How many of you have implemented a command-line user interface?

How many of you have implemented a graphical user interface?
- HTML/CSS
- Java Swing
- .NET Framework
- Mozilla’s XUL
- Mobile platform (iOS, Android, Blackberry, ...)
- Something else?
What’s the difference?

Command-line model (e.g., UNIX shell, DOS)
– Interaction controlled by system
– User queried when input is needed

Event-driven model (e.g., GUIs)
– Interaction controlled by the user
– System waits for user actions and then reacts
– More complicated programming and architecture
– Need to build the “look” and “feel” of interface

“Look” of a GUI

Bitmap (“Raster”) Display
2D, origin usually at top-left, units vary (often pixels)

Graphics Context
  Device-independent drawing abstraction
    • Clipping region
    • Color
    • Typefaces
    • Stroke model
    • Coordinate transforms

Rendering methods
  • Draw, fill shapes
  • Draw text strings
  • Draw images

Component/Container Model

Component (aka widget, control, etc.)
– Encapsulation of an interactive element
– Drawn using the 2D graphics library
– Low-level input event processing
– Repaint management
– In OOP systems, each component is implemented as a sub-class of a base “Component” class
Name some components?

- Button
- Checkbox
- Radio button
- Text box
- Combo box (drop-down list)
- List box
- Slider
- Menu
- Menu item
- NumericPicker
- DateTimePicker
- more...

Java Swing Components

.NET Framework Controls
HTML Form Controls

Component/Container Model
Container
- Component that contains one or more other components
- Creates the structure of the user interface
- Manages child components
  - Layout, painting, event dispatch
- Some have interactive features (e.g., tab panel)

Container Structure
Container Structure

Containers specify layout of their children
Layout

Containers specify layout of their children

```
Layout
Window
Panel
Panel
Label
Textbox
```

“Feel”: Events

User input is modeled as “events” that must be handled by the system

Examples?
- Mouse
  - button down, button up, button clicked, entered, exited, moved, dragged
- Keyboard
  - key down, key up, key pressed
- Window
  - movement, resizing
- Touchscreen
  - Touching, swiping, dragging, pinching

Anatomy of an Event

An event encapsulates the information needed for handlers to react to the input

- Event type (mouse button down, key up, etc.)
- Event target (component in which event occurred)
- Timestamp
- Modifiers (Ctrl, Shift, Alt, etc.)
- Type-specific content
  - Mouse: x,y coordinates, # clicks
  - Keyboard: key code
Event Handlers

Events are dispatched to components

- Application developers can specify code to be executed when the event occurs (callbacks)
- Built-in components will have code to handle most keyboard and mouse events
  - Buttons handle mouse up/down to change graphic
  - Text boxes update their contents on key press
- Built-in components often generate new “high-level” events from combinations of low-level events
  - Text boxes generate “change” events when contents change and focus is lost
  - Sliders generate “change” events when thumb is dragged

Event Loop

Input Devices

Event Queue

Event Loop

mouse up (10,20)
key down (h)
key up (h)
key down (i)

while (!done) {
  evt = dequeue_event();
  dispatch_event(evt);
  repaint_screen();
}

Input Devices

Event Queue

Event Loop

mouse up (10,20)
key down (h)
key up (h)
key down (i)

Blocks until an event arrives
Event Loop

Input Devices → Event Queue → Event Loop

mouse up (10,20)
mouse down (10,20)
key down ('h')
key up ('h')
key down ('i')

while (!done) {
    evt = dequeue_event();
    dispatch_event(evt);
    repaint_screen();
}

Most of the work happens here

Dispatching Events

mouse down (10,50)

function onMouseDown(evt) {
    // do something...
}

Dispatching Events

mouse down (10,50)

function onMouseDown(evt) {
    // do something...
}
Dispatching Events

```javascript
function onMouseDown(evt) {
    // do something...
}
```

Dispatching Events

```javascript
function onMouseDown(evt) {
    // do something...
}
```

Dispatching Events

```javascript
function onMouseDown(evt) {
    // do something...
}
```
**Events in the Web Browser**

Events are dispatched very much like this within the web browser.

DOM structure of HTML document is used.

Two-stage dispatch process:
- Capture phase: Event is sent down the tree to target.
- Bubbling phase: Event goes back up the tree to the window.

---

**MODEL VIEW CONTROLLER**

- Architecture for interactive apps.
- Partitions application in a way that is:
  - Scalable
  - Maintainable

![Model View Controller Diagram](image)

---

**MVC**

- Architectural design pattern which works to separate data and UI for a more cohesive and modularized system.

- Presented by Trygve Reenskaug in 1979.
- First used in the Smalltalk-80 framework:
  - Used in making Apple interfaces (Lisa and Macintosh).
MVC
• Model represents the data model—"Manages behavior and data of the application domain"
• View represents the screen(s) shown to the user—"Manages the graphical and/or textual output to the portion of the bitmapped display that is allocated to its application"
• Controller represents interactions from the user that changes the data and the view—"Interprets the mouse and keyboard inputs from the user, commanding the model and/or the view to change as appropriate"

Example Application
Blue circles: 4
Cardinal squares: 2

Model
Information the app is trying to manipulate
Representation of real world objects
– Circuit for a CAD program
– Shapes in a drawing program
– List of people in a contact management program
**View**

Implements a visual display of the model
May have multiple views
   – E.g., shape view and numeric 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 numeric view
Any time the model is changed, each view must be notified so that it can update later
Controller

- Receives all input events from the user
- Decides what they mean and what to do
  - Communicates with view to determine the objects being manipulated (e.g., selection)
  - Calls model methods to make changes to objects

Controller

Blue circles: 3
Cardinal squares: 2

Controller

Blue circles: 3
Cardinal squares: 2
Controller

- Combining View & Controller
  - View and controller are tightly intertwined
    - Lots of communication between the two
    - E.g. determine what was clicked on
  - Almost always occur in pairs
    - i.e., for each view, need a separate controller
  - Many architectures combine into a single unit
**Why MVC?**

- Mixing all pieces in one place 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 can be extended, but old views still work
  - Views can be changed later (e.g., add 3D)

**Adding Views Later**

Add circles and squares to illustrate.

**Nesting MVC**

MVC is useful on both large and small scales
- For a whole application
- Within a complex widget
  - Complex components need to store internal state (a model) that affects their drawing and event handling
  - Simplifies internal implementation
  - Allows for code re-use in other components
  - E.g., many Java Swing components have an internal MVC architecture
MVC and the Web

MVC is a very useful architectural style for the web.

Review: How does MVC work?

• User inputs a command
• Controller handles input and updates model or changes the view
• View, which relies on model to show data to user, updates if necessary
• Rinse and Repeat
Review: Why MVC?

• Provides a logical structure for heavily interactive system
• Adheres to good engineering design principles and practices
  – Information hiding, less coupling, simplicity, etc.
  – Delegated control style
• Just plain easier!
  – Studies show that most introductory CS college students tend
to clump a UI program into one big monolithic main class
  – “Unfamiliarity stems from student tendency to program
according to styles presented in textbooks and not being
introduced to design principles early” (Scott F. Morse)

Conclusion

• MVC is a sound architectural design for almost
  any GUI application