The Iterative Design Recipe

CS 5010 Program Design Paradigms "Bootcamp"

Lesson 3.3

Introduction

In this lesson, you will learn a recipe (a "workflow") for systematically adding new features to a working program.

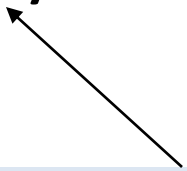
We will illustrate the recipe by modifying our falling-cat program so that it responds to mouse events.

Learning Objectives

- At the end of this lesson you should be able to:
  - list the steps in adding functionality to a working program
  - design a Universe program that responds to mouse events

The Iterative Design Recipe

- The Iterative Design Recipe is a recipe (a "workflow") to keep you organized as you add new features to a working program.
- We call this the “iterative design recipe” because it tells us how to build a system by iteratively adding more complex features.



“iteratively” means “repeatedly” or “in stages”

The Iterative Design Recipe

Adding a New Feature to an Existing Program
1. Perform information analysis for new feature
2. Modify data definitions as needed
3. Update existing functions to work with new data definitions
4. Write wishlist of functions for new feature
5. Design new functions following the Design Recipe
6. Repeat for the next new feature

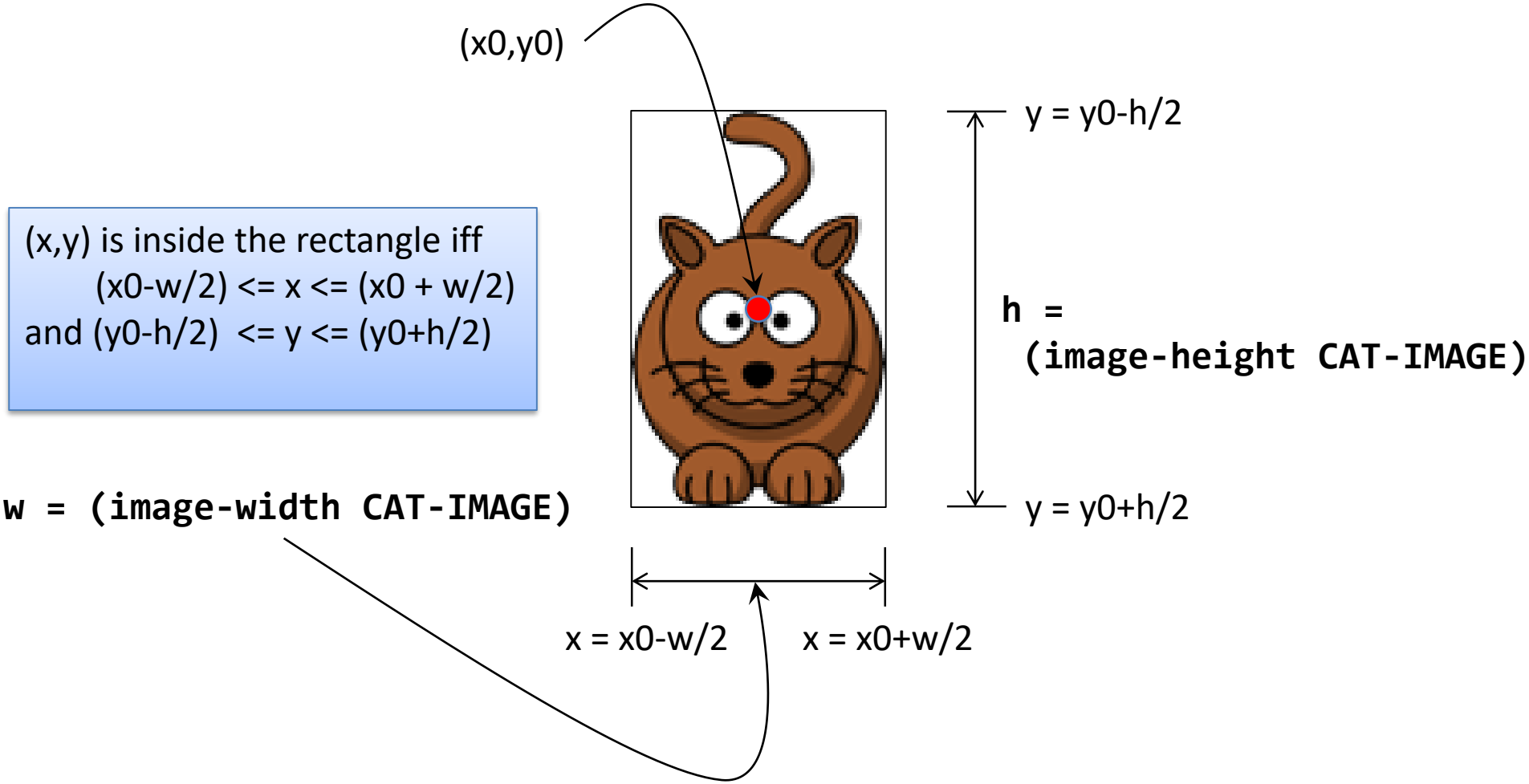
draggable-cat: Requirements

- Like falling cat, but user can drag the cat with the mouse.
- button-down to select, drag to move, button-up to release.
- A selected cat doesn't fall. When unselected, cat resumes its previous pausedness
  - if it was falling, it will continue to fall when released
  - if it was paused, it will remain paused when released

Video: draggable-cat demo

YouTube link

in-cat? relies on Bounding Box

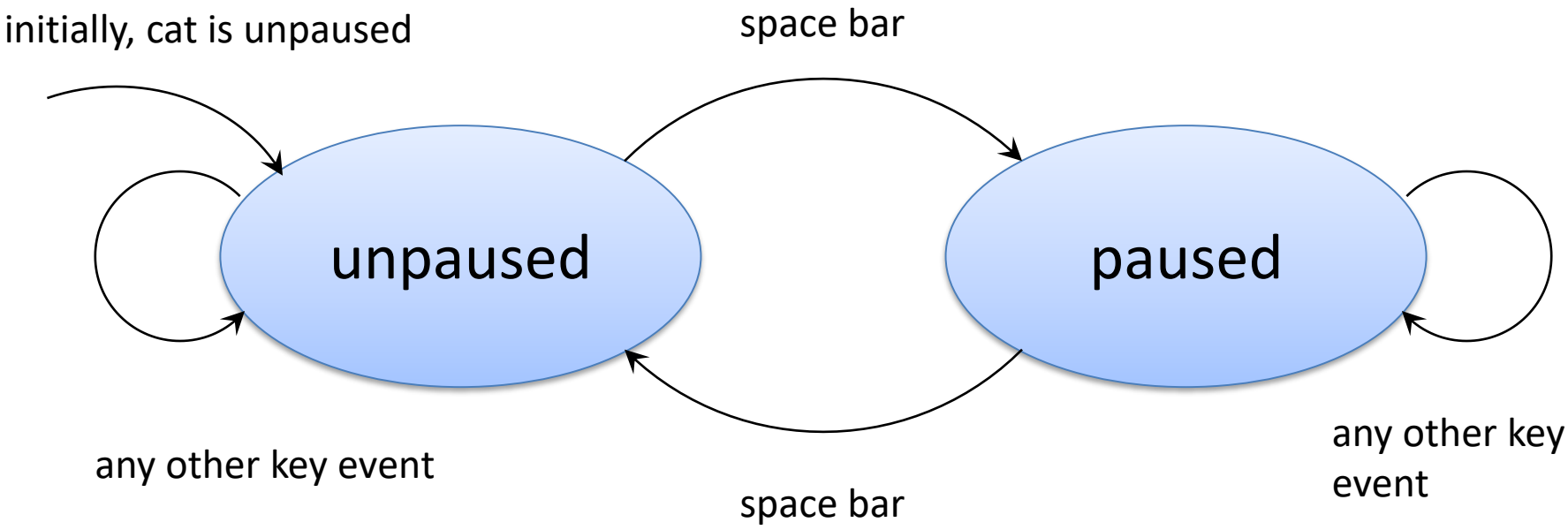


(x,y) is inside the rectangle iff
(x0-w/2) <= x <= (x0 + w/2)
and (y0-h/2) <= y <= (y0+h/2)

Information Analysis

- What are the possible behaviors of the cat?
  - as it falls?
  - as it is dragged?
- If we can answer these questions, we can determine what information needs to be represented for the cat.
- Let's write down the answers in graphical form.

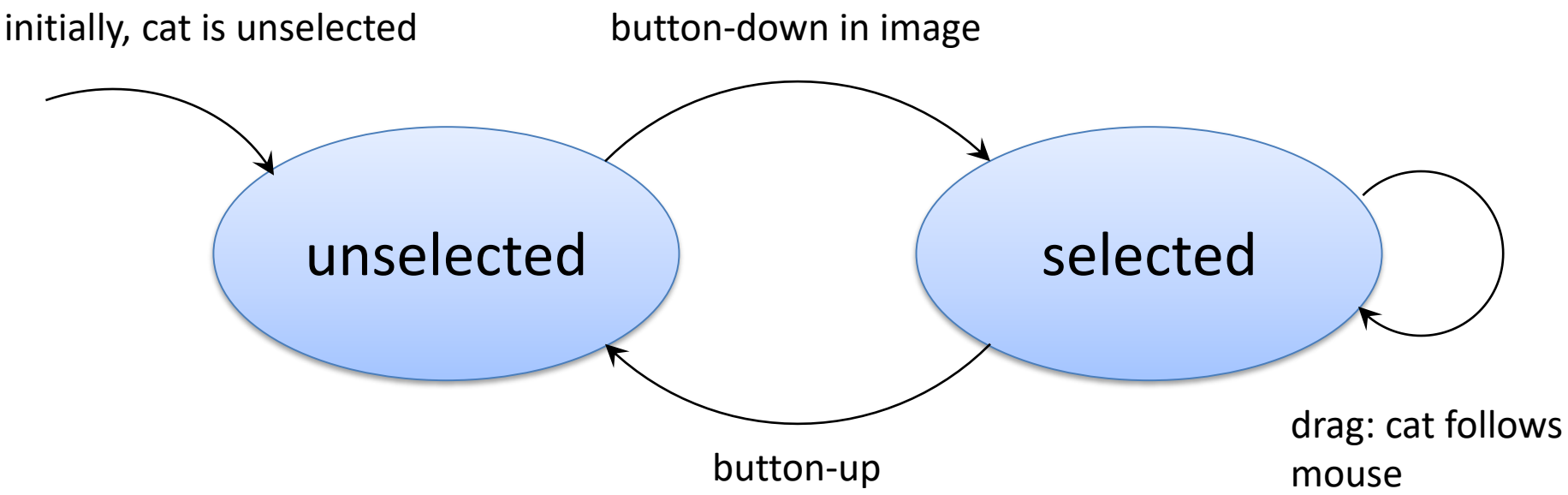
Life Cycle of a falling cat



As the cat falls, it is either paused or unpaused. When the user hits the space bar, an unpaused cat becomes paused, and a paused cat becomes unpaused. Any other key event is ignored.

This is, of course, the same analysis that we did for falling-cat, but it's helpful to see it in graphical form.

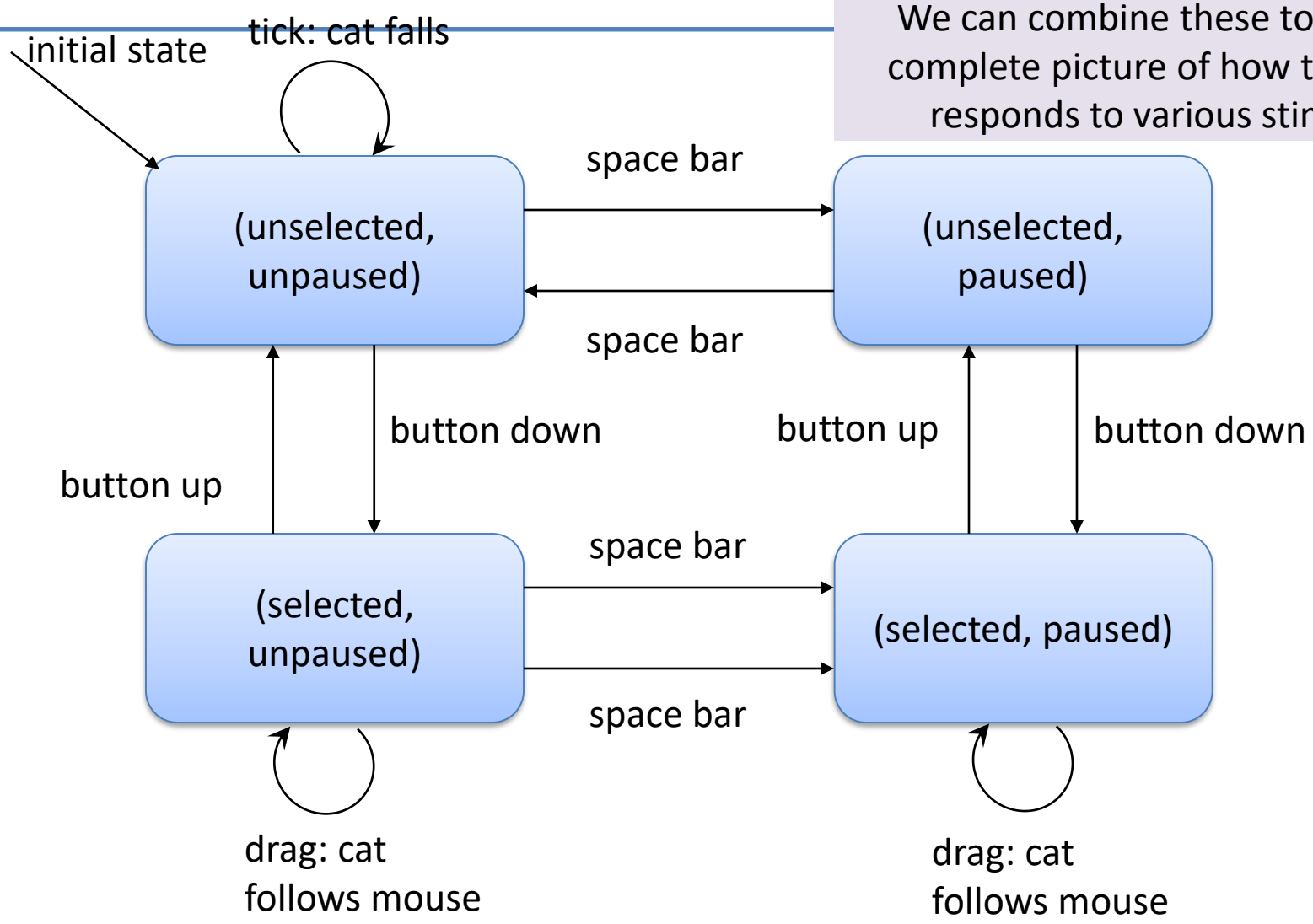
Life Cycle of a dragged cat



We can do a similar analysis for the cat as it is dragged. As the cat is dragged, it is either selected or unselected. Here is a state diagram that shows what things cause the cat to change from selected to unselected or vice versa.

Life cycle of a dragged, falling cat

We can combine these to get a complete picture of how the cat responds to various stimuli



Information Analysis: the Cat

- As before, our world consists of a single cat.
- Since the cat can be dragged in the **x** direction, we need to keep track of both the **x** position and **y** position of the cat.
- We also keep track of two Boolean values, telling us whether the cat is paused and whether the cat is selected.
- Here is the data definition, including the template.

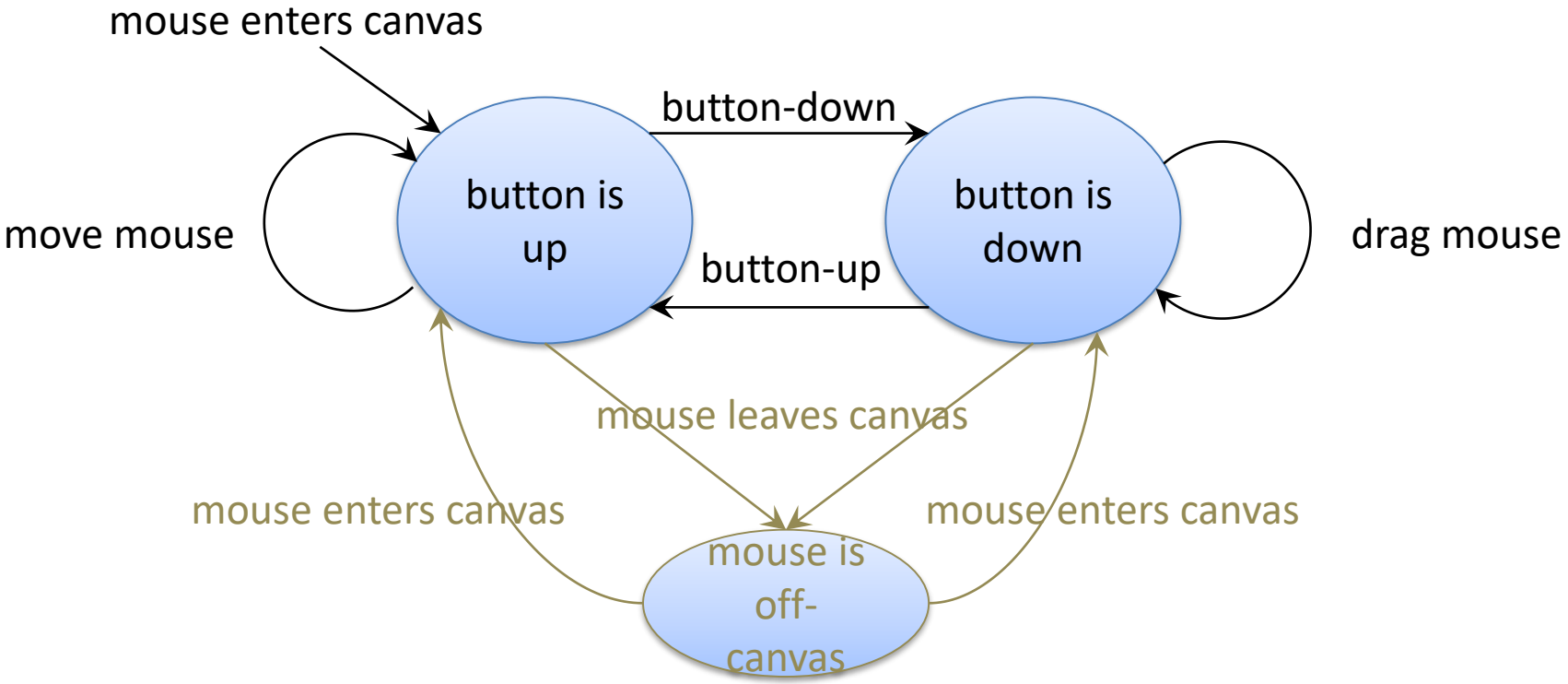
Data Design for Cat

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

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

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

;; OBSERVER TEMPLATE:
;; template:
;; world-fn : World -> ??
(define (world-fn w)
 (... (world-x-pos w)
      (world-y-pos w)
      (world-paused? w)
      (world-selected? w)))
```

Life Cycle of Mouse Movements

- What are the possible movements of a mouse?
- Initially, the mouse enters the canvas (an **"enter"** event) and the button is up.
- While the button is up, the user can only do 3 things:
  - move the mouse (a **"move"** event) or
  - move the mouse off the canvas (a **"leave"** event) or
  - depress the mouse button (a **"button-down"** event).
- While the button is down, again the user can do exactly 3 things:
  - move the mouse (this is called a **"drag"** event)
  - move the mouse off the canvas (a **"leave"** event) or
  - release the mouse button (a **"button-up"** event)
- When the mouse is off the canvas, no events are possible.
- Similarly, we can draw a state-transition diagram for the movements of the mouse.

Life Cycle of Mouse Movements



Mouse movements have their own life cycle. We've drawn the off-canvas events in a lighter color because most of the time we don't need to worry about them.

Information Analysis: Mouse Events

- Looking at the life cycle of a dragged cat, we see that only three mouse events are relevant:
  - **"button-down"** ,
  - **"drag"**, and
  - **"button-up"**.
- Other mouse events, like **"enter"**, **"leave"**, or **"move"** are ignored.
- We can write a template for doing cases on MouseEvents for this application:

Case analysis for mouse events

```
; mev-fn : MouseEvent -> ??
; STRATEGY: Cases on MouseEvent mev
;(define (mev-fn mev)
;  (cond
;    [(mouse=? mev "button-down") ...]
;    [(mouse=? mev "drag") ...]
;    [(mouse=? mev "button-up") ...]
;    [else ...]))
```

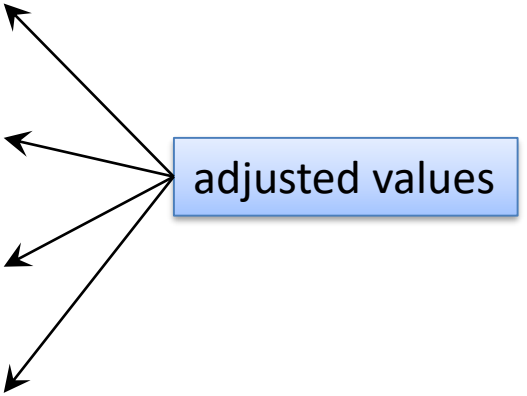
We won't require you to write down this template, but you may find that writing it down is helpful, since you are likely to use the same set of cases several times in your program.

Getting your old program to work with the new data definitions

- Don't try adding the new features yet!
- First, get all your old functions working with the new data definitions.
- Make sure your old tests work
  - Don't change your tests!
  - If you used mostly symbolic names for the test inputs and outputs, so you should be able to just change those definitions.
  - The tests themselves should work unchanged.

Testing your old functions

```
(define unpaused-world-at-20
  (make-world CAT-X-COORD 20 false false))
(define paused-world-at-20
  (make-world CAT-X-COORD 20 true false))
(define unpaused-world-at-28
  (make-world CAT-X-COORD 28 false false))
(define paused-world-at-28
  (make-world CAT-X-COORD 28 true false))
...
(check-equal?
  (world-after-key-event paused-world-at-20 pause-key-event)
  unpaused-world-at-20
  "after pause key, a paused world should become unpaused")
```



same tests

Everything OK?

- Good. Now we are ready to move on to the new features.

Responding to Mouse Events

```
(big-bang ...
  (on-mouse world-after-mouse-event))


world-after-mouse-event :
  World Integer Integer MouseEvent -> World
```

Look in the Help Desk for details about **on-mouse**

world-after-mouse-event

```
;; world-after-mouse-event :
;;    World Integer Integer FallingCatMouseEvent
;;    -> World
;; RETURNS: the world that should follow the given mouse event
;; examples:  See slide on life cycle of dragged cat
;; strategy: cases on mouse events
(define (world-after-mouse-event w mx my mev)
  (cond
    [(mouse=? mev "button-down")
     (world-after-button-down w mx my)]
    [(mouse=? mev "drag")
     (world-after-drag w mx my)]
    [(mouse=? mev "button-up")
     (world-after-button-up w mx my)]
    [else w]))
```

How to test this function?

- 3 mouse events (+ a test for the else clause)
- cat selected or unselected
  - mouse works the same way whether the cat is paused or not.
- event inside cat or not.
- 3 x 2 x 2 = 12 tests
- plus test for else clause
- plus: cat remains paused or unpaused across selection.
- Demo: draggable-cat.rkt

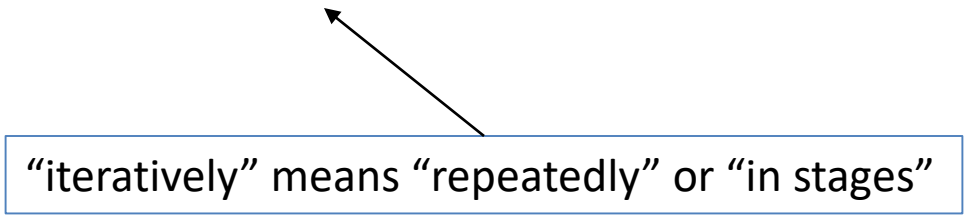
Draggable-cat readthrough

YouTube link

Remember, in the time since this video was recorded, we've changed many of the details. Look carefully at the file in the Examples folder. That's the one that your code should resemble.

Review: The Iterative Design Recipe

- We started with a simple system, and we added some new features to it.
- In doing this, we were following a recipe.
- We call this the “iterative design recipe” because it tells us how to build a system by iteratively adding more complex features.



The Iterative Design Recipe

Adding a New Feature to an Existing Program
1. Perform information analysis for new feature
2. Modify data definitions as needed
3. Update existing functions to work with new data definitions
4. Write wishlist of functions for new feature
5. Design new functions following the Design Recipe
6. Repeat for the next new feature

Summary

- In this lesson, you had the opportunity to
  - create a Universe program that responds to mouse events
  - use the Iterative Design Recipe for adding functionality to a working program

Next Steps

- Study 03-3-draggable-cat.rkt in the Examples folder.
- If you have questions about this lesson, ask them on the Discussion Board
- Go on to the next lesson