CS160: User Interface Design

Widgets, Events, MVC 02/22/10

Berkeley
UNIVERSITY OF CALIFORNIA

Due

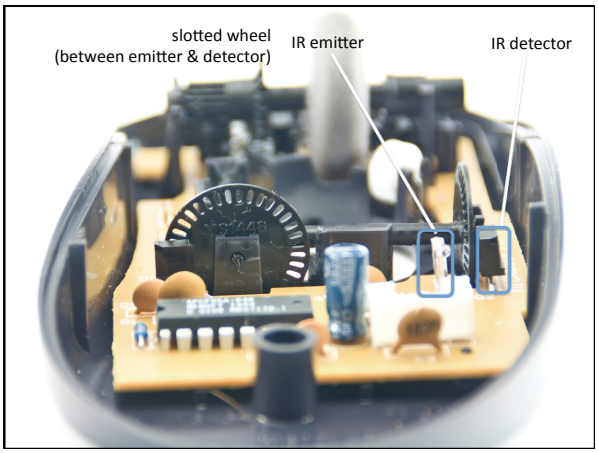
Today: Contextual Inquiry
Mar 3: Prog. Assignment 4

Contextual Inquiry

Coach Class

Group F

BART System Map



Review: Input Device Design Space

Table I. Physical Properties Used by Input Devices

	Linear	Rotary
Position		
Absolute	Position **P**	Rotation **R**
Relative	Movement **dP**	Delta rotation **dR**
Force		
Absolute	Force **F**	Torque **T**
Relative	Delta force **dF**	Delta torque **dT**

Card, S. K., Mackinlay, J. D., and Robertson, G. G. 1991. A morphological analysis of the design space of input devices. *ACM Trans. Inf. Syst.* 9, 2 (Apr. 1991), 99-122.

Review: Fitts' Law

Time T_{pos} to move the hand to target size S which is distance D away is given by:

$$T_{pos} = a + b \log_2 (D/S + 1)$$

Index of Difficulty (ID)

Only *relative precision* matters

Source: Landay, James. "Human Abilities". CS160 UC Berkeley.

Review: Fitts' Law

Time T_{pos} to move the hand to target size S which is distance D away is given by:

$$T_{pos} = a + b \log_2 (D/S + 1)$$

Device Characteristics
(bandwidth of human muscle group & of device)

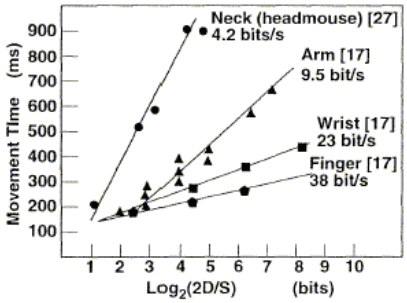
a: start/stop time
b: speed

Source: Landay, James. "Human Abilities". CS160 UC Berkeley.

Review: Which is faster?

Source: Card, Stu. Lecture on Human Information Interaction. Stanford, 2007.

Review: Bandwidth of Human Muscle Groups



Source: Card, Stu. Lecture on Human Information Interaction. Stanford, 2007.

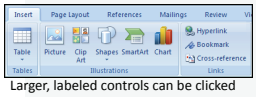
Review: Fitts' Law in Windows & Mac OS

Windows 95: Missed by a pixel
Windows XP: Good to the last drop

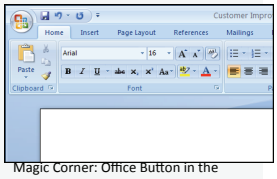
The Apple menu in
Mac OS X v10.4 Tiger.

Source: Jensen Harris, An Office User Interface Blog : Giving You Fitts. Microsoft, 2007; Apple

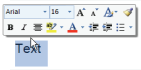
Review: Fitts' Law in Microsoft Office 2007



Larger, labeled controls can be clicked more quickly



Magic Corner: Office Button in the upper-left corner



Mini Toolbar: Close to the cursor

Source: Jensen Harris, An Office User Interface Blog : Giving You Fitts. Microsoft, 2007.

Topics for today

Interactive application programming

Component Model

Event-Driven User Interfaces

Model-View-Controller

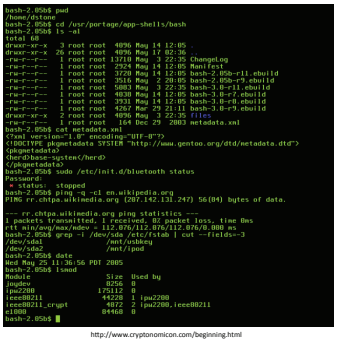
Architecture for interactive components

Why do we need it?

Changing the display

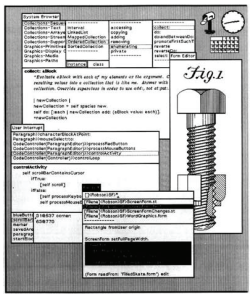
Interactive Application Programming

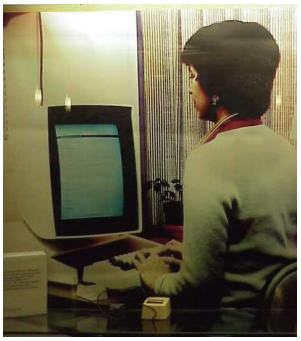
In the beginning…



http://www.cryptonomicon.com/beginning.html

The Xerox Alto (1973)





Event-Driven UIs

Old model (e.g., UNIX shell, DOS)

Interaction controlled by system, user queried for input when needed by system

Event-Driven Interfaces (e.g., GUIs)

Interaction controlled by user

System waits for user actions and then reacts

More complicated programming and architecture

Console program pseudo-code

Do some work…
Prompt user for input
Wait for user input
Process user input…
Do some more work…
Exit

Console program pseudo-code

```
//Objective-C:
int userInput;
NSLog( @"Enter a number: " );
scanf( "%i", &userInput );
NSLog( @"You typed %i.", userInput );
```

Console program

```
// Java Example:
Console console = System.console();
String name = console.readLine("Your name:");
System.out.println("You have entered : " + name);
```

Minimal "interactive" program

```
Do until a quit command: {
    wait for user input
    process it…
    (optionally) update display
}
```

Minimal "interactive" program

```
Do until a quit command: {
    wait for user input
    switch (input-cmd) {
        case insert: do-insert(…)
        case delete: do-delete(…)
        case backspace: …
    (optionally) update display
}
```

Minimal "interactive" program

Can't use this (global) approach for window systems, because the result of a user command **depends on the active window** (and the active component within that window).

Too many possible combinations of input x target window, and window structure is dynamic.

Widgets

Encapsulation and organization of interactive controls
Class hierarchy encapsulating widgets
Top-level "Component" class
Implements basic bounds management, and event processing

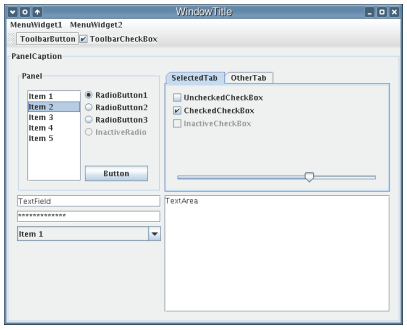
Drawn using underlying 2D graphics library

Input event processing and handling
Typically mouse and keyboard events

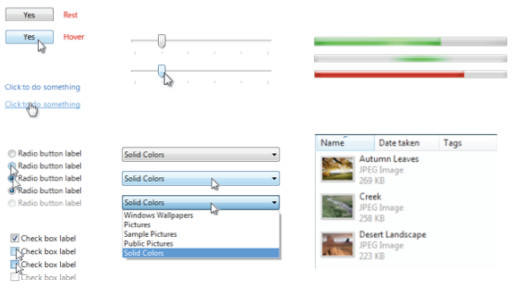
Bounds management (damage/redraw)
Only redraw areas in need of updating

Widgets

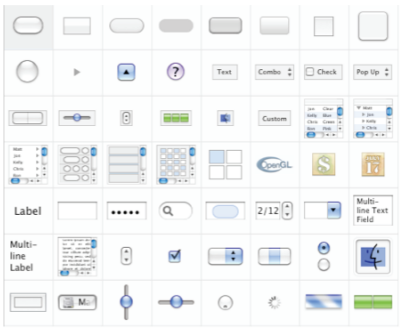
Java Swing Widgets



Windows Vista Widgets



Mac Cocoa Widgets



Interface Builder - Library

User Interface Components

Each component is an object with

Bounding box

Paint method for drawing itself

Drawn in the component's coordinate system

Callbacks to process input events

Mouse clicks, typed keys


```
Java:
public void paint(Graphics g) {
  g.fillRect(…); // interior
  g.drawString(…); // label
  g.drawRect(…); // outline
}
```

Cocoa:
(void)**drawRect:**(NSRect)*rect*

2D Graphics Model

Widget canvas and coordinate system

Origin often at top-left, increasing down and to the right

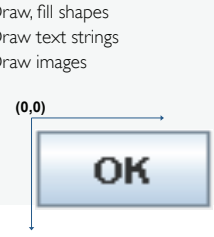
Units depend on output medium (e.g., pixels for screen)

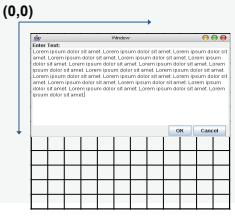
Rendering methods

Draw, fill shapes

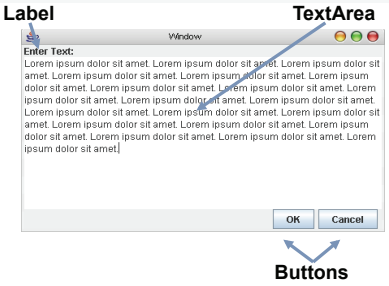
Draw text strings

Draw images



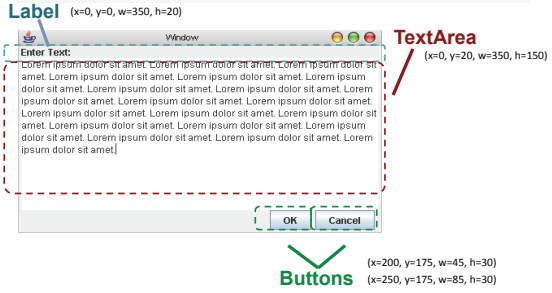


Composing a User Interface



How might we instruct the computer to generate this layout?

Absolute Layout



But this is inflexible and doesn't scale or resize well.

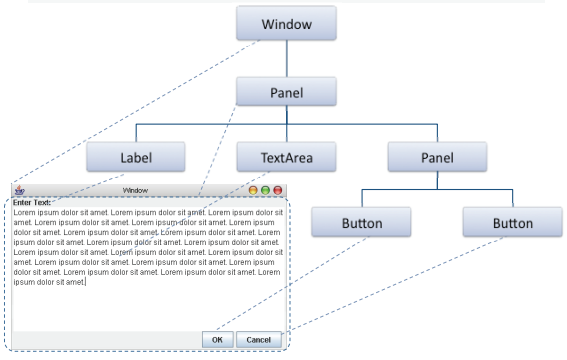
(Almost) No Layout

```
<h3>Enter Text:</h3>
<form method="post" action="">
    <textarea name="theText" cols="45" rows="5"></textarea>
    <br/>
    <input type="submit" name="btnOK"  value="Ok" />
    <input type="submit" name="btnCancel" id="button" value="Cancel" />
</form>
```

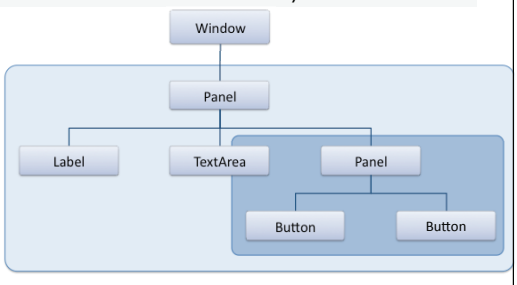
Enter Text:

Ok Cancel

Containment Hierarchy

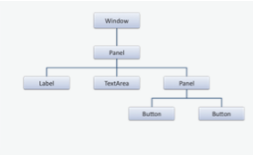


Containment Hierarchy



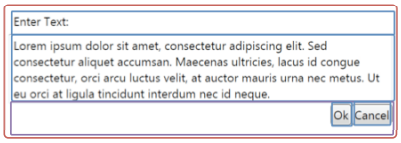
Principle: Each container is responsible for allocating space and positioning its contents.

Example Declarative Layout (WPF)



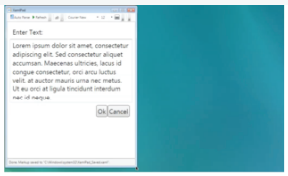

```
<StackPanel>
  <Label>Enter Text:</Label>
  <TextBox TextWrapping="Wrap">…</TextBox>
  <StackPanel Orientation="Horizontal"
              HorizontalAlignment="Right">
    <Button>Ok</Button>
    <Button>Cancel</Button>
  </StackPanel>
</StackPanel>
```

Example Declarative Layout (WPF)



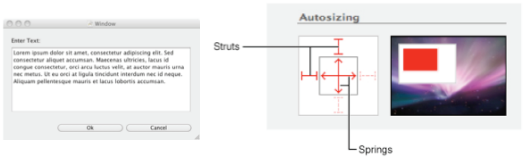

```
<StackPanel>
  <Label>Enter Text:</Label>
  <TextBox TextWrapping="Wrap">…</TextBox>
  <StackPanel Orientation="Horizontal"
              HorizontalAlignment="Right">
    <Button>Ok</Button>
    <Button>Cancel</Button>
  </StackPanel>
</StackPanel>
```

Example Declarative Layout (WPF)



```
<StackPanel>
  <Label>Enter Text:</Label>
  <TextBox TextWrapping="Wrap">…</TextBox>
  <StackPanel Orientation="Horizontal"
              HorizontalAlignment="Right">
    <Button>Ok</Button>
    <Button>Cancel</Button>
  </StackPanel>
</StackPanel>
```

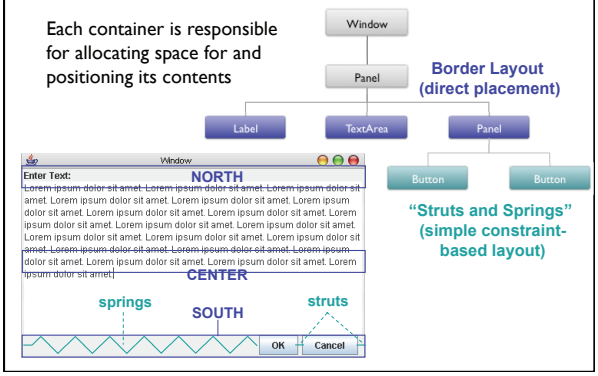
Layout in Cocoa: Springs + Struts



Interface Builder Demo

Component Layout

Each container is responsible for allocating space for and positioning its contents



Border Layout (direct placement)

"Struts and Springs" (simple constraint-based layout)

Specifying Layout

Declarative
e.g., HTML, XAML, MXML,…

Procedural
e.g., Java Swing

GUI Builders exist for either approach (but generating procedural code is brittle)

Is your UI layout determined statically or dynamically at runtime? If at runtime, may need procedural approach.

```
<StackPanel>
  <Label>Enter Text:</Label>
  <TextBox TextWrapping="Wrap">…</TextBox>
  <StackPanel Orientation="Horizontal"
              HorizontalAlignment="Right">
    <Button>Ok</Button>
    <Button>Cancel</Button>
  </StackPanel>
</StackPanel>
```

Specifying Layout

Declarative
e.g., HTML, XAML, MXML,…

Procedural
e.g., Java Swing

GUI Builders exist for either approach (but generating procedural code is brittle)

Is your UI layout determined statically or dynamically at runtime? If at runtime, may need procedural approach.

```
public void init() {
    Container c = getContentPane();
    c.setLayout(new BorderLayout());
    c.add(new JButton("One"),
         BorderLayout.NORTH);
    c.add(new JButton("Two"),
         BorderLayout.WEST);
    c.add(new JButton("Three"),
        BorderLayout.CENTER);
}
```

Events

Events

User input is modeled as "events" that must be handled by the system and applications.

Examples?

- Mouse input (and touch, pen, etc.)
  - Mouse entered, exited, moved, clicked, dragged
  - Inferred events: double-clicks, gestures
- Keyboard (key down, key up)
- Sensor inputs
- Window movement, resizing

Anatomy of an Event

Encapsulates info needed for handlers to react to input

Event Type (mouse moved, key down, etc)
Event Source (the input component)
Timestamp (when did event occur)
Modifiers (Ctrl, Shift, Alt, etc)
Event Content
Mouse: x,y coordinates, button pressed, # clicks
Keyboard: which key was pressed

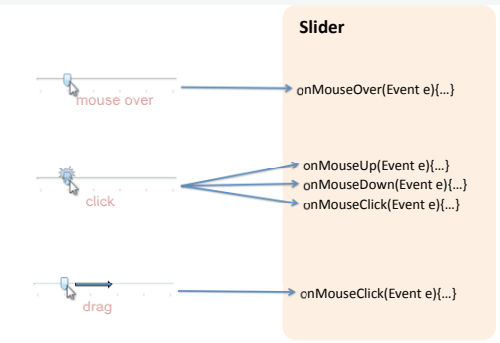
Abstracting Events

Level of abstraction may vary. Consider:

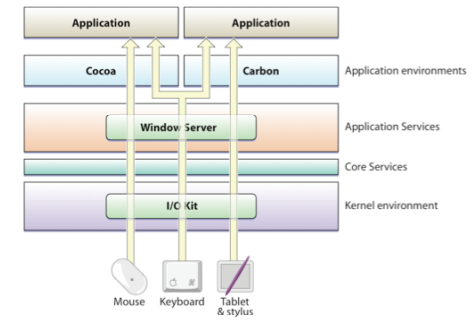
Mouse down vs. **double click** vs. **drag**

Pen move vs. **gesture**

Callbacks



Event Dispatch



Apple, Cocoa Event-Handling Guide

Event Dispatch Loop

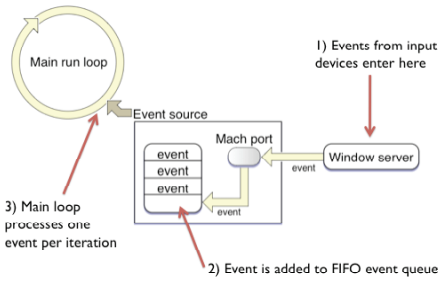
Mouse moved (t_0,x,y)

Event Queue
- Queue of input events

Event Loop (runs in dedicated thread)
- Remove next event from queue
- Determine event type
- Find proper component(s)
- Invoke callbacks on components
- Repeat, or wait until event arrives

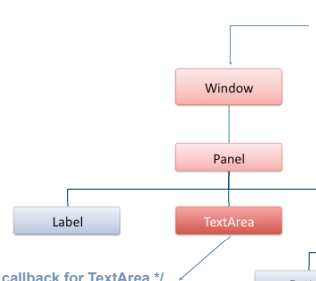
Component
- Invoked callback method
- Update application state
- Request repaint, if needed

Event Dispatch Loop



Apple, Cocoa Event-Handling Guide

Event Dispatch

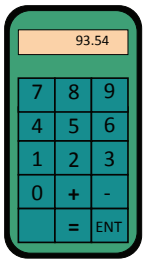


Event Queue
- Mouse moved (t_0,x,y)
- **Mouse pressed (t_1,x,y,1)**
- **Mouse dragged (t_2,x,y,1)**
- **Key typed (t_3, 'F1')**
- **…**

(queues and dispatches incoming events in a dedicated thread)

```
/* callback for TextArea */
public void mouseMoved(e) {
    // process mouse moved event
}
```

Interactor Tree



- Display Screen
  - Outer Win [*black*]
    - Inner Win [*green*]
      - Result Win [*tan*]
        - Result String
      - Keypad [*Teal*]
        - = button
        - - button
        - + button
        - **0** button

Mouse/Touch vs. Keyboard Events

Mouse Events are (usually) routed to the top-most (in z-order) visible component underneath the cursor using **hit testing.**

Exception: "captured" mouse events after beginning interaction

Keyboard events are (usually) routed to the component that has **key focus**.

Exceptions: keys that change focus, accelerator keys

Event Dispatch in ObjC / Cocoa

Mouse events:

Dispatched to NSView of object under cursor

Keyboard events:

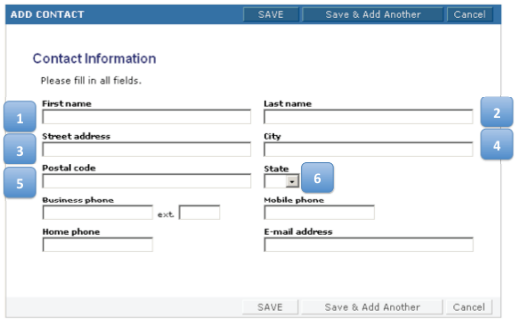
Dispatched to "first responder" (i.e., object in focus)

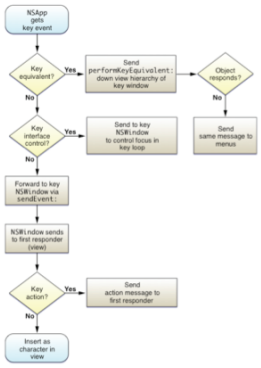
Default NSView implementation does not handle, forwards to "next responder":

"the event, if not handled, proceeds up the view hierarchy to the NSWindow object representing the window itself." (Apple)

If view does

Key Focus: Form Example





Threading Issues

To maintain responsiveness, expensive or I/O-bound computation should execute in a separate thread.

Examples: progress bar animation, loading from URLs

However, changes to the UI are usually only permitted in the main event-dispatching thread!

Many UI frameworks have utility functions to do this:

ObjC: performSelectorOnMainThread:

Java Swing: SwingUtilities.invokeLater()

Single Tap vs. Double Tap (or Click)

How should the application
be notified of events that have duration?

Graphics: Apple iPhone Programming Guide

Single Tap vs. Double Tap (or Click)

Option 1: Two separate events

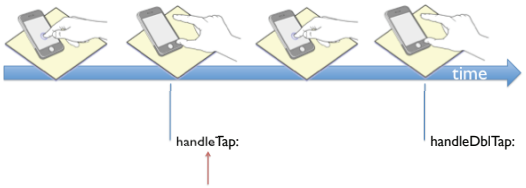
Graphics: Apple iPhone Programming Guide

Single Tap vs. Double Tap (or Click)

Option 1: Two separate events

Graphics: Apple iPhone Programming Guide

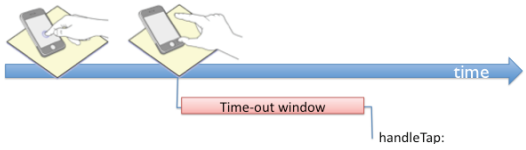
Single Tap vs. Double Tap (or Click)



How do you prevent this?

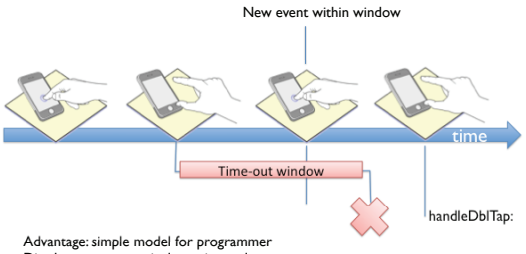
Graphics: Apple iPhone Programming Guide

Single Tap vs. Double Tap (or Click)



Graphics: Apple iPhone Programming Guide

Single Tap vs. Double Tap (or Click)

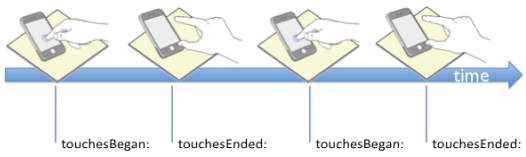


Advantage: simple model for programmer
Disadvantage: every single tap incurs latency

Graphics: Apple iPhone Programming Guide

Single Tap vs. Double Tap (or Click)

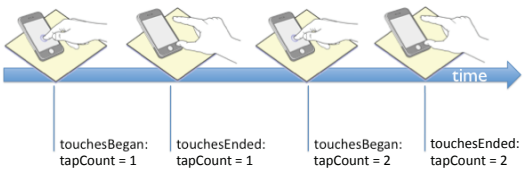
Option 2: Let the programmer deal with it.



Graphics: Apple iPhone Programming Guide

Single Tap vs. Double Tap (or Click)

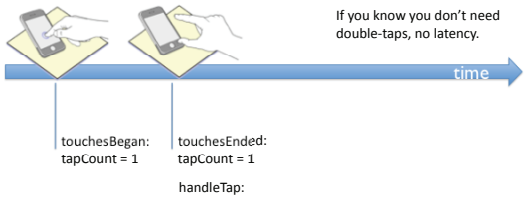
Option 2: Let the programmer deal with it.



Graphics: Apple iPhone Programming Guide

Single Tap vs. Double Tap (or Click)

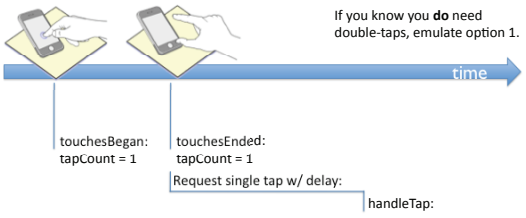
Option 2: Let the programmer deal with it.



Graphics: Apple iPhone Programming Guide

Single Tap vs. Double Tap (or Click)

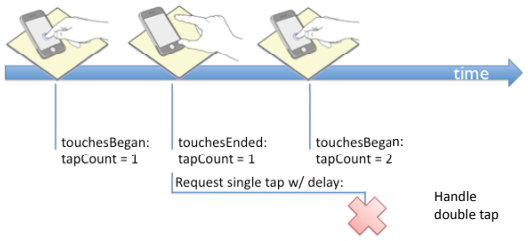
Option 2: Let the programmer deal with it.



Graphics: Apple iPhone Programming Guide

Single Tap vs. Double Tap (or Click)

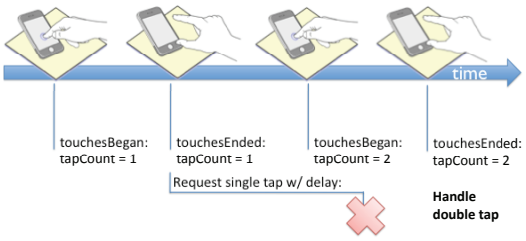
Option 2: Let the programmer deal with it.



Graphics: Apple iPhone Programming Guide

Single Tap vs. Double Tap (or Click)

Option 2: Let the programmer deal with it.



Graphics: Apple iPhone Programming Guide

Model-View-Controller Architecture

Model-View-Controller

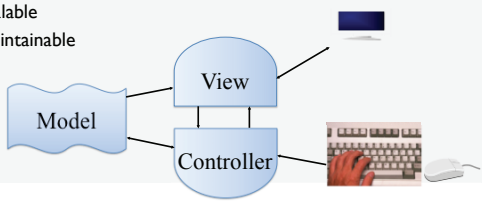
Architecture for interactive apps

introduced by Smalltalk developers at PARC

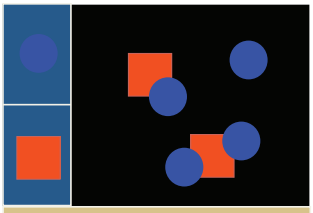
Partitions application in a way that is

scalable

maintainable



Example Application



Blue circles: 4
Cardinal squares: 2

Model

Information the app is manipulating

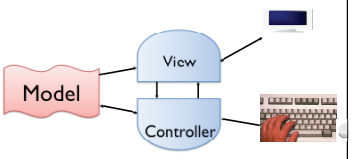
Representation of real world objects

circuit for a CAD program

logic gates and wires connecting them

shapes in a drawing program

geometry and color

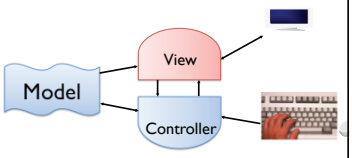


View

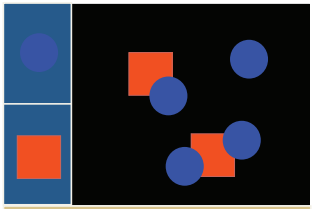
Implements a visual display of the model

May have multiple views

e.g., shape view and numerical view



Multiple Views



Blue circles: 4
Cardinal squares: 2

View

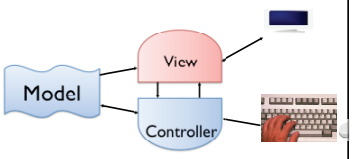
Implements a visual display of the model

May have multiple views

e.g., shape view and numerical view

Any time model changes each view must be notified so it can update

e.g., adding a new shape



Controller

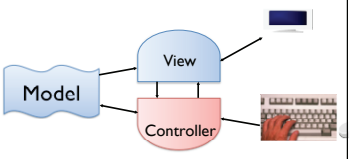
Receives all input events from the user

Decides what events mean and what to do

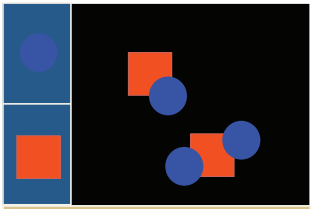
communicates with view to determine the objects being manipulated (e.g., selection)

calls model methods to make changes on objects

model makes change and notifies views to update

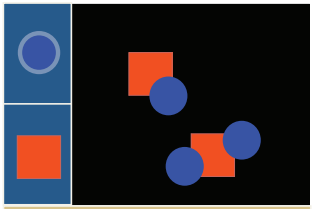


Controller



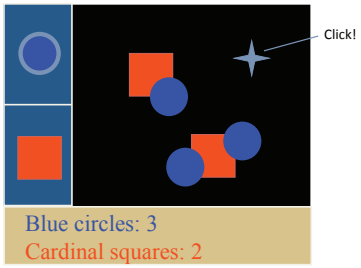
Blue circles: 3
Cardinal squares: 2

Controller

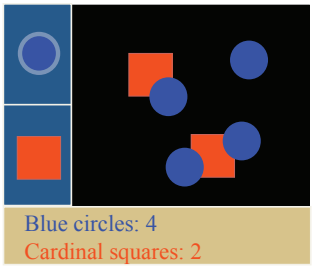


Blue circles: 3
Cardinal squares: 2

Controller



Controller

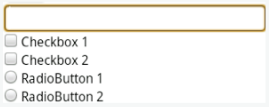


Relationship of View & Controller

"pattern of behavior in response to user events (controller issues) is independent of visual geometry (view issues)"
– Olsen, Chapter 5.2

Relationship of View & Controller

"pattern of behavior in response to user events (controller issues) is independent of visual geometry (view issues)"
– Olsen, Chapter 5.2



But controller must usually contact view to interpret what user events mean (e.g., selection)

Combining View & Controller

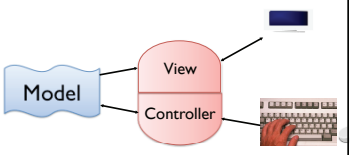
View and controller are tightly intertwined

lots of communication between the two

Almost always occur in pairs

i.e., for each view, need a separate controller

Many architectures combine into a single class



Why MVC?

Why MVC?

Combining MVC into one class will not scale

model may have more than one view

each is different and needs update when model changes

Separation eases maintenance and extensibility

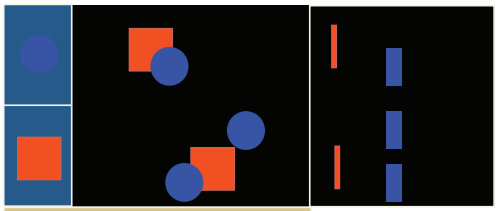
easy to add a new view later

model info can be extended, but old views still work

can change a view later, e.g., draw shapes in 3D

flexibility of changing input handling when using separate controllers

Adding Views Later

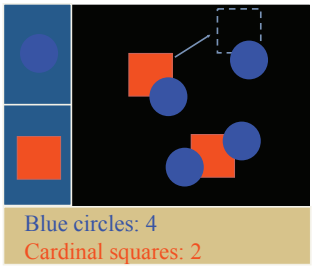


Blue circles: 4
Cardinal squares: 2

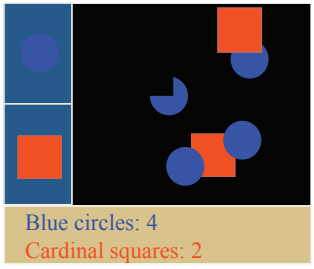
Changing the Display

How do we redraw graphics when a shape moves?

Moving Cardinal Square



Erase w/ Background Color and Redraw



Changing the Display

Erase and redraw

using background color to erase fails

drawing shape in new position loses ordering

Move in model and then redraw view

change position of shapes in model

model keeps shapes in a desired order

tell all views to redraw themselves in order

- slow for large / complex drawings
- flashing! (can solve w/ double buffering)

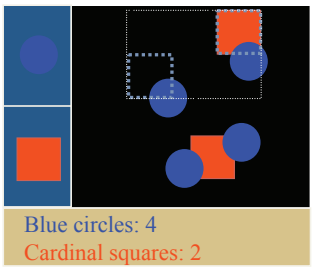
Damage / Redraw Method

View informs windowing system of areas that are damaged
does not redraw them right away…

Windowing system
batches updates
clips them to visible portions of window

Next time waiting for input
windowing system calls Repaint() method
passes region that needs to be updated

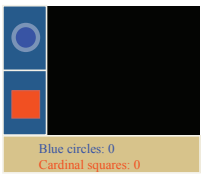
Damage old, Change position in model, Damage new



Event Flow

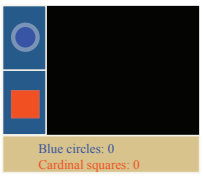
Creating a new shape

Event Flow (cont.)



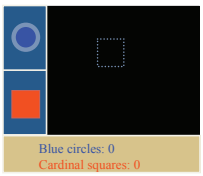
Assume blue circle selected

Event Flow (cont.)



- Press mouse over tentative position
- Windowing system identifies proper window for event
- Controller for drawing area gets mouse click event
- Checks mode and sees "circle"
- Calls model's AddCircle() method with new position

Event Flow (cont.)

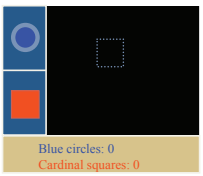


AddCircle adds new circle to model's list of objects

Model then notifies list of views of change
drawing area view and text summary view

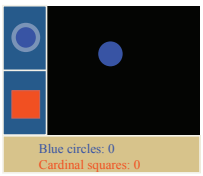
Views notifies windowing system of damage
both views notify WS without making changes yet!
model may override

Event Flow (cont.)



Views return to model, which returns to controller
Controller returns to event handler
Event handler notices damage requests pending and responds
If one of the views was obscured, it would be ignored

Event Flow (cont.)



Event handler calls views' Repaint() methods with damaged areas
Views redraw all objects in model that are in damaged area

Dragging at Interactive Speeds

Damage old, move, damage new method may be too slow

must take less than ~100 ms to be smooth

Solutions

don't draw object, draw an outline (cartoon)

save portion of frame buffer before dragging

draw bitmap rather than redraw the component

modern hardware often alleviates the problem

Summary

Event-Driven Interfaces

Hierarchy of components or widgets

Input events dispatched to components

Components process events with callback methods

Model-View-Controller

Break up a component into

Model of the data backing the widget(s)

View determining the look of the widget

Controller for handling input events

Provides scalability and extensibility

Looking Forward

Containment hierarchy model is now over 20 years old, designed in a context of significantly less processing and graphics power.

Dominant model in use today, and still quite useful, but in many cases limiting.

Limitations:

Assumes rectangular components

Limited support for animation

Level of extensibility (varies by toolkit)

Suitability for next-generation interfaces?

Next Time

Low-Fidelity Prototyping

Prototyping for Tiny Fingers.
Marc Rettig, CACM.

Don't forget to read and submit comment!

Continue work on Programming Assignment IV!