Introduction and Learning Objectives

• In this lesson, you will learn how to build more complicated worlds with more than one object.

• By the end of this lesson you should be able to
  – Write more complex data definitions, representing information in appropriate places.
  – Use structural decomposition to guide the development of programs incorporating multiple data definitions.
Requirements

- Like draggable-cat, except:
- We have 2 cats in the scene
- Each cat can be individually selected, as in draggable-cat
- Space pauses or unpauses the entire animation
- Demo: two-draggable-cats: http://www.youtube.com/watch?v=XvODwv7ivrA
two-draggable-cats: demo

Note: I've added a bunch of tests since this video was made. Study them!
Information Analysis

• The world has two cats and a paused?
  – it is the whole world that is paused or not
Data Definitions: World

(define-struct world (cat1 cat2 paused?)) ;; A World is a (make-world Cat Cat Boolean) ;; cat1 and cat2 are the two cats ;; paused? describes whether or not the world ;; is paused

;; template:
;; world-fn : World -> ??
;; (define (world-fn w)
;;   (... (world-cat1 w)
;;       (world-cat2 w)
;;       (world-paused? w)))
Information Analysis

• Each cat has x-pos, y-pos, and selected?
• What about paused?
  – cats aren't individually paused
  – it's the whole thing that is paused or not.
Data Definitions: Cat

(define-struct cat (x-pos y-pos selected?))
;; A Cat is a
;;   (make-cat Integer Integer Boolean)
;; Interpretation:
;; x-pos, y-pos give the position of the cat.
;; selected? describes whether or not the cat is
;; selected.

;; template:
;; cat-fn : Cat -> ??
;(define (cat-fn c)
 ; (... (cat-x-pos w)
 ;       (cat-y-pos w)
 ;       (cat-selected? w)))
Data Design Principles

• Every value of the information should be represented by some value of the data
  – otherwise, we lose immediately!

• Every value of the data should represent some value of the information
  – no meaningless or nonsensical combinations
  – if each cat had a paused? field, then what does it mean for one cat to be paused and the other not?
Follow the template!

• If your world has some cats in it, then your world function will just call a cat function on each cat.
• The structure of your program will follow the structure of your data definitions.
• Let's watch this at work:
world-after-tick

;;;; world-after-tick : World -> World
;;;; produces the world that should follow the
;;;; given world after a tick
;;;; strategy: structural decomposition on
;;;;   w : World
(define (world-after-tick w)
  (if (world-paused? w)
      w
      (make-world
        (cat-after-tick (world-cat1 w))
        (cat-after-tick (world-cat2 w))
        false)))

(world-cat1 w) is a cat, so we just call a cat function on it
cat-after-tick

;; cat-after-tick : Cat -> Cat
;; produces the state of the given cat after a tick in an
;; unpaused world.

;; examples:
;; cat selected
;; (cat-after-tick selected-cat-at-20) = selected-cat-at-20
;; cat paused:
;; (cat-after-tick unselected-cat-at-20) = unselected-cat-at-28

;; strategy: structural decomposition on c : Cat

(define (cat-after-tick c)
  (cat-after-tick-helper
   (cat-x-pos c) (cat-y-pos c) (cat-selected? c)))
cat-after-tick-helper

;; cat-after-tick-helper
;; : Integer Integer Boolean -> Cat
;; RETURNS: the cat that should follow one in the given
;; position in an unpaused world
;; strategy: function composition
(define (cat-after-tick-helper x-pos y-pos selected?)
  (if selected?
      (make-cat x-pos y-pos selected?)
      (make-cat
       x-pos
       (+ y-pos CATSPEED)
       selected?)))
world-to-scene

• world-to-scene follows the same pattern: the world consists of two cats, so we call two cat functions.

• Both cats have to appear in the same scene, so we will have to be a little clever about our cat function.
world-to-scene

;; world-to-scene : World -> Scene
;; produces a Scene that portrays the given world.
;; strategy: structural decomposition
;; on w : World
(define (world-to-scene w)
  (place-cat
    (place-cat
      (world-cat1 w)
      (place-cat
        (world-cat2 w)
        EMPTY-CANVAS)))))

The pieces are cats, so create a wishlist function to place a cat on a scene
place-cat

;; place-cat : Cat Scene -> Scene
;; returns a scene like the given one, but with
;; the given cat painted on it.
;; strategy : structural decomposition
;; on c : Cat
(define (place-cat c s)
  (place-image
   (place-image
    CAT-IMAGE
    (cat-x-pos c) (cat-y-pos c) s)))
Summary

• In this lesson, you had the opportunity to
  – Build a more complex world
  – Write more complex data definitions, representing information in appropriate places.
  – Use structural decomposition to guide the development of programs incorporating multiple data definitions.
Next Steps

• Run two-draggable-cats.rkt and study the code (including the tests!)
• If you have questions about this lesson, ask them on the Discussion Board
• Do Problem Set 02