #true))  226.8)
  (check-expect (carr-maxdist (make-carr 9 21 10 #false)) 189)
  ```
  ```

Structures and Variety

A Bottom-Up Design

- After defining the functions for the `mv` subtypes, the bottom-up design process continues with the design of the function for the supertype: `mv-maxdist`

Structures and Variety

A Bottom-Up Design

- After defining the functions for the `mv` subtypes, the bottom-up design process continues with the design of the function for the supertype: `mv-maxdist`
- What is `mv-maxdist`'s return type?
- Processing an `mc` or a `pt` returns an `integer>=0`
- Processing a `carr` returns a `number>=0`

Structures and Variety

A Bottom-Up Design

- After defining the functions for the mv subtypes, the bottom-up design process continues with the design of the function for the supertype: `mv-maxdist`
- What is `mv-maxdist`'s return type?
- Processing an `mc` or a `pt` returns an `integer>=0`
- Processing a `carr` returns a `number>=0`
- An obvious solution is to create a new union type data definition:

```
A distance (dist) is either:
1. integer>=0
2. number>=0
```

Structures and Variety

A Bottom-Up Design

- After defining the functions for the `mv` subtypes, the bottom-up design process continues with the design of the function for the supertype: `mv-maxdist`
- What is `mv-maxdist`'s return type?
- Processing an `mc` or a `pt` returns an `integer>=0`
- Processing a `carr` returns a `number>=0`
- An obvious solution is to create a new union type data definition:

```
A distance (dist) is either:
1. integer>=0
2. number>=0
```

- Carefully think about this data definition
- `integer>=0` is a subtype of `number>=0`

Structures and Variety

A Bottom-Up Design

- After defining the functions for the `mv` subtypes, the bottom-up design process continues with the design of the function for the supertype: `mv-maxdist`
- What is `mv-maxdist`'s return type?
- Processing an `mc` or a `pt` returns an `integer>=0`
- Processing a `carr` returns a `number>=0`
- An obvious solution is to create a new union type data definition:

```
A distance (dist) is either:
1. integer>=0
2. number>=0
```

- Carefully think about this data definition
- `integer>=0` is a subtype of `number>=0`
- The value returned by `mv-maxdist` is always a `number>=0`
- Given that for this problem there is no need to distinguish between the varieties the data definition for `dist` is not needed
- We can safely claim that `mv-maxdist` returns a `number>=0`

Structures and Variety

A Bottom-Up Design

- ```
  ;; Sample expressions for mv-maxdist
  (define MVCARR1-VAL (carr-maxdist MVCARR1))
  (define MVCARR2-VAL (carr-maxdist MVCARR2))
  (define MVMC1-VAL   (mc-maxdist   MVMC1))
  (define MVPT1-VAL   (pt-maxdist   MVPT1))
  ```
  ```

Structures and Variety

A Bottom-Up Design

- ```
  ;; mv → number>=0
  ;; Purpose: Return max distance the given mv may travel on full
  (define (mv-maxdist amv)
  ```

- ```
  ;; Sample expressions for mv-maxdist
  (define MVCARR1-VAL (carr-maxdist MVCARR1))
  (define MVCARR2-VAL (carr-maxdist MVCARR2))
  (define MVMC1-VAL   (mc-maxdist   MVMC1))
  (define MVPT1-VAL   (pt-maxdist   MVPT1))
  ```

Structures and Variety

A Bottom-Up Design

- ```
  ;; mv → number>=0
  ;; Purpose: Return max distance the given mv may travel on full
  (define (mv-maxdist amv)
  ```

- ```
  ;; Sample expressions for mv-maxdist
  (define MVCARR1-VAL (carr-maxdist MVCARR1))
  (define MVCARR2-VAL (carr-maxdist MVCARR2))
  (define MVMC1-VAL   (mc-maxdist   MVMC1))
  (define MVPT1-VAL   (pt-maxdist   MVPT1))
  ```

- ```
  ;; Tests using sample computations
  (check-expect (mv-maxdist MVCARR1) MVCARR1-VAL)
  (check-expect (mv-maxdist MVCARR2) MVCARR2-VAL)
  (check-expect (mv-maxdist MVMC1)   MVMC1-VAL)
  (check-expect (mv-maxdist MVPT1)   MVPT1-VAL)

  ;; Tests using sample values
  (check-expect (mv-maxdist (make-carr 25 40 140 #true))  1200)
  (check-expect (mv-maxdist (make-carr 25 40 140 #false)) 1000)
  ```
  ```

Structures and Variety

A Bottom-Up Design

- ```
  ;; mv → number>=0
  ;; Purpose: Return max distance the given mv may travel on full
  (define (mv-maxdist amv)
  ```
- ```
    (cond
      [(carr? amv) (carr-maxdist amv)]
      [(mc? amv)   (mc-maxdist  amv)]
      [else (pt-maxdist amv)]))
  ```
- ```
  ;; Sample expressions for mv-maxdist
  (define MVCARR1-VAL (carr-maxdist MVCARR1))
  (define MVCARR2-VAL (carr-maxdist MVCARR2))
  (define MVMC1-VAL   (mc-maxdist   MVMC1))
  (define MVPT1-VAL   (pt-maxdist   MVPT1))
  ```
- ```
  ;; Tests using sample computations
  (check-expect (mv-maxdist MVCARR1) MVCARR1-VAL)
  (check-expect (mv-maxdist MVCARR2) MVCARR2-VAL)
  (check-expect (mv-maxdist MVMC1)   MVMC1-VAL)
  (check-expect (mv-maxdist MVPT1)   MVPT1-VAL)

  ;; Tests using sample values
  (check-expect (mv-maxdist (make-carr 25 40 140 #true))  1200)
  (check-expect (mv-maxdist (make-carr 25 40 140 #false)) 1000)
  ```

Structures and Variety

Homework

- Problems: 101–103

Structures and Variety

Code Refactoring

- It is possible to do better:

```
(* 1.2 (carr-gallons a-carr) (carr-mpg a-carr)) ;;in mv-maxdist
(*     (carr-gallons a-carr) (carr-mpg a-carr)) ;;in mv-maxdist
(*     (mc-gallons an-mc)    (mc-mpg an-mc))    ;;in mc-maxdist
(*     (pt-hpc a-pt)         (pt-mph a-pt))     ;;in pt-maxdist
```

- These expressions all look very similar
- The last three expressions are structurally the same
- Suggests that abstraction over expressions ought to be used to create a single function to compute these values

Structures and Variety

Code Refactoring

- It is possible to do better:

```
(* 1.2 (carr-gallons a-carr) (carr-mpg a-carr)) ;;in mv-maxdist
(*     (carr-gallons a-carr) (carr-mpg a-carr)) ;;in mv-maxdist
(*     (mc-gallons an-mc)    (mc-mpg an-mc))    ;;in mc-maxdist
(*     (pt-hpc a-pt)         (pt-mph a-pt))     ;;in pt-maxdist
```

- These expressions all look very similar
- The last three expressions are structurally the same
- Suggests that abstraction over expressions ought to be used to create a single function to compute these values
- But, what about the first expression?
- It is structurally different from the other expressions

Structures and Variety

Code Refactoring

- It is possible to do better:

```
(* 1.2 (carr-gallons a-carr) (carr-mpg a-carr)) ;;in mv-maxdist
(*     (carr-gallons a-carr) (carr-mpg a-carr)) ;;in mv-maxdist
(*     (mc-gallons an-mc)    (mc-mpg an-mc))    ;;in mc-maxdist
(*     (pt-hpc a-pt)         (pt-mph a-pt))     ;;in pt-maxdist
```

- These expressions all look very similar
- The last three expressions are structurally the same
- Suggests that abstraction over expressions ought to be used to create a single function to compute these values
- But, what about the first expression?
- It is structurally different from the other expressions
- In such a case, consider using a technique called *code refactoring*
- Code refactoring restructures existing expressions (or functions) without changing their external behavior
- Expressions may be restructured to look the same and, therefore, allow for abstraction over similar expressions

Structures and Variety

Code Refactoring

- It is possible to do better:

```
(* 1.2 (carr-gallons a-carr) (carr-mpg a-carr)) ;;in mv-maxdist
(*     (carr-gallons a-carr) (carr-mpg a-carr)) ;;in mv-maxdist
(*     (mc-gallons an-mc)    (mc-mpg an-mc))    ;;in mc-maxdist
(*     (pt-hpc a-pt)         (pt-mph a-pt))     ;;in pt-maxdist
```

- These expressions all look very similar
- The last three expressions are structurally the same
- Suggests that abstraction over expressions ought to be used to create a single function to compute these values
- But, what about the first expression?
- It is structurally different from the other expressions
- In such a case, consider using a technique called *code refactoring*
- Code refactoring restructures existing expressions (or functions) without changing their external behavior
- Expressions may be restructured to look the same and, therefore, allow for abstraction over similar expressions
- The first expression above has, 1.2, a constant of proportionality not present in the other three expressions
- Refactor the last three expressions:

```
(* 1.2 (carr-gallons a-carr) (carr-mpg a-carr))
(* 1   (carr-gallons a-carr) (carr-mpg a-carr))
(* 1   (mc-gallons an-mc)    (mc-mpg an-mc))
(* 1   (pt-hpc a-pt)         (pt-mph a-pt))
```

Structures and Variety

Code Refactoring

- ```
  ;; Sample expressions for refactored-helper
  (define DIST1 (* 1.2 (carr-gallons CARR1) (carr-mpg CARR1)))
  (define DIST2 (* 1   (carr-gallons CARR2) (carr-mpg CARR2)))
  (define DIST3 (* 1   (mc-gallons MC1)     (mc-mpg MC1)))
  (define DIST4 (* 1   (pt-hpc PT1)         (pt-mph PT1)))
  ```
  ```

Structures and Variety

Code Refactoring

- ```
  ;; number>=0 integer>=0 integer>=0 --> number>=0
  ;; Purpose: Compute maximum distance
  (define (refactored-helper k energy-units miles/energy-unit)
  ```

- ```
  ;; Sample expressions for refactored-helper
  (define DIST1 (* 1.2 (carr-gallons CARR1) (carr-mpg CARR1)))
  (define DIST2 (* 1   (carr-gallons CARR2) (carr-mpg CARR2)))
  (define DIST3 (* 1   (mc-gallons MC1)     (mc-mpg MC1)))
  (define DIST4 (* 1   (pt-hpc PT1)         (pt-mph PT1)))
  ```

Structures and Variety

Code Refactoring

- ```
  ;; number>=0 integer>=0 integer>=0 --> number>=0
  ;; Purpose: Compute maximum distance
  (define (refactored-helper k energy-units miles/energy-unit)
  ```

- ```
  ;; Sample expressions for refactored-helper
  (define DIST1 (* 1.2 (carr-gallons CARR1) (carr-mpg CARR1)))
  (define DIST2 (* 1   (carr-gallons CARR2) (carr-mpg CARR2)))
  (define DIST3 (* 1   (mc-gallons MC1)     (mc-mpg MC1)))
  (define DIST4 (* 1   (pt-hpc PT1)         (pt-mph PT1)))
  ```

- ```
  ;; Tests using sample computations for refactored-helper
  (check-expect
   (refactored-helper 1.2 (carr-gallons CARR1) (carr-mpg CARR1))
   DIST1)
  (check-expect
   (refactored-helper 1 (carr-gallons CARR2) (carr-mpg CARR2)) DIST2)
  (check-expect
   (refactored-helper 1 (mc-gallons MC1) (mc-mpg MC1)) DIST3)
  (check-expect
   (refactored-helper 1 (pt-hpc PT1) (pt-mph PT1)) DIST4)
  ;; Tests using sample values for refactored-helper
  (check-expect (refactored-helper 1.2 11 30) 396)
  ```
  ```

Structures and Variety

Code Refactoring

- ```
  ;; number>=0 integer>=0 integer>=0 --> number>=0
  ;; Purpose: Compute maximum distance
  (define (refactored-helper k energy-units miles/energy-unit)
  ```
- ```
    (*  k energy-units miles/energy-unit))
  ```
- ```
  ;; Sample expressions for refactored-helper
  (define DIST1 (* 1.2 (carr-gallons CARR1) (carr-mpg CARR1)))
  (define DIST2 (* 1   (carr-gallons CARR2) (carr-mpg CARR2)))
  (define DIST3 (* 1   (mc-gallons MC1)     (mc-mpg MC1)))
  (define DIST4 (* 1   (pt-hpc PT1)         (pt-mph PT1)))
  ```
- ```
  ;; Tests using sample computations for refactored-helper
  (check-expect
   (refactored-helper 1.2 (carr-gallons CARR1) (carr-mpg CARR1))
   DIST1)
  (check-expect
   (refactored-helper 1 (carr-gallons CARR2) (carr-mpg CARR2)) DIST2)
  (check-expect
   (refactored-helper 1 (mc-gallons MC1) (mc-mpg MC1)) DIST3)
  (check-expect
   (refactored-helper 1 (pt-hpc PT1) (pt-mph PT1)) DIST4)
  ;; Tests using sample values for refactored-helper
  (check-expect (refactored-helper 1.2 11 30) 396)
  ```

Structures and Variety

Code Refactoring

- ```
  ;; Sample expressions for mv-maxdistance
  (define MVCARR1-VAL (refactored-helper 1.2 (carr-gallons MVCARR1) (carr-mpg MVCARR1)))
  (define MVCARR2-VAL (refactored-helper 1 (carr-gallons MVCARR2) (carr-mpg MVCARR2)))
  (define MVMC2-VAL   (refactored-helper 1 (mc-gallons MVMC2) (mc-mpg MVMC2)))
  (define MVPT1-VAL   (refactored-helper 1 (pt-hpc MVPT1) (pt-mph MVPT1)))
  ```
  ```

Structures and Variety

Code Refactoring

- ```
  ;; mv → number>=0
  ;; Purpose: Return max distance mv may travel
  (define (mv-maxdistance an-mv)
  ```

- ```
  ;; Sample expressions for mv-maxdistance
  (define MVCARR1-VAL (refactored-helper 1.2 (carr-gallons MVCARR1) (carr-mpg MVCARR1)))
  (define MVCARR2-VAL (refactored-helper 1 (carr-gallons MVCARR2) (carr-mpg MVCARR2)))
  (define MVMC2-VAL   (refactored-helper 1 (mc-gallons MVMC2) (mc-mpg MVMC2)))
  (define MVPT1-VAL   (refactored-helper 1 (pt-hpc MVPT1) (pt-mph MVPT1)))
  ```

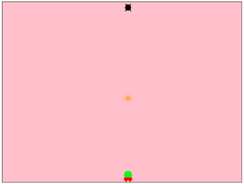
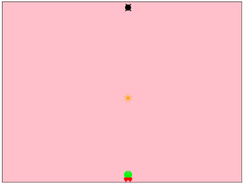
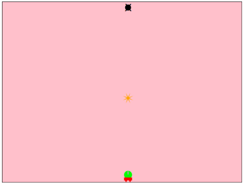
Structures and Variety

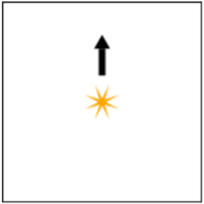
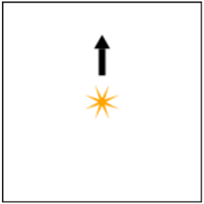
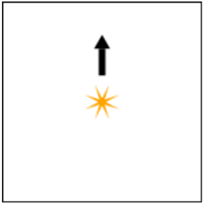
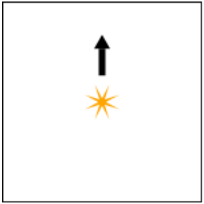
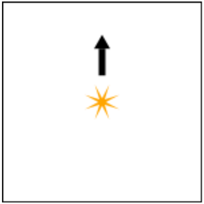
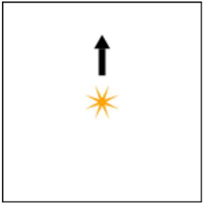
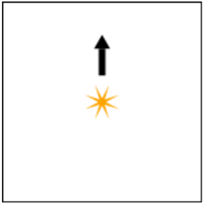
Code Refactoring

- ```
  ;; mv → number>=0
  ;; Purpose: Return max distance mv may travel
  (define (mv-maxdistance an-mv)
  ```

- ```
  ;; Sample expressions for mv-maxdistance
  (define MVCARR1-VAL (refactored-helper 1.2 (carr-gallons MVCARR1) (carr-mpg MVCARR1)))
  (define MVCARR2-VAL (refactored-helper 1 (carr-gallons MVCARR2) (carr-mpg MVCARR2)))
  (define MVMC2-VAL   (refactored-helper 1 (mc-gallons MVMC2) (mc-mpg MVMC2)))
  (define MVPT1-VAL   (refactored-helper 1 (pt-hpc MVPT1) (pt-mph MVPT1)))
  ```

- ```
  ;; Tests using sample computations for mv-maxdistance
  (check-expect (mv-maxdist MVCARR1) MVCARR1-VAL)
  (check-expect (mv-maxdist MVCARR2) MVCARR2-VAL)  ...

  ;; Tests using sample values for mv-maxdistance
  (check-expect (mv-maxdist (make-carr 25 40 140 #true))  1200 )
  (check-expect (mv-maxdist (make-carr 25 40 140 #false)) 1000 ) ...
  ```
  ```

Structures and Variety

Code Refactoring

- ```
  ;; mv → number>=0
  ;; Purpose: Return max distance mv may travel
  (define (mv-maxdistance an-mv)
  ```
- ```
   (cond
    [(carr? an-mv)
     (if (carr-mode an-mv)
       (refactored-helper 1.2 (carr-gallons an-mv) (carr-mpg an-mv))
       (refactored-helper 1   (carr-gallons an-mv) (carr-mpg an-mv)))]
    [(mc? an-mv)
     (refactored-helper 1 (mc-gallons an-mv) (mc-mpg an-mv))]
    [else (refactored-helper 1 (pt-hpc an-mv) (pt-mph an-mv))]))
  ```
- ```
  ;; Sample expressions for mv-maxdistance
  (define MVCARR1-VAL (refactored-helper 1.2 (carr-gallons MVCARR1) (carr-mpg MVCARR1)))
  (define MVCARR2-VAL (refactored-helper 1 (carr-gallons MVCARR2) (carr-mpg MVCARR2)))
  (define MVMC2-VAL   (refactored-helper 1 (mc-gallons MVMC2) (mc-mpg MVMC2)))
  (define MVPT1-VAL   (refactored-helper 1 (pt-hpc MVPT1) (pt-mph MVPT1)))
  ```
- ```
  ;; Tests using sample computations for mv-maxdistance
  (check-expect (mv-maxdist MVCARR1) MVCARR1-VAL)
  (check-expect (mv-maxdist MVCARR2) MVCARR2-VAL)  ...

  ;; Tests using sample values for mv-maxdistance
  (check-expect (mv-maxdist (make-carr 25 40 140 #true))  1200 )
  (check-expect (mv-maxdist (make-carr 25 40 140 #false)) 1000 ) ...
  ```

Structures and Variety

Code Refactoring

- Code refactoring has significantly reduced the size of the code
- The original code's four functions are reduced to two
- Code refactoring plays a central role in software development
- It is used to improve design, structure, and efficiency without changing software functionality

Structures and Variety

Homework

- Problems: 106–106

Aliens Attack Version 3

- Writing union types that contain structure-based subtypes endows us with the power to refine Aliens Attack version 2
- The world data definition is now refined to contain a shot

Aliens Attack Version 3

- Writing union types that contain structure-based subtypes endows us with the power to refine Aliens Attack version 2
- The world data definition is now refined to contain a shot
- A shot is part of the `world` means the player can win the game
- When does this happen?

Aliens Attack Version 3

- Writing union types that contain structure-based subtypes endows us with the power to refine Aliens Attack version 2
- The world data definition is now refined to contain a shot
- A shot is part of the `world` means the player can win the game
- When does this happen?
- Clearly, when the alien is hit by the shot
- When does that happen?

Aliens Attack Version 3

- Writing union types that contain structure-based subtypes endows us with the power to refine Aliens Attack version 2
- The world data definition is now refined to contain a shot
- A shot is part of the `world` means the player can win the game
- When does this happen?
- Clearly, when the alien is hit by the shot
- When does that happen?
- When the coordinates of the alien and the shot are the same

Aliens Attack Version 3

- Writing union types that contain structure-based subtypes endows us with the power to refine Aliens Attack version 2
- The world data definition is now refined to contain a shot
- A shot is part of the `world` means the player can win the game
- When does this happen?
- Clearly, when the alien is hit by the shot
- When does that happen?
- When the coordinates of the alien and the shot are the same
- Consider an alien at `(0 7)`, a shot at `(0 6)`, and the direction being `'down`
- This puts the alien and the shot at the left edge of the scene
- Alien moves to coordinate `(0 6)` and the shot moves to coordinates `(0 7)`
- Has the alien been hit?

Aliens Attack Version 3

- Writing union types that contain structure-based subtypes endows us with the power to refine Aliens Attack version 2
- The world data definition is now refined to contain a shot
- A shot is part of the `world` means the player can win the game
- When does this happen?
- Clearly, when the alien is hit by the shot
- When does that happen?
- When the coordinates of the alien and the shot are the same
- Consider an alien at `(0 7)`, a shot at `(0 6)`, and the direction being `'down`
- This puts the alien and the shot at the left edge of the scene
- Alien moves to coordinate `(0 6)` and the shot moves to coordinates `(0 7)`
- Has the alien been hit?
- On the one hand, the alien and the shot are never the same
- On the other hand, the alien and the shot cross each other

Aliens Attack Version 3

- Writing union types that contain structure-based subtypes endows us with the power to refine Aliens Attack version 2
- The world data definition is now refined to contain a shot
- A shot is part of the `world` means the player can win the game
- When does this happen?
- Clearly, when the alien is hit by the shot
- When does that happen?
- When the coordinates of the alien and the shot are the same
- Consider an alien at `(0 7)`, a shot at `(0 6)`, and the direction being `'down`
- This puts the alien and the shot at the left edge of the scene
- Alien moves to coordinate `(0 6)` and the shot moves to coordinates `(0 7)`
- Has the alien been hit?
- On the one hand, the alien and the shot are never the same
- On the other hand, the alien and the shot cross each other
- This is a situation that forces game designer to choose a feature
- If it is not a hit it makes the game a bit more challenging
- If it is a hit it makes the game a bit easier for the to win

Aliens Attack Version 3

- Writing union types that contain structure-based subtypes endows us with the power to refine Aliens Attack version 2
- The world data definition is now refined to contain a shot
- A shot is part of the `world` means the player can win the game
- When does this happen?
- Clearly, when the alien is hit by the shot
- When does that happen?
- When the coordinates of the alien and the shot are the same
- Consider an alien at `(0 7)`, a shot at `(0 6)`, and the direction being `'down`
- This puts the alien and the shot at the left edge of the scene
- Alien moves to coordinate `(0 6)` and the shot moves to coordinates `(0 7)`
- Has the alien been hit?
- On the one hand, the alien and the shot are never the same
- On the other hand, the alien and the shot cross each other
- This is a situation that forces game designer to choose a feature
- If it is not a hit it makes the game a bit more challenging
- If it is a hit it makes the game a bit easier for the to win
- Our choice: An alien moving down “sees" the shot coming and skittles beneath it to avoid the hit.

Aliens Attack Version 3

- Adding a shot to the world means that `world`'s function template and `world` sample instances must also be refined
- In addition, any functions and tests that build a world must be updated

Aliens Attack Version 3

- Adding a shot to the world means that `world`'s function template and `world` sample instances must also be refined
- In addition, any functions and tests that build a world must be updated
- Must define a shot and design of new functions

Aliens Attack Version 3

Data Definitions

- 

Aliens Attack Version 3

Data Definitions

- 
- `#| A world is a structure: (make-world rocket alien dir shot)`

Aliens Attack Version 3

Data Definitions

- 
- #| A world is a structure: (make-world rocket alien dir shot)
-
```
  ;; world ... → ...   ;; Purpose: ...
  (define (f-on-world a-world ...)
     ...(f-on-rocket (world-rocket a-world) ...)
     ...(f-on-alien  (world-alien  a-world) ...)
     ...(f-on-dir    (world-dir a-world)    ...)
     ...(f-on-shot   (world-shot a-world)   ...))
  ;; Sample instances for world
  (define WORLD1 (make-world ...))
  ;; Sample expressions for f-on-world
  (define WORLD-VAL1 ...)

  ;; Tests using sample computations for f-on-world
  (check-expect (f-on-world WORLD1 ...) WORLD-VAL1) ...
  ;; Tests using sample values for f-on-world
  (check-expect (f-on-world ...) ...)                            #|
```

Aliens Attack Version 3

Data Definitions

- Think about what a shot is
- A player uses a key, say the space bar key to shoot
- A shot moves as follows:



- What may change from one instance on a shot to another?

Aliens Attack Version 3

Data Definitions

- Think about what a shot is
- A player uses a key, say the space bar key to shoot
- A shot moves as follows:



- What may change from one instance on a shot to another?
- An `image-y` coordinate is needed to represent a shot that moves up

Aliens Attack Version 3

Data Definitions

- Think about what a shot is
- A player uses a key, say the space bar key to shoot
- A shot moves as follows:



- What may change from one instance on a shot to another?
- An `image-y` coordinate is needed to represent a shot that moves up
- The `image-x` coordinate may vary among shots: determined by the position of the rocket when the shot is created

Aliens Attack Version 3

Data Definitions

- Think about what a shot is
- A player uses a key, say the space bar key to shoot
- A shot moves as follows:



- What may change from one instance on a shot to another?
- An `image-y` coordinate is needed to represent a shot that moves up
- The `image-x` coordinate may vary among shots: determined by the position of the rocket when the shot is created
- Using a `posn` to represent a shot is a good option

Aliens Attack Version 3

Data Definitions

- Think about what a shot is
- A player uses a key, say the space bar key to shoot
- A shot moves as follows:



- What may change from one instance on a shot to another?
- An `image-y` coordinate is needed to represent a shot that moves up
- The `image-x` coordinate may vary among shots: determined by the position of the rocket when the shot is created
- Using a `posn` to represent a shot is a good option
- What `posn` should be used for the shot at the beginning of the game?

Aliens Attack Version 3

Data Definitions

- Think about what a shot is
- A player uses a key, say the space bar key to shoot
- A shot moves as follows:



- What may change from one instance on a shot to another?
- An `image-y` coordinate is needed to represent a shot that moves up
- The `image-x` coordinate may vary among shots: determined by the position of the rocket when the shot is created
- Using a `posn` to represent a shot is a good option
- What `posn` should be used for the shot at the beginning of the game?
- This is an interesting question because the player has not created a shot
- If a shot has not been created, how can it have `image-x` and `image-y` coordinates?

Aliens Attack Version 3

Data Definitions

- Think about what a shot is
- A player uses a key, say the space bar key to shoot
- A shot moves as follows:



- What may change from one instance on a shot to another?
- An `image-y` coordinate is needed to represent a shot that moves up
- The `image-x` coordinate may vary among shots: determined by the position of the rocket when the shot is created
- Using a `posn` to represent a shot is a good option
- What `posn` should be used for the shot at the beginning of the game?
- This is an interesting question because the player has not created a shot
- If a shot has not been created, how can it have `image-x` and `image-y` coordinates?
- Suggests there is variety in `shots`: non-existing and existing shots

Aliens Attack Version 3

Data Definitions

- `(define NO-SHOT 'no-shot)`

Aliens Attack Version 3

Data Definitions

- ```
  (define NO-SHOT 'no-shot)
  ```
- ```
  #|A shot is either:
     1. NO-SHOT
     2. A structure (make-posn image-x image-y)
  ;; Sample instances of shot
  (define SHOT1 ...)    (define SHOT2 ...)
  ```

Aliens Attack Version 3

Data Definitions

- `(define NO-SHOT 'no-shot)`
- ```
  #|A shot is either:
     1. NO-SHOT
     2. A structure (make-posn image-x image-y)
  ;; Sample instances of shot
  (define SHOT1 ...)    (define SHOT2 ...)
  ```
- ```
  ;; shot ... → ...    Purpose: ...
  (define (f-on-shot a-shot ...)
    (if (eq? a-shot NO-SHOT)
        ...
        ...(f-on-image-x (posn-x a-shot))
        ...(f-on-image-y (posn-y a-shot))...)))

  ;; Sample expressions for f-on-shot
  (define SHOT1-VAL ...)
  (define SHOT2-VAL ...) ...

  ;; Tests using sample computations for f-on-shot
  (check-expect (f-on-shot SHOT1 ...) SHOT1-VAL)
  (check-expect (f-on-shot SHOT2 ...) SHOT2-VAL) ...
  ;; Tests using sample values for f-on-shot
  (check-expect (f-on-shot ...) ...) ...        |#
  ```

Aliens Attack Version 3

Data Definitions

- Sample instances:

```
;; Sample instances of shot
(define INIT-SHOT NO-SHOT)
(define SHOT2 (make-posn (/ MAX-CHARS-HORIZONTAL 2)
                         (/ (sub1 MAX-CHARS-VERTICAL) 2)))
(define SHOT3 (make-posn 4 MAX-IMG-Y))
(define SHOT4 (make-posn 14 MIN-IMG-Y))
```

Aliens Attack Version 3

Data Definitions

- Sample instances:

```
;; Sample instances of shot
(define INIT-SHOT NO-SHOT)
(define SHOT2 (make-posn (/ MAX-CHARS-HORIZONTAL 2)
                         (/ (sub1 MAX-CHARS-VERTICAL) 2)))
(define SHOT3 (make-posn 4 MAX-IMG-Y))
(define SHOT4 (make-posn 14 MIN-IMG-Y))
```

-
```
;; Sample instances of world
(define INIT-WORLD  (make-world INIT-ROCKET INIT-ALIEN
                                INIT-DIR    INIT-SHOT))
(define INIT-WORLD2 (make-world INIT-ROCKET2 INIT-ALIEN2
                                DIR2         SHOT2))
(define WORLD3 (make-world 7      (make-posn 3 3)
                           'right (make-posn 3 3)))
```

Aliens Attack Version 3

The `draw-world` Refinement

- In addition to rendering the `rocket` and the `alien`, the `shot` must be rendered
- The only decision that needs to be made is whether the `alien` or the `shot` is rendered on top when a hit occurs

Aliens Attack Version 3

The `draw-world` Refinement

- In addition to rendering the `rocket` and the `alien`, the `shot` must be rendered
- The only decision that needs to be made is whether the `alien` or the `shot` is rendered on top when a hit occurs
- This is mostly a matter of personal preferences
- In order to clearly see that a hit has occurred, let's render the `shot` on top of the `alien`
- This means that the `shot` must be rendered in a `scene` that contains the `alien`.
- All that is needed is function composition to draw the `shot` in a scene with a rocket and an alien

Aliens Attack Version 3

The draw-world Refinement

- ```
  ;; Sample expressions for draw-world
  (define WORLD-SCN1
          (draw-shot (world-shot INIT-WORLD)
                     (draw-alien (world-alien INIT-WORLD)
                                 (draw-rocket (world-rocket INIT-WORLD) E-SCENE))))
  (define WORLD-SCN2
          (draw-shot (world-shot INIT-WORLD2)
                     (draw-alien (world-alien INIT-WORLD2)
                                 (draw-rocket (world-rocket INIT-WORLD2) E-SCENE))))
  ```
  ```

Aliens Attack Version 3

The draw-world Refinement

- ```
  ;; world → scene   Purpose: To draw the world in E-SCENE
  (define (draw-world a-world)
  ```

- ```
  ;; Sample expressions for draw-world
  (define WORLD-SCN1
          (draw-shot (world-shot INIT-WORLD)
                     (draw-alien (world-alien INIT-WORLD)
                                 (draw-rocket (world-rocket INIT-WORLD) E-SCENE))))
  (define WORLD-SCN2
          (draw-shot (world-shot INIT-WORLD2)
                     (draw-alien (world-alien INIT-WORLD2)
                                 (draw-rocket (world-rocket INIT-WORLD2) E-SCENE))))
  ```

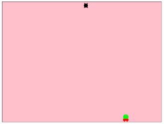
Aliens Attack Version 3

The draw-world Refinement

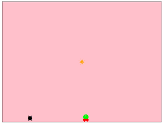
```
;; world → scene   Purpose: To draw the world in E-SCENE
(define (draw-world a-world)
```

```
;; Sample expressions for draw-world
(define WORLD-SCN1
        (draw-shot (world-shot INIT-WORLD)
                   (draw-alien (world-alien INIT-WORLD)
                               (draw-rocket (world-rocket INIT-WORLD) E-SCENE))))
(define WORLD-SCN2
        (draw-shot (world-shot INIT-WORLD2)
                   (draw-alien (world-alien INIT-WORLD2)
                               (draw-rocket (world-rocket INIT-WORLD2) E-SCENE))))
```

```
;; Tests using sample values for draw-world
(check-expect
  (draw-world (make-world INIT-ROCKET2 INIT-ALIEN INIT-DIR3 NO-SHOT))
```



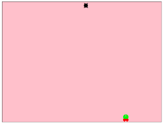
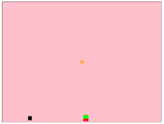
```
  )
(check-expect
  (draw-world (make-world INIT-ROCKET INIT-ALIEN2 INIT-DIR2   SHOT2))
```



```
  )
```

Aliens Attack Version 3

The draw-world Refinement

- ```
  ;; world → scene   Purpose: To draw the world in E-SCENE
  (define (draw-world a-world)
  ```
- ```
    (draw-shot (world-shot a-world)
               (draw-alien (world-alien a-world)
                           (draw-rocket (world-rocket a-world) E-SCENE))))
  ```
- ```
  ;; Sample expressions for draw-world
  (define WORLD-SCN1
          (draw-shot (world-shot INIT-WORLD)
                     (draw-alien (world-alien INIT-WORLD)
                                 (draw-rocket (world-rocket INIT-WORLD) E-SCENE))))
  (define WORLD-SCN2
          (draw-shot (world-shot INIT-WORLD2)
                     (draw-alien (world-alien INIT-WORLD2)
                                 (draw-rocket (world-rocket INIT-WORLD2) E-SCENE))))
  ```
- ```
  ;; Tests using sample values for draw-world
  (check-expect
    (draw-world (make-world INIT-ROCKET2 INIT-ALIEN INIT-DIR3 NO-SHOT))
  ```
  
  ```
  )
  (check-expect
    (draw-world (make-world INIT-ROCKET INIT-ALIEN2 INIT-DIR2   SHOT2))
  ```
  
  ```
  )
  ```

Aliens Attack Version 3

The `draw-world` Refinement

- Next: design the auxiliary function to draw a `shot`
- Template for a function on a `shot` is specialized

Aliens Attack Version 3

The `draw-world` Refinement

- Next: design the auxiliary function to draw a `shot`
- Template for a function on a `shot` is specialized
- There are two varieties of `shot`
- Think about each variety independently

Aliens Attack Version 3

The `draw-world` Refinement

- Next: design the auxiliary function to draw a `shot`
- Template for a function on a `shot` is specialized
- There are two varieties of `shot`
- Think about each variety independently
- If the given `shot` is `'NO-SHOT` then there is nothing to draw in the given scene

Aliens Attack Version 3

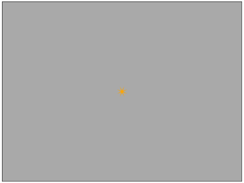
The `draw-world` Refinement

- Next: design the auxiliary function to draw a `shot`
- Template for a function on a `shot` is specialized
- There are two varieties of `shot`
- Think about each variety independently
- If the given `shot` is `'NO-SHOT` then there is nothing to draw in the given scene
- If the given `shot` is a `posn` then the `shot` must be drawn at the image coordinates contained in the `shot`

Aliens Attack Version 3

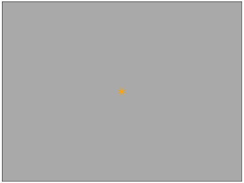
The draw-world Refinement

- ```
  ;; Sample expressions for draw-shot
  (define INIT-SHOT-VAL E-SCENE)
  (define SHOT2-VAL (draw-ci SHOT-IMG (posn-x SHOT2) (posn-y SHOT2) E-SCENE))
  (define SHOT3-VAL (draw-ci SHOT-IMG (posn-x SHOT3) (posn-y SHOT3) E-SCENE2))
  ```
  ```

Aliens Attack Version 3

The draw-world Refinement

- ```
  ;; shot scene → scene
  ;; Purpose: To draw the shot in the given scene
  (define (draw-shot a-shot scn)
  ```

- ```
  ;; Sample expressions for draw-shot
  (define INIT-SHOT-VAL E-SCENE)
  (define SHOT2-VAL (draw-ci SHOT-IMG (posn-x SHOT2) (posn-y SHOT2) E-SCENE))
  (define SHOT3-VAL (draw-ci SHOT-IMG (posn-x SHOT3) (posn-y SHOT3) E-SCENE2))
  ```

Aliens Attack Version 3

The draw-world Refinement

- ```
  ;; shot scene → scene
  ;; Purpose: To draw the shot in the given scene
  (define (draw-shot a-shot scn)
  ```

- ```
  ;; Sample expressions for draw-shot
  (define INIT-SHOT-VAL E-SCENE)
  (define SHOT2-VAL (draw-ci SHOT-IMG (posn-x SHOT2) (posn-y SHOT2) E-SCENE))
  (define SHOT3-VAL (draw-ci SHOT-IMG (posn-x SHOT3) (posn-y SHOT3) E-SCENE2))
  ```

- ```
  ;; Tests using sample computations for draw-shot
  (check-expect (draw-shot INIT-SHOT E-SCENE)  INIT-SHOT-VAL)
  (check-expect (draw-shot SHOT2     E-SCENE)  SHOT2-VAL)
  (check-expect (draw-shot SHOT3     E-SCENE2) SHOT3-VAL)

  ;; Tests using sample values for draw-shot
  (check-expect (draw-shot INIT-SHOT E-SCENE2) E-SCENE2)
  (check-expect (draw-shot SHOT2 E-SCENE2)
  ```

  

)
  ```

Aliens Attack Version 3

The draw-world Refinement

- ```
  ;; shot scene → scene
  ;; Purpose: To draw the shot in the given scene
  (define (draw-shot a-shot scn)
  ```
- ```
    (if (eq? a-shot NO-SHOT)
        scn
      (draw-ci SHOT-IMG (posn-x a-shot) (posn-y a-shot) scn)))
  ```
- ```
  ;; Sample expressions for draw-shot
  (define INIT-SHOT-VAL E-SCENE)
  (define SHOT2-VAL (draw-ci SHOT-IMG (posn-x SHOT2) (posn-y SHOT2) E-SCENE))
  (define SHOT3-VAL (draw-ci SHOT-IMG (posn-x SHOT3) (posn-y SHOT3) E-SCENE2))
  ```
- ```
  ;; Tests using sample computations for draw-shot
  (check-expect (draw-shot INIT-SHOT E-SCENE)  INIT-SHOT-VAL)
  (check-expect (draw-shot SHOT2     E-SCENE)  SHOT2-VAL)
  (check-expect (draw-shot SHOT3     E-SCENE2) SHOT3-VAL)

  ;; Tests using sample values for draw-shot
  (check-expect (draw-shot INIT-SHOT E-SCENE2) E-SCENE2)
  (check-expect (draw-shot SHOT2 E-SCENE2)
  ```
  
)

Aliens Attack Version 3

The `process-key` Refinement

- The key event handler must be updated to construct `worlds` that contain a `shot`
- In addition, this function must also process the key event associated with shooting

Aliens Attack Version 3

The `process-key` Refinement

- The key event handler must be updated to construct `worlds` that contain a `shot`
- In addition, this function must also process the key event associated with shooting
- The key choice is arbitrary: we use the space key

Aliens Attack Version 3

The `process-key` Refinement

- The key event handler must be updated to construct `worlds` that contain a `shot`
- In addition, this function must also process the key event associated with shooting
- The key choice is arbitrary: we use the space key
- The `key` data definition and the template for a function on a `key` must be refined

Aliens Attack Version 3

The process-key Refinement

- A key is either:
  1. "right" 2. "left" 3. " " 4. not "right", "left", or " "

Aliens Attack Version 3

The process-key Refinement

- A key is either:
  1. "right" 2. "left" 3. " " 4. not "right", "left", or " "
- ```
  ;; Sample instances of key
  (define KEY1 ...)  (define KEY2 ...)
  (define KEY3 ...)  (define KEY4 ...)
  ```
  ```

Aliens Attack Version 3

The process-key Refinement

- A key is either:
  1. "right" 2. "left" 3. " " 4. not "right", "left", or " "
- ```
  ;; Sample instances of key
  (define KEY1 ...)  (define KEY2 ...)
  (define KEY3 ...)  (define KEY4 ...)
  ```
- ```
  ;; key ... → ...   Purpose: ...
  (define (f-on-key a-key ...)
    (cond [(key=? a-key "right") ...]
          [(key=? a-key "left") ...]
          [(key=? a-key " ") ...]
          [else ...]))
  ;; Sample expressions for f-on-key
  (define KEY1-VAL ... KEY1 ...)
  (define KEY2-VAL ... KEY2 ...)
  (define KEY3-VAL ... KEY3 ...)
  (define KEY4-VAL ... KEY4 ...)
  ;; Tests using sample computations for f-on-key
  (check-expect (f-on-key KEY1 ...) KEY1-VAL)
  (check-expect (f-on-key KEY2 ...) KEY2-VAL)
  (check-expect (f-on-key KEY3 ...) KEY3-VAL)
  (check-expect (f-on-key KEY4 ...) KEY4-VAL)  ...
  ;; Tests using sample values for f-on-key
  (check-expect (f-on-key ...) ...)  ...
  ```

Aliens Attack Version 3

The `process-key` Refinement

- When should a shot be created?

Aliens Attack Version 3

The `process-key` Refinement

- When should a shot be created?
- There can only be at most one shot in a `world`
- A new shot cannot be created every time the player tries to shoot
- A decision must be made as to whether to construct a new `shot` or not

Aliens Attack Version 3

The `process-key` Refinement

- When should a shot be created?
- There can only be at most one shot in a `world`
- A new shot cannot be created every time the player tries to shoot
- A decision must be made as to whether to construct a new `shot` or not
- The details of how this is done is left to an auxiliary function, but there are two cases
- If there is no `shot` in the game then a new `shot` is created and added to the world
- Otherwise, a new `shot` is not created and the `shot` in the world is left unchanged

Aliens Attack Version 3

The process-key Refinement

- ```
  ;; Sample instances of key
  (define KEY1 "right")
  (define KEY2 "left")
  (define KEY3 "m")
  (define KEY4 " ")
  ```
  ```

Aliens Attack Version 3

The process-key Refinement

- ```
  ;; Sample instances of key
  (define KEY1 "right")
  (define KEY2 "left")
  (define KEY3 "m")
  (define KEY4 " ")
  ```
- ```
  ;; Sample expressions for process-key
  (define KEY-RVAL  (make-world (move-rckt-right
                                  (world-rocket INIT-WORLD))
                                  (world-alien INIT-WORLD)
                                  (world-dir INIT-WORLD)
                                  (world-shot INIT-WORLD)))
  (define KEY-LVAL  (make-world (move-rckt-left
                                  (world-rocket INIT-WORLD))
                                  (world-alien INIT-WORLD)
                                  (world-dir INIT-WORLD)
                                  (world-shot INIT-WORLD)))
  (define KEY-SVAL  (make-world (world-rocket INIT-WORLD)
                                  (world-alien INIT-WORLD)
                                  (world-dir INIT-WORLD)
                                  (process-shooting
                                    (world-shot INIT-WORLD)
                                    (world-rocket INIT-WORLD))))
  (define KEY-SVAL2 (make-world (world-rocket INIT-WORLD2)
                                (world-alien INIT-WORLD2)
                                (world-dir INIT-WORLD2)
                                (process-shooting
                                  (world-shot INIT-WORLD2)
                                  (world-rocket INIT-WORLD2))))
  (define KEY-OVAL INIT-WORLD2)
  ```

Aliens Attack Version 3

The process-key Refinement

- ```
  ;; world key → world
  ;; Purpose: Process a key event to return next world
  (define (process-key a-world a-key)
  ```
  ```

Aliens Attack Version 3

The process-key Refinement

- ```
  ;; world key → world
  ;; Purpose: Process a key event to return next world
  (define (process-key a-world a-key)
  ```
- ```
    (cond [(key=? a-key "right")
           (make-world (move-rckt-right (world-rocket a-world))
                       (world-alien a-world)
                       (world-dir a-world)
                       (world-shot a-world))]
          [(key=? a-key "left")
           (make-world (move-rckt-left (world-rocket a-world))
                       (world-alien a-world)
                       (world-dir a-world)
                       (world-shot a-world))]
          [(key=? a-key " ")
           (make-world (world-rocket a-world)
                       (world-alien a-world)
                       (world-dir a-world)
                       (process-shooting
                         (world-shot a-world)
                         (world-rocket a-world)))]
          [else a-world]))
  ```

Aliens Attack Version 3

The process-key Refinement

- ```
  ;; Tests using sample computations for process-key
  (check-expect (process-key INIT-WORLD  "right") KEY-RVAL)
  (check-expect (process-key INIT-WORLD  "left")  KEY-LVAL)
  (check-expect (process-key INIT-WORLD2 "m")     KEY-OVAL)
  (check-expect (process-key INIT-WORLD  " ") KEY-SVAL)
  (check-expect (process-key INIT-WORLD2 " ") KEY-SVAL2)

  ;; Tests using sample values for process-key
  (check-expect (process-key (make-world
                               (sub1 MAX-CHARS-HORIZONTAL)
                               INIT-ALIEN
                               'right
                               INIT-SHOT)
                              "right")
                (make-world (sub1 MAX-CHARS-HORIZONTAL)
                            INIT-ALIEN
                            'right
                            INIT-SHOT))
  (check-expect (process-key (make-world 0
                                         INIT-ALIEN
                                         'left
                                         INIT-SHOT)
                              " ")
                (make-world 0
                            INIT-ALIEN
                            'left
                            (make-posn 0 MAX-IMG-Y)))
  ```
  ```

Aliens Attack Version 3

The `process-key` Refinement

- Design of the auxiliary function to process the space key

Aliens Attack Version 3

The `process-key` Refinement

- Design of the auxiliary function to process the space key
- Inputs: a `shot` and a `rocket`
- Which template ought to be used to design this function?

Aliens Attack Version 3

The `process-key` Refinement

- Design of the auxiliary function to process the space key
- Inputs: a `shot` and a `rocket`
- Which template ought to be used to design this function?
- A new shot is created only when the game's `shot` is `'NO-SHOT`
- Suggests designing this function by specializing the template for a function on a `shot` because its body distinguishes among `shot` varieties
- Can this function be designed by specializing the template for a function on a `rocket`?

Aliens Attack Version 3

The `process-key` Refinement

- Design of the auxiliary function to process the space key
- Inputs: a `shot` and a `rocket`
- Which template ought to be used to design this function?
- A new shot is created only when the game's `shot` is `'NO-SHOT`
- Suggests designing this function by specializing the template for a function on a `shot` because its body distinguishes among `shot` varieties
- Can this function be designed by specializing the template for a function on a `rocket`?
- If you think carefully about this the answer is no
- The given rocket does not have enough information to decide whether or not a `shot` ought to be created
- This is a situation in which one input dominates another input
- When facing such a problem design around the dominating input

Aliens Attack Version 3

The process-key Refinement

- ```
  ;; Sample expressions for process-shooting
  (define PS-SHOT-VAL1 (make-posn INIT-ROCKET MAX-IMG-Y))
  (define PS-SHOT2-VAL SHOT2)
  (define PS-SHOT3-VAL SHOT3)
  ```
  ```

Aliens Attack Version 3

The process-key Refinement

- ```
  ;; shot rocket → shot
  ;; Purpose: To process a shooting attempt
  (define (process-shooting a-shot a-rocket)
  ```

- ```
  ;; Sample expressions for process-shooting
  (define PS-SHOT-VAL1 (make-posn INIT-ROCKET MAX-IMG-Y))
  (define PS-SHOT2-VAL SHOT2)
  (define PS-SHOT3-VAL SHOT3)
  ```

Aliens Attack Version 3

The process-key Refinement

- ```
  ;; shot rocket → shot
  ;; Purpose: To process a shooting attempt
  (define (process-shooting a-shot a-rocket)
  ```

- ```
  ;; Sample expressions for process-shooting
  (define PS-SHOT-VAL1 (make-posn INIT-ROCKET MAX-IMG-Y))
  (define PS-SHOT2-VAL SHOT2)
  (define PS-SHOT3-VAL SHOT3)
  ```
- ```
  ;; Tests using sample computations for process-shooting
  (check-expect (process-shooting INIT-SHOT INIT-ROCKET)
                PS-SHOT-VAL1)
  (check-expect (process-shooting SHOT2 INIT-ROCKET)
                PS-SHOT2-VAL)
  (check-expect (process-shooting SHOT3 INIT-ROCKET2)
                PS-SHOT3-VAL)

  ;; Tests using sample values for process-shooting
  (check-expect (process-shooting NO-SHOT 8)
                (make-posn 8 MAX-IMG-Y))
  (check-expect (process-shooting (make-posn 17 9) 8)
                (make-posn 17 9))
  ```
  ```

Aliens Attack Version 3

The process-key Refinement

- ```
  ;; shot rocket → shot
  ;; Purpose: To process a shooting attempt
  (define (process-shooting a-shot a-rocket)
  ```
- ```
    (if (eq? a-shot NO-SHOT)
        (make-posn a-rocket MAX-IMG-Y)
        a-shot))
  ```
- ```
  ;; Sample expressions for process-shooting
  (define PS-SHOT-VAL1 (make-posn INIT-ROCKET MAX-IMG-Y))
  (define PS-SHOT2-VAL SHOT2)
  (define PS-SHOT3-VAL SHOT3)
  ```
- ```
  ;; Tests using sample computations for process-shooting
  (check-expect (process-shooting INIT-SHOT INIT-ROCKET)
                PS-SHOT-VAL1)
  (check-expect (process-shooting SHOT2 INIT-ROCKET)
                PS-SHOT2-VAL)
  (check-expect (process-shooting SHOT3 INIT-ROCKET2)
                PS-SHOT3-VAL)

  ;; Tests using sample values for process-shooting
  (check-expect (process-shooting NO-SHOT 8)
                (make-posn 8 MAX-IMG-Y))
  (check-expect (process-shooting (make-posn 17 9) 8)
                (make-posn 17 9))
  ```

Aliens Attack Version 3

Homework

- Problems: 107–108

Aliens Attack Version 3

The `process-tick` Refinement

- Create a new world by moving the alien and computing a direction like before
- Does a shot change after a clock tick?

Aliens Attack Version 3

The `process-tick` Refinement

- Create a new world by moving the alien and computing a direction like before
- Does a shot change after a clock tick?
- Once again, reason about each `shot` variety

Aliens Attack Version 3

The `process-tick` Refinement

- Create a new world by moving the alien and computing a direction like before
- Does a shot change after a clock tick?
- Once again, reason about each `shot` variety
- If the `shot` is `'NO-SHOT` then it does not change

Aliens Attack Version 3

The `process-tick` Refinement

- Create a new world by moving the alien and computing a direction like before
- Does a shot change after a clock tick?
- Once again, reason about each `shot` variety
- If the `shot` is `'NO-SHOT` then it does not change
- If the shot is a `posn` then it either must move up or disappear because it has reached the top of the `scene`
- The details of how the new `shot` is computed is left to an auxiliary function that processes a `shot`

Aliens Attack Version 3

The process-tick Refinement

- ```
  ;; Sample expressions for process-tick
  (define AFTER-TICK-WORLD1
          (make-world
            (world-rocket INIT-WORLD)
            (move-alien (world-alien INIT-WORLD)
                        (world-dir INIT-WORLD))
            (new-dir-after-tick (move-alien
                                  (world-alien INIT-WORLD)
                                  (world-dir INIT-WORLD))
                                (world-dir INIT-WORLD))
            (move-shot (world-shot INIT-WORLD))))

  (define AFTER-TICK-WORLD2
          (make-world
            (world-rocket INIT-WORLD2)
            (move-alien (world-alien INIT-WORLD2)
                        (world-dir INIT-WORLD2))
            (new-dir-after-tick (move-alien
                                  (world-alien INIT-WORLD2)
                                  (world-dir INIT-WORLD2))
                                (world-dir INIT-WORLD2))
            (move-shot (world-shot INIT-WORLD2))))
  ```
  ```

Aliens Attack Version 3

The process-tick Refinement

- ```
  ;; world → world
  ;; Purpose: Create a new world after a clock tick
  (define (process-tick a-world)
    (make-world (world-rocket a-world)
                (move-alien (world-alien a-world)
                            (world-dir a-world))
                (new-dir-after-tick (move-alien
                                      (world-alien a-world)
                                      (world-dir a-world))
                                    (world-dir a-world))
                (move-shot (world-shot a-world))))
  ```
  ```

Aliens Attack Version 3

The process-tick Refinement

- ```
  ;; Tests using sample values for process-tick
  (check-expect
   (process-tick (make-world
                   INIT-ROCKET (make-posn 1 5) 'left INIT-SHOT))
   (make-world INIT-ROCKET (make-posn MIN-IMG-X 5) 'down INIT-SHOT))

  (check-expect
   (process-tick (make-world INIT-ROCKET2
                             (make-posn (- MAX-CHARS-HORIZONTAL 2) 10)
                             'right
                             SHOT2))
   (make-world
     INIT-ROCKET2 (make-posn MAX-IMG-X 10) 'down (move-shot SHOT2)))

  (check-expect
   (process-tick
     (make-world
       INIT-ROCKET2 (make-posn MAX-IMG-X 2) 'down (make-posn 15 6)))
     (make-world
       INIT-ROCKET2 (make-posn MAX-IMG-X 3) 'left (make-posn 15 5)))

  (check-expect
    (process-tick
      (make-world INIT-ROCKET2 (make-posn MIN-IMG-X 2) 'down (make-posn 2 MIN-IMG-Y)))
      (make-world INIT-ROCKET2 (make-posn MIN-IMG-X 3) 'right NO-SHOT))
  ```
  ```

Aliens Attack Version 3

The `move-shot` Design

- Writing tests for `process-tick` has provided an insight into how to move a shot: a `posn shot` is moved by decreasing its `image-y` coordinate
- How is a `'NO-SHOT shot` moved?

Aliens Attack Version 3

The `move-shot` Design

- Writing tests for `process-tick` has provided an insight into how to move a shot: a `posn shot` is moved by decreasing its `image-y` coordinate
- How is a `'NO-SHOT shot` moved?
- If there is no `shot` in the game then it remains unchanged
- Think about how a `posn shot` is moved
- Does a `shot` always move up?

Aliens Attack Version 3

The move-shot Design

- Writing tests for process-tick has provided an insight into how to move a shot: a posn shot is moved by decreasing its image-y coordinate
- How is a 'NO-SHOT shot moved?
- If there is no shot in the game then it remains unchanged
- Think about how a posn shot is moved
- Does a shot always move up?
- Allow the player to shoot again when a shot goes off the top of the scene
- How does this affect how a shot is moved?

Aliens Attack Version 3

The `move-shot` Design

- Writing tests for `process-tick` has provided an insight into how to move a shot: a `posn` `shot` is moved by decreasing its `image-y` coordinate
- How is a `'NO-SHOT` `shot` moved?
- If there is no `shot` in the game then it remains unchanged
- Think about how a `posn` `shot` is moved
- Does a `shot` always move up?
- Allow the player to shoot again when a `shot` goes off the top of the scene
- How does this affect how a `shot` is moved?
- Must become a `'NO-SHOT` `shot`

Aliens Attack Version 3

The move-shot Design

- Writing tests for `process-tick` has provided an insight into how to move a shot: a `posn shot` is moved by decreasing its `image-y` coordinate
- How is a `'NO-SHOT shot` moved?
- If there is no `shot` in the game then it remains unchanged
- Think about how a `posn shot` is moved
- Does a `shot` always move up?
- Allow the player to shoot again when a `shot` goes off the top of the scene
- How does this affect how a `shot` is moved?
- Must become a `'NO-SHOT shot`
- Suggests that there are three conditions that must be tested:

```
                      (eq?  a-shot NO-SHOT)  →  remains unchanged
          (= (posn-y a-shot) MIN-IMG-Y)  →  changes to 'NO-SHOT
    (not (= (posn-y a-shot) MIN-IMG-Y))  →  image-y is decreased
```

Aliens Attack Version 3

The move-shot Design

- ```
  ;; Sample expressions for move-shot
  (define MSHOT1-VAL INIT-SHOT)
  (define MSHOT2-VAL (make-posn (posn-x SHOT2)
                                (move-up-image-y (posn-y SHOT2))))
  (define MSHOT4-VAL NO-SHOT)
  ```
  ```

Aliens Attack Version 3

The move-shot Design

- ```
  ;; shot → shot
  ;; Purpose: To move the given shot
  (define (move-shot a-shot)
  ```

- ```
  ;; Sample expressions for move-shot
  (define MSHOT1-VAL INIT-SHOT)
  (define MSHOT2-VAL (make-posn (posn-x SHOT2)
                                (move-up-image-y (posn-y SHOT2))))
  (define MSHOT4-VAL NO-SHOT)
  ```

Aliens Attack Version 3

The move-shot Design

- ```
  ;; shot → shot
  ;; Purpose: To move the given shot
  (define (move-shot a-shot)
  ```

- ```
  ;; Sample expressions for move-shot
  (define MSHOT1-VAL INIT-SHOT)
  (define MSHOT2-VAL (make-posn (posn-x SHOT2)
                                (move-up-image-y (posn-y SHOT2))))
  (define MSHOT4-VAL NO-SHOT)
  ```

- ```
  ;; Tests using sample computations for move-shot
  (check-expect (move-shot INIT-SHOT) MSHOT1-VAL)
  (check-expect (move-shot SHOT2) MSHOT2-VAL)
  (check-expect (move-shot SHOT4) MSHOT4-VAL)

  ;; Tests using sample values for move-shot
  (check-expect (move-shot (make-posn 5 5)) (make-posn 5 4))
  (check-expect (move-shot (make-posn 5 MIN-IMG-Y)) NO-SHOT)
  ```
  ```

Aliens Attack Version 3

The move-shot Design

- ```
  ;; shot → shot
  ;; Purpose: To move the given shot
  (define (move-shot a-shot)
  ```
- ```
    (cond [(eq? a-shot NO-SHOT) a-shot]
          [(= (posn-y a-shot) MIN-IMG-Y) NO-SHOT]
          [else (make-posn (posn-x a-shot)
                           (move-up-image-y (posn-y a-shot)))]))
  ```
- ```
  ;; Sample expressions for move-shot
  (define MSHOT1-VAL INIT-SHOT)
  (define MSHOT2-VAL (make-posn (posn-x SHOT2)
                                (move-up-image-y (posn-y SHOT2))))
  (define MSHOT4-VAL NO-SHOT)
  ```
- ```
  ;; Tests using sample computations for move-shot
  (check-expect (move-shot INIT-SHOT) MSHOT1-VAL)
  (check-expect (move-shot SHOT2) MSHOT2-VAL)
  (check-expect (move-shot SHOT4) MSHOT4-VAL)

  ;; Tests using sample values for move-shot
  (check-expect (move-shot (make-posn 5 5)) (make-posn 5 4))
  (check-expect (move-shot (make-posn 5 MIN-IMG-Y)) NO-SHOT)
  ```

Aliens Attack Version 3

The move-shot Design

- Solve moving an image-y value up

- ```
  ;; Sample expressions for move-up-image-y
  (define IMGY>MIN-VAL1 (sub1 IMG-Y>MIN1))
  (define IMGY>MIN-VAL2 (sub1 IMG-Y>MIN2))
  ```
  ```

Aliens Attack Version 3

The move-shot Design

- Solve moving an image-y value up
- Moving up simply means decreasing the value of the given image-y
- ```
  ;; image-y>min → image-y
  ;; Purpose: To move the given image-y>min up
  (define (move-up-image-y an-img-y>min)
  ```
- ```
  ;; Sample expressions for move-up-image-y
  (define IMGY>MIN-VAL1 (sub1 IMG-Y>MIN1))
  (define IMGY>MIN-VAL2 (sub1 IMG-Y>MIN2))
  ```

Aliens Attack Version 3

The move-shot Design

- Solve moving an image-y value up
- Moving up simply means decreasing the value of the given image-y
- ```
  ;; image-y>min → image-y
  ;; Purpose: To move the given image-y>min up
  (define (move-up-image-y an-img-y>min)
  ```
- ```
  ;; Sample expressions for move-up-image-y
  (define IMGY>MIN-VAL1 (sub1 IMG-Y>MIN1))
  (define IMGY>MIN-VAL2 (sub1 IMG-Y>MIN2))
  ```
- ```
  ;; Tests using sample computations for move-up-image-y
  (check-expect (move-up-image-y AN-IMG-Y)  IMGY>MIN-VAL1)
  (check-expect (move-up-image-y MAX-IMG-Y) IMGY>MIN-VAL2)

  ;; Tests using sample values for f-on-image-y>min
  (check-expect (move-up-image-y 6)  5)
  (check-expect (move-up-image-y 11) 10)
  ```
  ```

Aliens Attack Version 3

The move-shot Design

- Solve moving an image-y value up
- Moving up simply means decreasing the value of the given image-y
- ```
  ;; image-y>min → image-y
  ;; Purpose: To move the given image-y>min up
  (define (move-up-image-y an-img-y>min)
  ```
- ```
    (sub1 an-img-y>min))
  ```
- ```
  ;; Sample expressions for move-up-image-y
  (define IMGY>MIN-VAL1 (sub1 IMG-Y>MIN1))
  (define IMGY>MIN-VAL2 (sub1 IMG-Y>MIN2))
  ```
- ```
  ;; Tests using sample computations for move-up-image-y
  (check-expect (move-up-image-y AN-IMG-Y)  IMGY>MIN-VAL1)
  (check-expect (move-up-image-y MAX-IMG-Y) IMGY>MIN-VAL2)

  ;; Tests using sample values for f-on-image-y>min
  (check-expect (move-up-image-y 6)  5)
  (check-expect (move-up-image-y 11) 10)
  ```

Aliens Attack Version 3

The `game-over?` Refinement

- Clear that the game may end in two ways
- When the alien reaches earth the player loses
- When the alien is hit by a shot the player wins

Aliens Attack Version 3

The `game-over?` Refinement

- Clear that the game may end in two ways
- When the alien reaches earth the player loses
- When the alien is hit by a shot the player wins
- At least three sample expressions are needed: game is not over, player loses, and player wins

Aliens Attack Version 3

The game-over? Refinement

- ```
  ;; Sample expressions for game-over?
  (define GAME-OVER1 (or (alien-reached-earth?
                            (world-alien INIT-WORLD2))
                         (hit? (world-shot INIT-WORLD2)
                               (world-alien INIT-WORLD2))))

  (define GAME-OVER2 (or (alien-reached-earth?
                            (world-alien WORLD3))
                         (hit? (world-shot WORLD3)
                               (world-alien WORLD3))))

  (define GAME-NOT-OVER (or (alien-reached-earth?
                               (world-alien INIT-WORLD))
                            (hit? (world-shot INIT-WORLD)
                                  (world-alien INIT-WORLD))
  ```
  ```

Aliens Attack Version 3

The game-over? Refinement

- ```
  ;; world → Boolean
  ;; Purpose: Detect if the game is over
  (define (game-over? a-world)
     (or (alien-reached-earth? (world-alien a-world))
         (hit? (world-shot a-world) (world-alien a-world))))
  ```
  ```

Aliens Attack Version 3

The game-over? Refinement

- ```
  ;; Tests using sample computations for game-over?
  (check-expect (game-over? INIT-WORLD2) GAME-OVER1)
  (check-expect (game-over? WORLD3)      GAME-OVER2)
  (check-expect (game-over? INIT-WORLD)  GAME-NOT-OVER)

  ;; Tests using sample values for game-over?
  (check-expect (game-over? (make-world 8
                                        (make-posn 0 3)
                                        'right
                                        NO-SHOT))
                #false)
  (check-expect (game-over? (make-world
                              8
                              (make-posn 0 MAX-IMG-Y)
                              'right
                              (make-posn 12 11)))
                #true)
  (check-expect (game-over? (make-world 8
                                        (make-posn 0 5)
                                        'right
                                        (make-posn 0 5)))
                #true)
  ```
  ```

Aliens Attack Version 3

The hit? Design

- ```
  (define ALIEN3 (make-posn 4 MAX-IMG-Y))
  ;; Sample expressions for hit?
  (define NHIT-VAL (and (posn? INIT-SHOT)
                        (= (posn-x INIT-SHOT) (posn-x INIT-ALIEN))
                        (= (posn-y INIT-SHOT) (posn-y INIT-ALIEN))))
  (define HIT-VAL  (and (posn? SHOT3)
                        (= (posn-x SHOT3) (posn-x ALIEN3))
                        (= (posn-y SHOT3) (posn-y ALIEN3))))
  ```
  ```

Aliens Attack Version 3

The hit? Design

- ```
  ;; shot → Boolean Purpose: Determine if shot hit alien
  (define (hit? a-shot an-alien)
  ```
- ```
  (define ALIEN3 (make-posn 4 MAX-IMG-Y))
  ;; Sample expressions for hit?
  (define NHIT-VAL (and (posn? INIT-SHOT)
                        (= (posn-x INIT-SHOT) (posn-x INIT-ALIEN))
                        (= (posn-y INIT-SHOT) (posn-y INIT-ALIEN))))
  (define HIT-VAL  (and (posn? SHOT3)
                        (= (posn-x SHOT3) (posn-x ALIEN3))
                        (= (posn-y SHOT3) (posn-y ALIEN3))))
  ```

Aliens Attack Version 3

The hit? Design

- ```
  ;; shot → Boolean Purpose: Determine if shot hit alien
  (define (hit? a-shot an-alien)
  ```

- ```
  (define ALIEN3 (make-posn 4 MAX-IMG-Y))
  ;; Sample expressions for hit?
  (define NHIT-VAL (and (posn? INIT-SHOT)
                        (= (posn-x INIT-SHOT) (posn-x INIT-ALIEN))
                        (= (posn-y INIT-SHOT) (posn-y INIT-ALIEN))))
  (define HIT-VAL  (and (posn? SHOT3)
                        (= (posn-x SHOT3) (posn-x ALIEN3))
                        (= (posn-y SHOT3) (posn-y ALIEN3))))
  ```

- ```
  ;; Tests using sample computations for hit?
  (check-expect (hit? INIT-SHOT INIT-ALIEN) NHIT-VAL)
  (check-expect (hit? SHOT3 ALIEN3) HIT-VAL)
  ;; Tests using sample values for hit?
  (check-expect (hit? (make-posn 0 0)  (make-posn 0 0)) #true)
  (check-expect (hit? (make-posn 15 6) (make-posn 1 2)) #false)
  ```
  ```

Aliens Attack Version 3

The hit? Design

- ```
  ;; shot → Boolean Purpose: Determine if shot hit alien
  (define (hit? a-shot an-alien)
  ```
- ```
    (and (posn? a-shot)
         (= (posn-x a-shot) (posn-x an-alien))
         (= (posn-y a-shot) (posn-y an-alien))))
  ```
- ```
  (define ALIEN3 (make-posn 4 MAX-IMG-Y))
  ;; Sample expressions for hit?
  (define NHIT-VAL (and (posn? INIT-SHOT)
                        (= (posn-x INIT-SHOT) (posn-x INIT-ALIEN))
                        (= (posn-y INIT-SHOT) (posn-y INIT-ALIEN))))
  (define HIT-VAL  (and (posn? SHOT3)
                        (= (posn-x SHOT3) (posn-x ALIEN3))
                        (= (posn-y SHOT3) (posn-y ALIEN3))))
  ```
- ```
  ;; Tests using sample computations for hit?
  (check-expect (hit? INIT-SHOT INIT-ALIEN) NHIT-VAL)
  (check-expect (hit? SHOT3 ALIEN3) HIT-VAL)
  ;; Tests using sample values for hit?
  (check-expect (hit? (make-posn 0 0)  (make-posn 0 0)) #true)
  (check-expect (hit? (make-posn 15 6) (make-posn 1 2)) #false)
  ```

Aliens Attack Version 3

The move-shot Design

- A winning world

```
(define FWORLD3 (make-world  7 (make-posn 19 3) 'left (make-posn 19 3)))
```

Aliens Attack Version 3

The move-shot Design

- A winning world

```
(define FWORLD3 (make-world  7 (make-posn 19 3) 'left (make-posn 19 3)))
```

- The sample expression using FWORLD3:

```
(define FWORLD3-VAL (place-image
                      (text "EARTH WAS SAVED!" 36 'green)
                      (/ E-SCENE-W 2)
                      (/ E-SCENE-H 4)
                      (draw-world FWORLD3)))
```

Aliens Attack Version 3

The move-shot Design

- A winning world

```
(define FWORLD3 (make-world  7 (make-posn 19 3) 'left (make-posn 19 3)))
```

- ```
  ;; world → scene throws error
  ;; Purpose: To draw the game's final scene
  (define (draw-last-world a-world)
  ```

- The sample expression using FWORLD3:

```
(define FWORLD3-VAL (place-image
                      (text "EARTH WAS SAVED!" 36 'green)
                      (/ E-SCENE-W 2)
                      (/ E-SCENE-H 4)
                      (draw-world FWORLD3)))
```
```

Aliens Attack Version 3

The move-shot Design

- A winning world

```
(define FWORLD3 (make-world  7 (make-posn 19 3) 'left (make-posn 19 3)))
```

- ```
  ;; world → scene throws error
  ;; Purpose: To draw the game's final scene
  (define (draw-last-world a-world)
  ```

- The sample expression using FWORLD3:

```
(define FWORLD3-VAL (place-image
                      (text "EARTH WAS SAVED!" 36 'green)
                      (/ E-SCENE-W 2)
                      (/ E-SCENE-H 4)
                      (draw-world FWORLD3)))
```

- New test

```
(check-expect (draw-last-world FWORLD3) FWORLD3-VAL)
```
```

Aliens Attack Version 3

The move-shot Design

- A winning world

```
(define FWORLD3 (make-world  7 (make-posn 19 3) 'left (make-posn 19 3)))
```

- ```
  ;; world → scene throws error
  ;; Purpose: To draw the game's final scene
  (define (draw-last-world a-world)
  ```

- ```
   (cond [(= (posn-y (world-alien a-world)) MAX-IMG-Y)
          (place-image (text "EARTH WAS CONQUERED!" 36 'red)
                       (/ E-SCENE-W 2)
                       (/ E-SCENE-H 4)
                       (draw-world a-world))]
         [(hit? (world-shot a-world) (world-alien a-world))
          (place-image (text "EARTH WAS SAVED!" 36 'green)
                       (/ E-SCENE-W 2)
                       (/ E-SCENE-H 4)
                       (draw-world a-world))]
         [else (error
                (format
                  "draw-last-world: Invalid world with ~s as the
                   alien value and ~s as the shot value."
                  (world-alien a-world)
                  (world-shot a-world)))]))
  ```

- The sample expression using FWORLD3:

```
(define FWORLD3-VAL (place-image
                      (text "EARTH WAS SAVED!" 36 'green)
                      (/ E-SCENE-W 2)
                      (/ E-SCENE-H 4)
                      (draw-world FWORLD3)))
```

- New test

```
(check-expect (draw-last-world FWORLD3) FWORLD3-VAL)
```

Aliens Attack Version 3

Double Bonus Quiz

- Problem: 112 (due in 1 week)