Two Draggable Cats

CS 5010 Program Design Paradigms
“Bootcamp”

Lesson 3.4

© Mitchell Wand, 2012-2014
This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

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 templates 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

YouTube link

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

Information Analysis: World

- The world has two cats and a paused?
  - it is the whole world that is paused or not

Data Definitions: World

```
;; REPRESENTATION:
;; A World is represented as a (make-world cat1 cat2 paused?)
;; INTERPRETATION:
;; cat1, cat2 : Cat      the two cats in the world
;; paused?    : Boolean  is the world paused?

;; IMPLEMENTATION:
(define-struct world (cat1 cat2 paused?))

;; CONSTRCTOR TEMPLATE:
;; (make-world Cat Cat Boolean)

;; OBSERVER TEMPLATE:
;; world-fn : World -> ??
(define (world-fn w)
  (... (world-cat1 w) (world-cat2 w) (world-paused? w)))
```

Information Analysis: Cat

- 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

```
;; REPRESENTATION:
;; A Cat is represented as (make-cat x-pos y-pos selected?)
;; INTERPRETATION:
;; x-pos, y-pos : Integer      the position of the center of the cat
;;                             in the scene
;; selected?                   describes whether or not the cat is selected.

;; IMPLEMENTATION
(define-struct cat (x-pos y-pos selected?))

;; CONSTRUCTOR TEMPLATE:
;; (make-cat Integer Integer Boolean)

;; OBSERVER TEMPLATE:
;; template:
;; cat-fn : Cat -> ??
(define (cat-fn w)
 (... (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 would it mean for one cat to be paused and the other not?
  - Is it possible for one cat to be paused and the other not?

Follow the template!

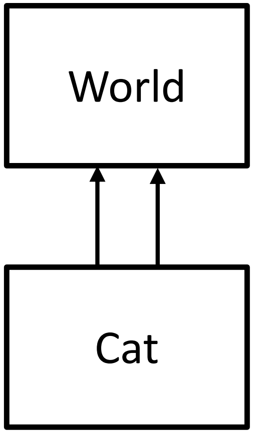
- If your world has some cats in it, then your world function will just call a cat function on each cat.
- The structure of your program will follow the structure of your data definitions.
- Let's watch this at work:

world-after-tick

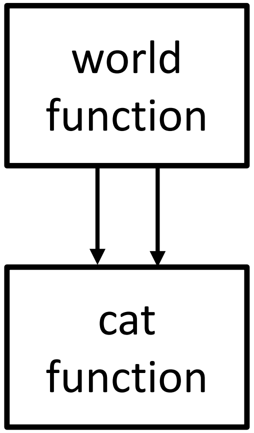
```
;; world-after-tick : World -> World
;; RETURNS: the world that should follow the
;; given world after a tick
;; STRATEGY: Cases on whether the world is paused

(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

Remember: The Shape of the Program Follows the Shape of the Data



Data Hierarchy (the world contains 2 cats)



Call Tree (the arrow goes from caller to callee)

cat-after-tick

```
;; cat-after-tick : Cat -> Cat
;; RETURNS: 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: Cases on whether the cat is selected, then use
;;           constructor template for cat.

;; function definition on next slide
```

It would be OK to write just "Use template on c"

cat-after-tick definition

```
(define (cat-after-tick c)
  (if (cat-selected? c)
    c
    (make-cat
      (cat-x-pos c)
      (+ (cat-y-pos c) CATSPEED)
      (cat-selected? c))))
```

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
;; RETURNS: a Scene that portrays the
;;   given world.
;; STRATEGY: Use template for World on w
(define (world-to-scene w)
  (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

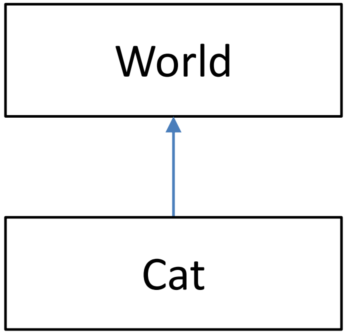
place-cat

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

The Structure of the Program Follows the Structure of the Data (1)

- Let's look again at the structure of our program.
- If we draw the call graph of our program (showing which functions call which), we can see that the call graph mirrors the structure of the data
- The world contains two cats, so world-after-tick calls cat-after-tick (twice).
- Let' draw some pictures:

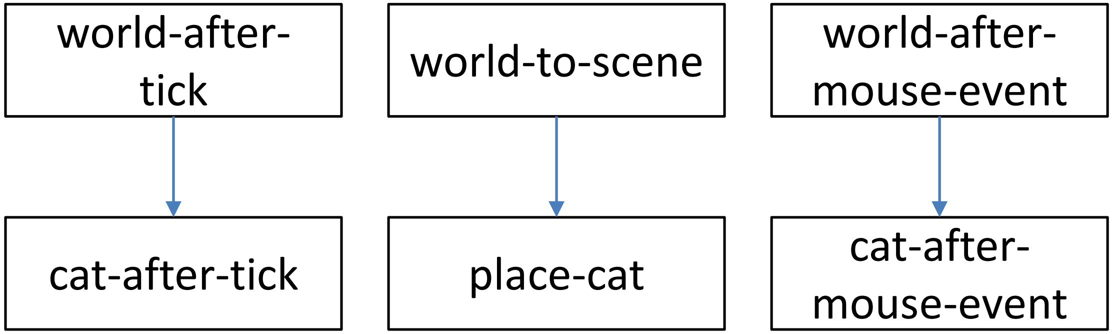
The Structure of the Program Follows the Structure of the Data (2)



World

Cat

Data Definitions



world-after-tick

world-to-scene

world-after-mouse-event

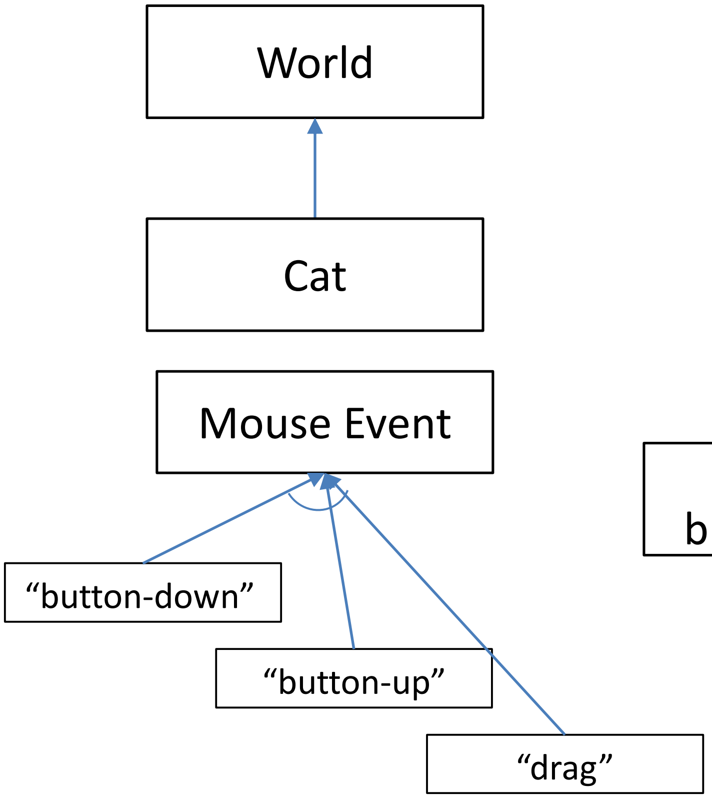
cat-after-tick

place-cat

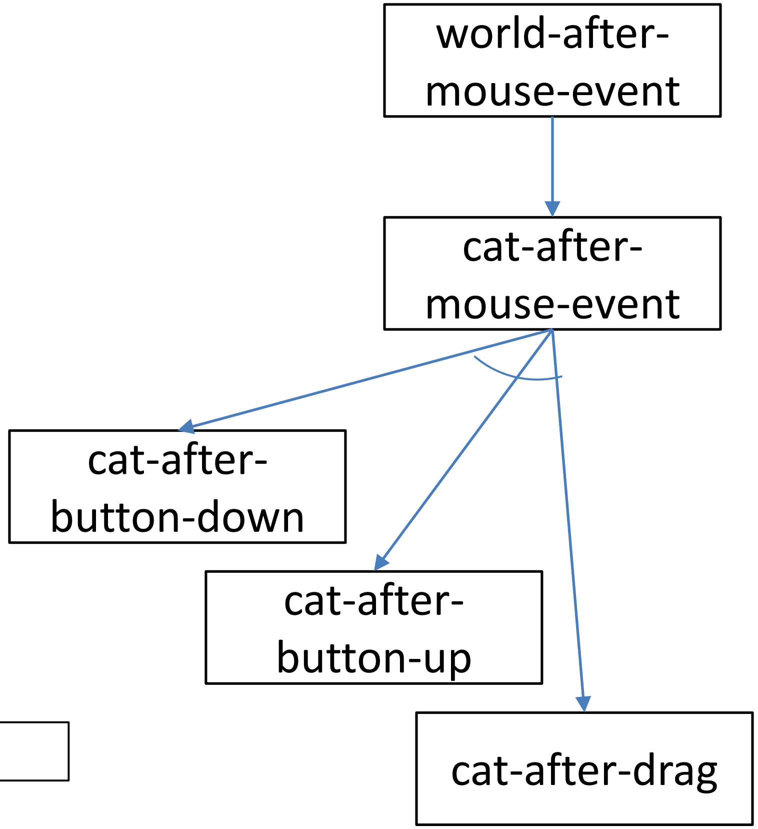
cat-after-mouse-event

Call Graphs

The Structure of the Program Follows the Structure of the Data (3)



Data Definitions

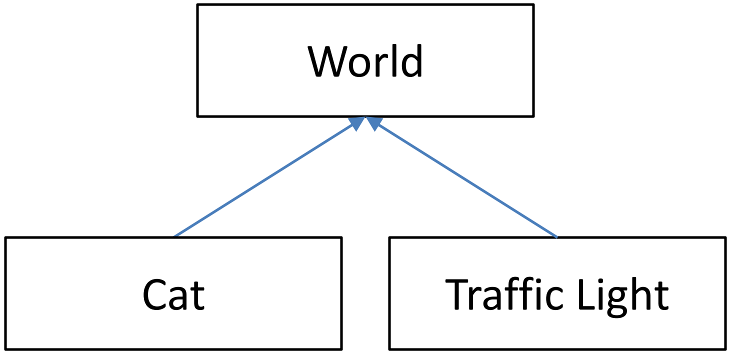


Call Graph

The world contains a cat (or cats), so **world-after-mouse-event** calls **cat-after-mouse-event** (once per cat).

A MouseEvent is either a button-down, a button-up, or a drag, so **cat-after-mouse-event** calls one of **cat-after-button-down**, **cat-after-button-up**, or **cat-after-drag**.

What if there were more things in the world?

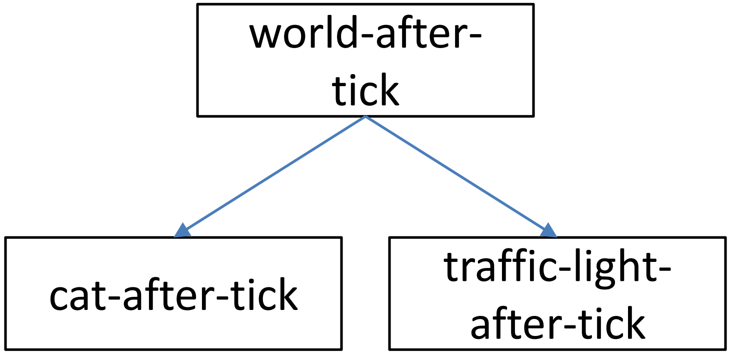


World

Cat

Traffic Light

Data Definitions



world-after-tick

cat-after-tick

traffic-light-after-tick

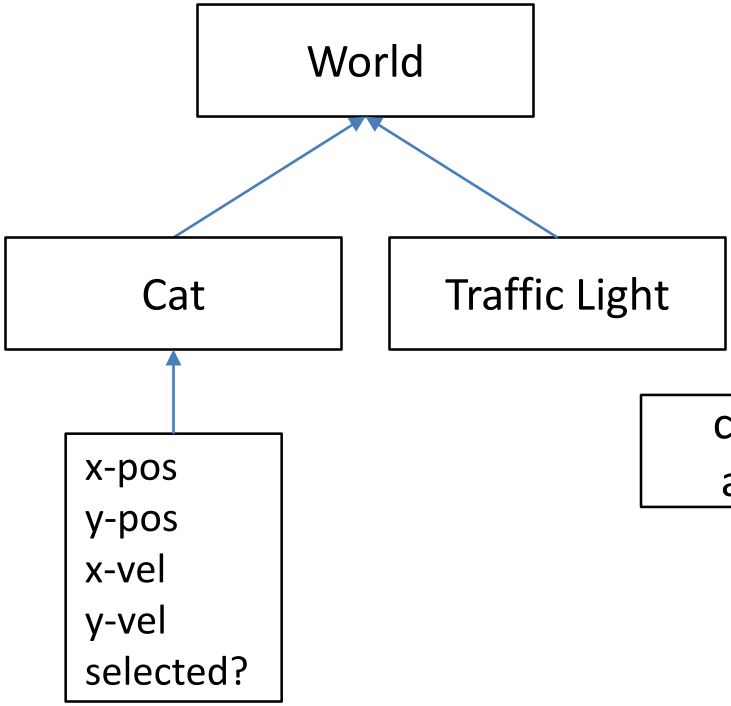
Call Graph

Maybe the world contains a cat and a traffic light...
Then **world-after-tick** would have to call both **cat-after-tick** and **traffic-light-after-tick**

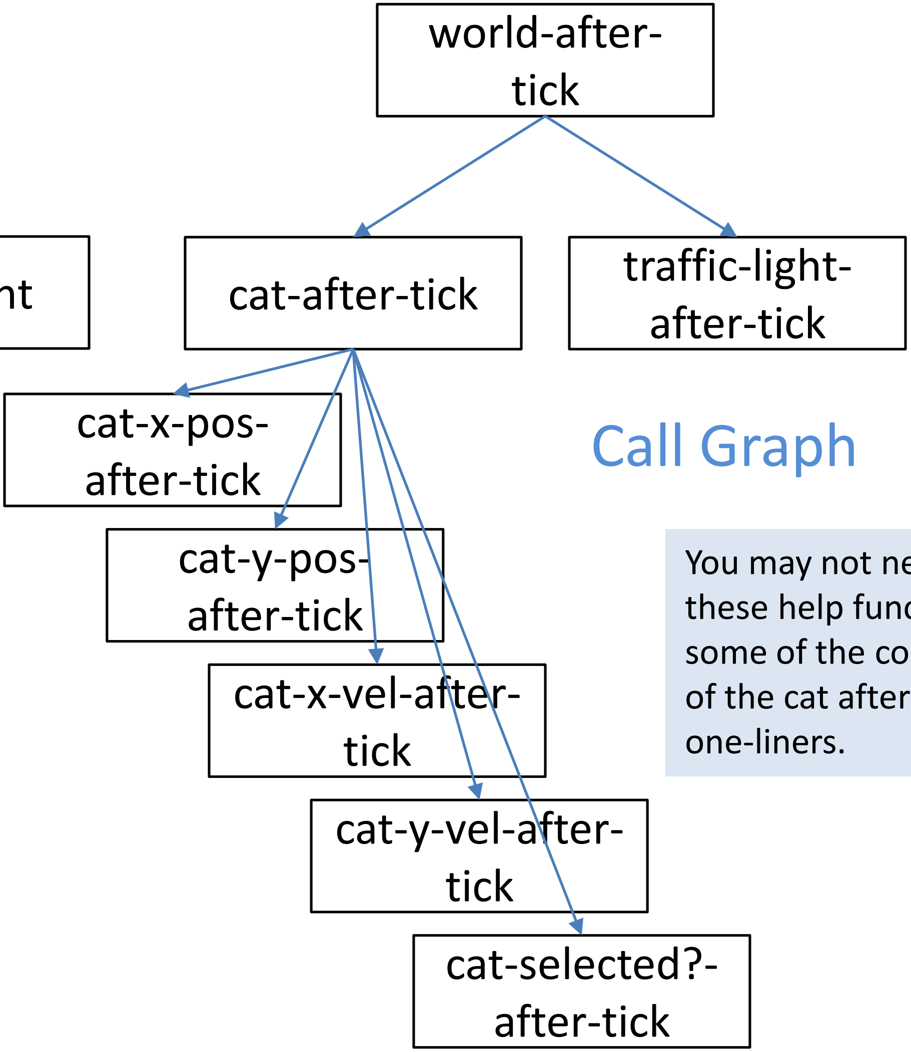
What if the motion of the cat were more complicated?

- In our problem, the components of the new cat were all "one-liners".
- If the motion of the cat were more complicated, you might need to do some complicated computation to determine the next x,y position and next x,y velocities of the cat.
- You'd turn some or all of these into help functions.
- This still winds up following the structure of the data:

What if the motion of the cat were more complicated? (2)



Data Definitions



Call Graph

You may not need all of these help functions if some of the components of the cat after the tick are one-liners.

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 the structure of the data to guide the development of programs incorporating multiple data definitions ("the structure of the program follows the structure of the data").

Next Steps

- Run **3-4-two-draggable-cats.rkt** and study the code (including the tests!)
- If you have questions about this lesson, ask them on the Discussion Board