Due

Today: Contextual Inquiry
Mar 3: Program Assignment 4

Contextual Inquiry

BART System Map
Review: Input Device Design Space

Table 1. Physical Properties Used by Input Devices

<table>
<thead>
<tr>
<th>Linear</th>
<th>Rotary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Movement</td>
<td>Rotation</td>
</tr>
<tr>
<td>Force</td>
<td>Delta Force</td>
</tr>
<tr>
<td>Absolute</td>
<td>Relative</td>
</tr>
</tbody>
</table>

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

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

---

Review: Which is faster?

---

Review: Bandwidth of Human Muscle Groups

---

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 10.4 Tiger.
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

In the beginning...

Interactive Application Programming
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

//Objective-C:
int userInput;
NSLog(@"Enter a number:" );
scanf("%i", &userInput );
NSLog(@"You typed %i.", userInput );
// Java Example:
Console console = System.console();
String name = console.readLine("Your name:");
System.out.println("You have entered: " + name);

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

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

Java Swing Widgets

Windows Vista Widgets
Mac Cocoa Widgets

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)

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

Label
TextArea
Buttons

How might we instruct the computer to generate this layout?
Absolute Layout

Label

TextArea

Buttons

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

(Almost) No Layout

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)

NORTH
CENTER
SOUTH

“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.

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

Slider

- `onMouseOver(Event e)[]`
- `onMouseUp(Event e)[]`
- `on.MouseDown(Event e)[]`
- `onMouseClick(Event e)[]`
- `onMouseDrag(Event e)[]`
Event Dispatch Loop

Event Dispatch

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

Option 1: Two separate events

```java
handleTap:
```
Single Tap vs. Double Tap (or Click)

Option 1: Two separate events

HandleTap:

How do you prevent this?

Advantage: simple model for programmer
Disadvantage: every single tap incurs latency
Single Tap vs. Double Tap (or Click)

Option 2: Let the programmer deal with it.

If you know you don't need double-taps, no latency:

- `touchBegan: touchCount = 1`
- `touchEnded: touchCount = 1`
- `handleTap:`

If you do need double-taps, emulate option 1:

- `touchBegan: touchCount = 1`
- `touchEnded: touchCount = 1`
- `touchBegan: touchCount = 2`
- `touchEnded: touchCount = 2`
- `Request single tap w/ delay`, `handleTap:`
Single Tap vs. Double Tap (or Click)

Option 2: Let the programmer deal with it.

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
"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?

- 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?

Erase w/ Background Color and Redraw

Blue circles: 4
Cardinal squares: 2

Moving Cardinal Square

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

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

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

Blue circles: 0
Cardinal squares: 0

Blue circles: 0
Cardinal squares: 0
Dragg 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!