

Input

Input Devices

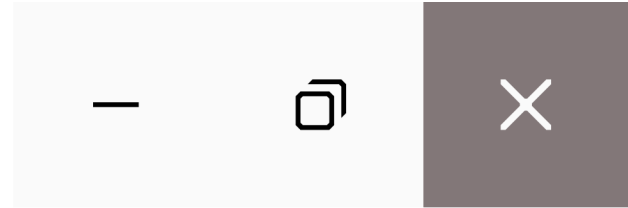
Text Entry

Positional Input

U

CS 349

June 21



U

CS 349

Input Devices

General Purpose Input Devices

Most computing platforms use general purpose input devices

Often targeted at two high level tasks:

- text entry
- positional input



Specific Purpose Input Devices

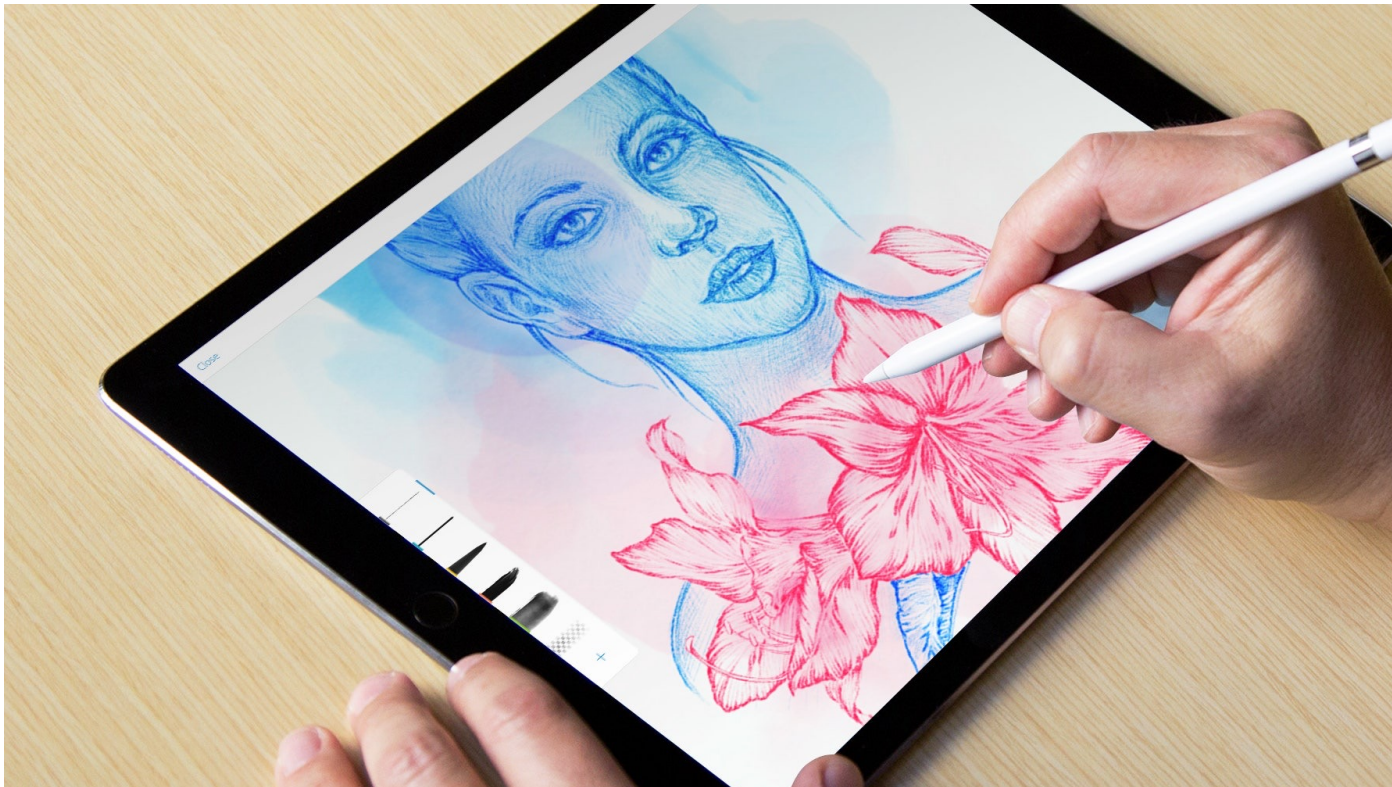
Devices can be designed for very specific UI tasks

- e.g., iPod wheel

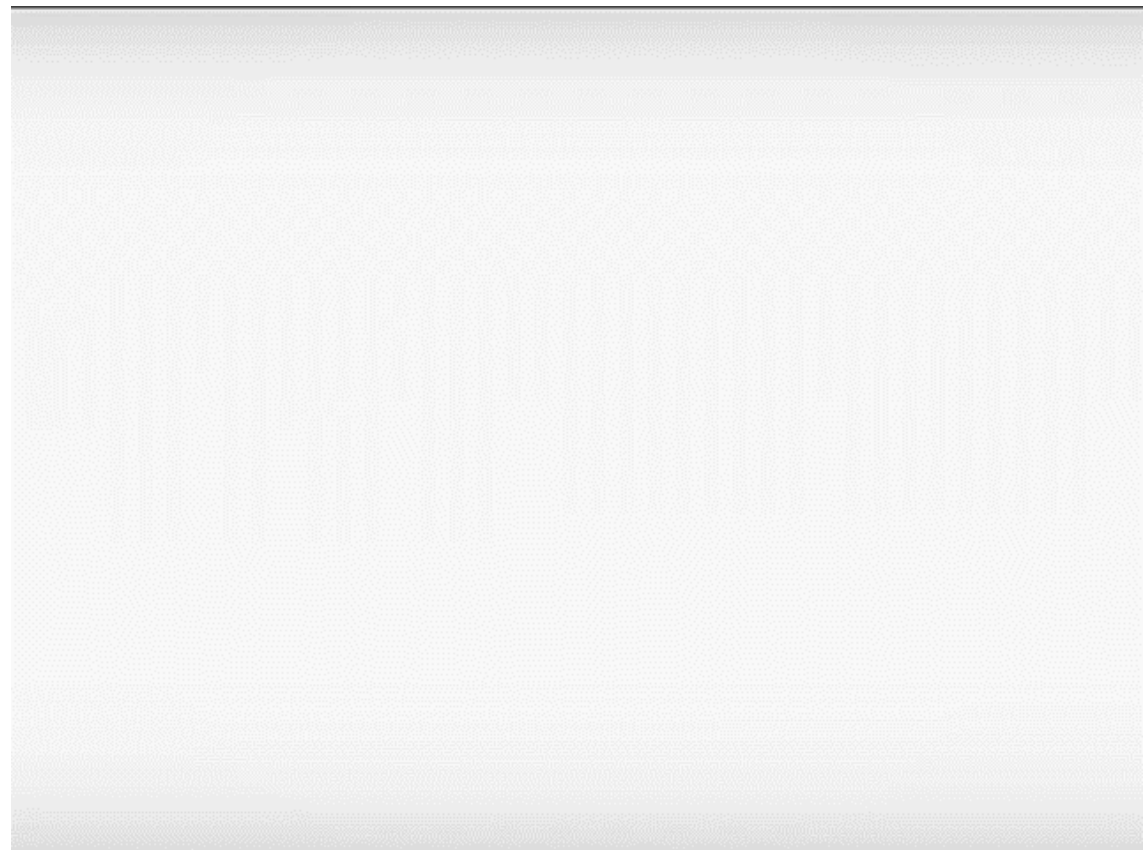


Some UI tasks are better with specific kinds of input devices

- e.g., drawing with a pen vs mouse



Specific Purpose Input Devices





Text Entry

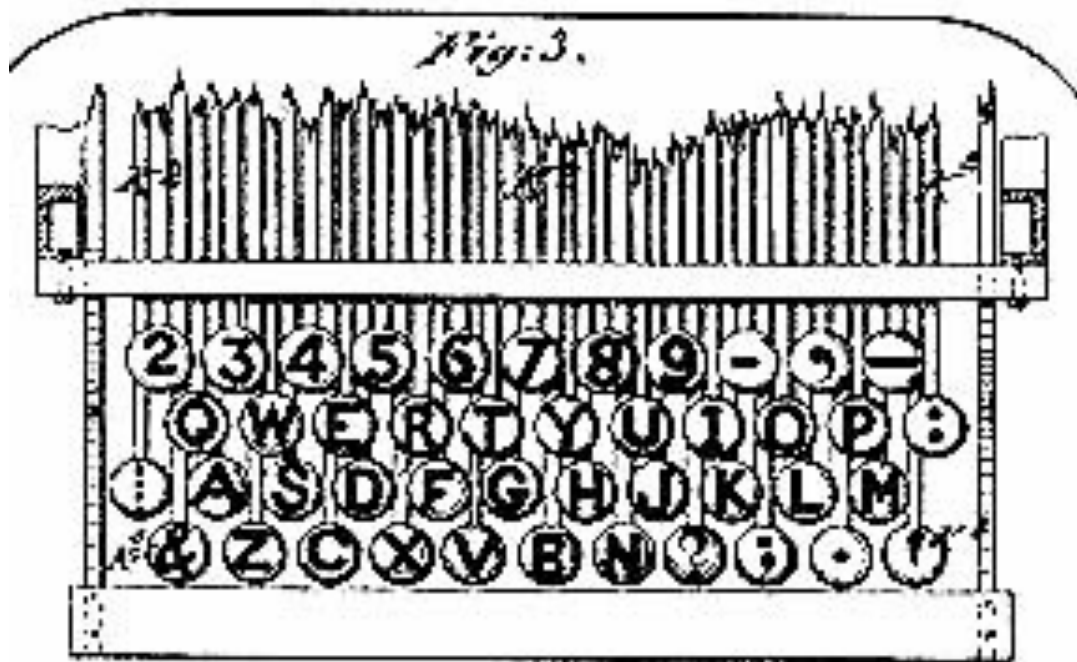
U

CS 349

Typewriters and QWERTY

Original design intended for typing on paper

QWERTY not designed to slow typing down.
Instead, designed to space “typebars” to
reduce jams and speed typing up



1874 QWERTY patent drawing



THE FIRST COMMERCIAL TYPEWRITER
MODEL 1 REMINGTON, SHOP NO. 1.

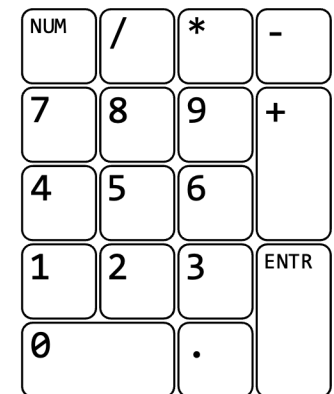
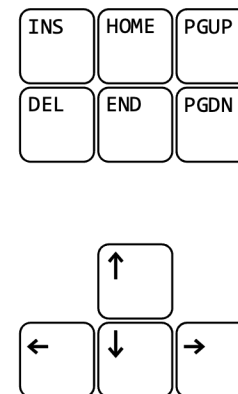
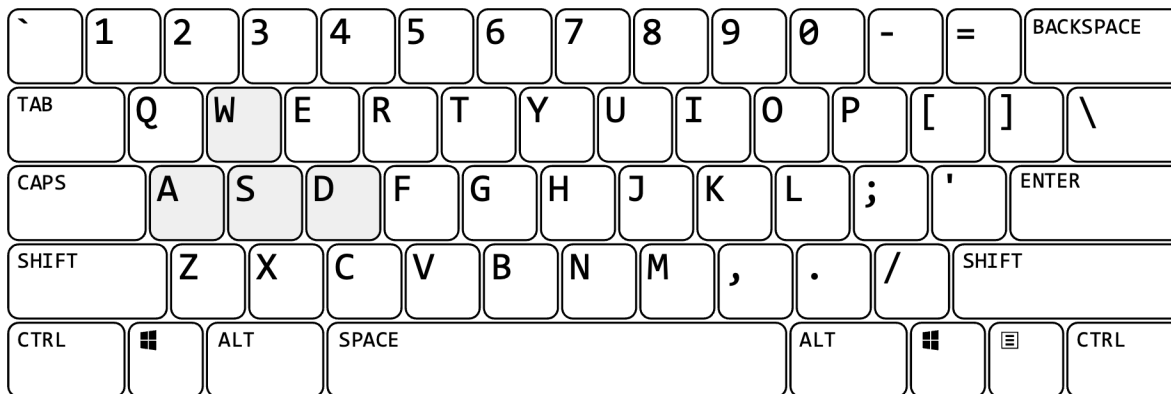
QWERTY Problems?

Common combinations

- awkward finger motions (e.g., t → r)
- jump over home row (e.g., b → r)
- all typed with one hand. (e.g., w → a → s, w → e → r → e)

On average more left-hand typing than right

About 16% of typing uses lower row, 52% top row, 32% home row



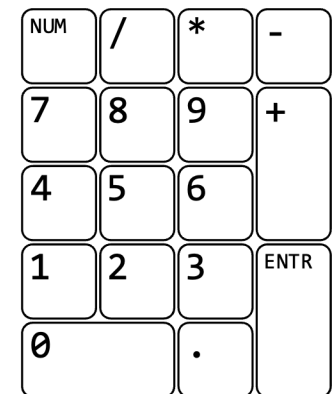
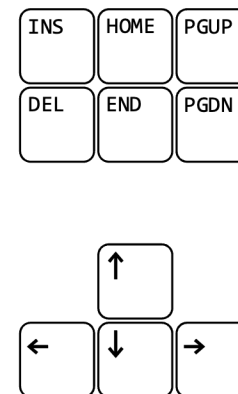
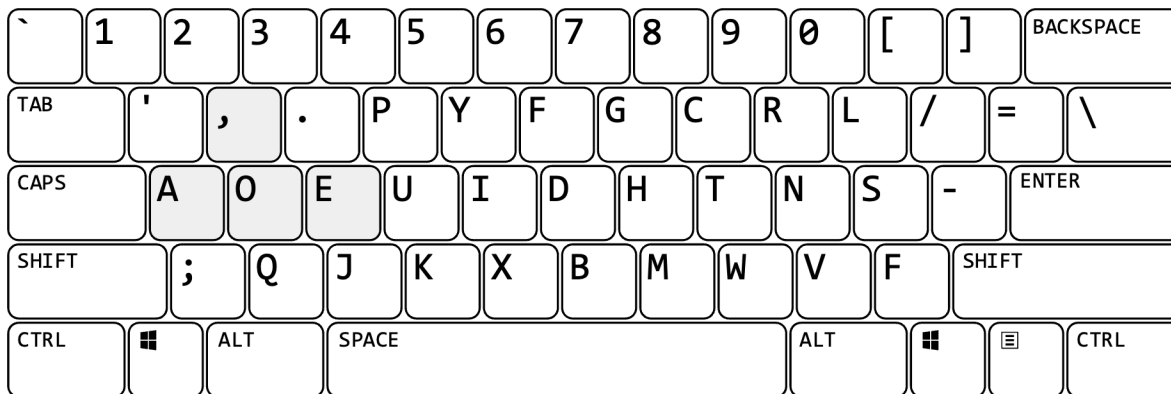
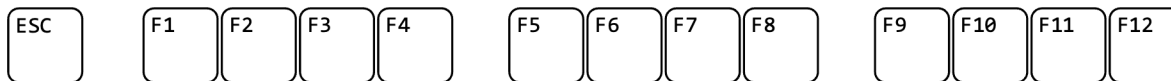
Dvorak Optimizations

Make common letters and digraphs easiest to type

- about 70% of keyboard strokes are on home row
- least common letters on bottom row (hardest row to reach)

Right hand does more typing (assumes most people are right-handed)

Has not caught on . Why?



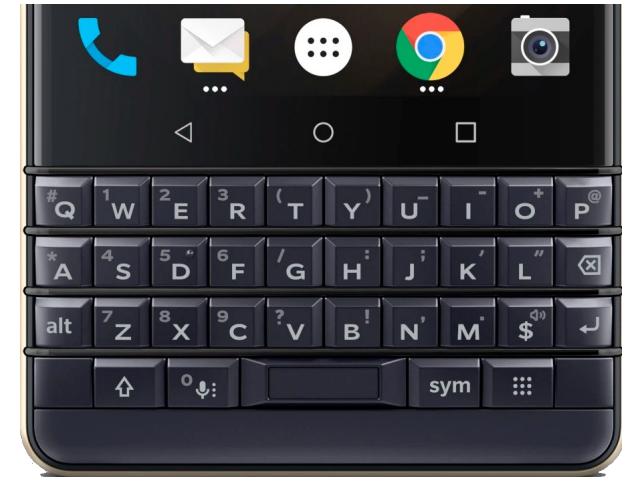
Physical Keyboards

Tactile keys with activation by physical movement.



To reduce cost or increase portability, possible adjustments include:

- Rubber domes instead of springs
- Fewer and / or smaller keys
- Reduced key travel distance



Adjustment can interfere with typing efficiency

Minimal Numeric Keyboard

Repeated presses as text entry method

- Each number is mapped to multiple letters, e.g., $2 \rightarrow \{a,b,c\}$, $5 \rightarrow \{j,k,l\}$, $6 \rightarrow \{m,n,o\}$
- Letters are typed by pressing the associated number multiple times, e.g., $2,2,2 \rightarrow c$
- Words are typed by typing multiple letters, e.g., $2,2 \rightarrow b$, $6,6,6 \rightarrow o$, [pause], $6,6,6 \rightarrow o$, $5,5 \rightarrow k$

Issues common to predictive text

- Generally slow due to number of button presses
- Repeated letters require additional pause



Predictive Text for Minimal Numeric Keyboard

T9 as text entry method

- Each number is mapped to multiple letters, e.g., $2 \rightarrow \{a,b,c\}$, $5 \rightarrow \{j,k,l\}$, $6 \rightarrow \{m,n,o\}$
- Words are typed as a sequence of numbers, e.g., 2-6-6-5
- The word is $\{a,b,c\}-\{m,n,o\}-\{m,n,o\}-\{j,k,l\}$
- Given this (ambiguous) set of characters, the most likely word from a dictionary is displayed, e.g., book over anno, cook, cool, etc.



Issues common to predictive text

- “Collisions” between common ambiguous words
- Entering words not in dictionary is difficult

Touch Keyboards

Problems:

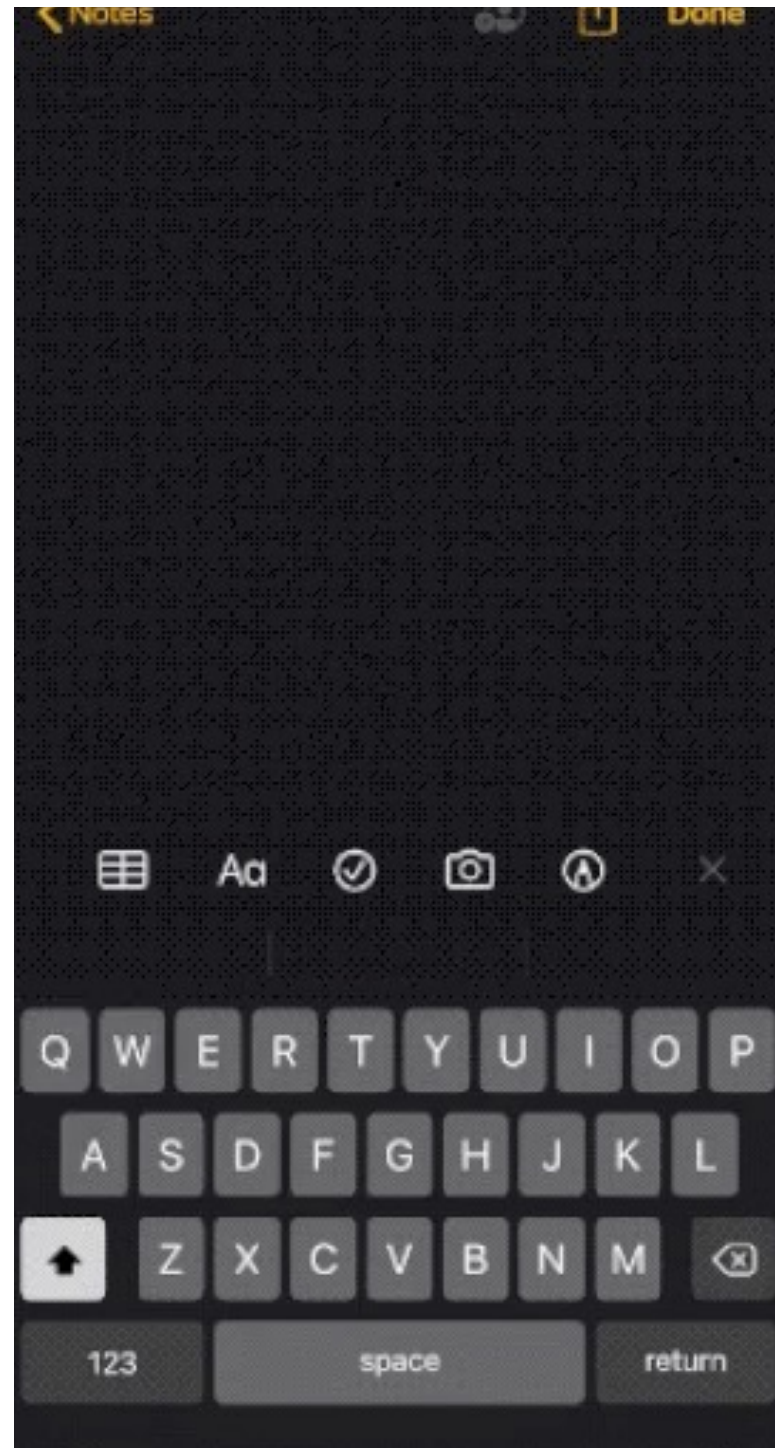
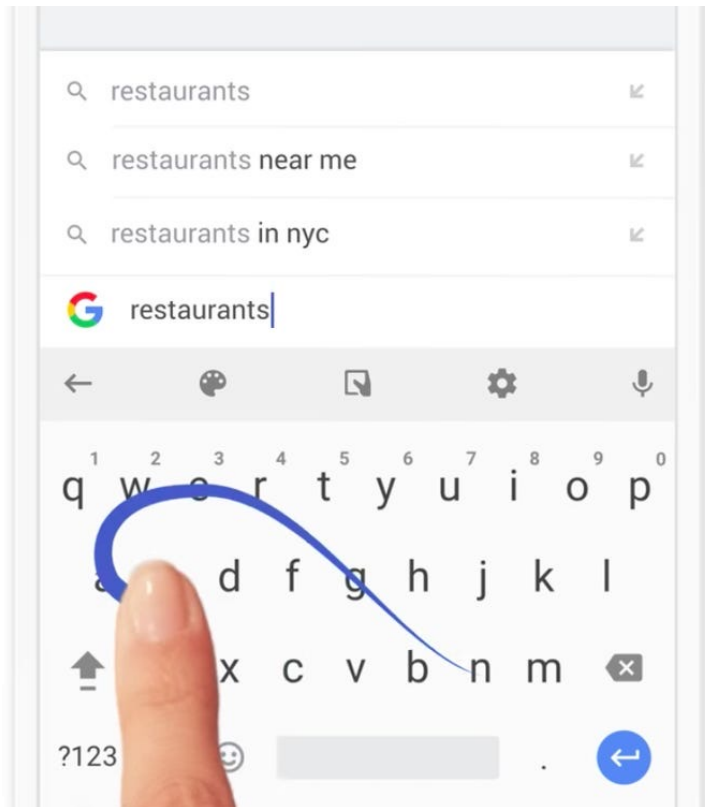
- no tactile feedback makes it hard to find the home row
- no tactile feedback makes it hard to tell if key was pressed (solved by vibration)
- resting of hands difficult
- small keys reduce accuracy

Advantage:

- portable, no extra hardware
- customizable keys (e.g., new language, symbols, emojis)
- customizable layout and functionality (e.g., swipe, thumb layout)



Gestural Text Input



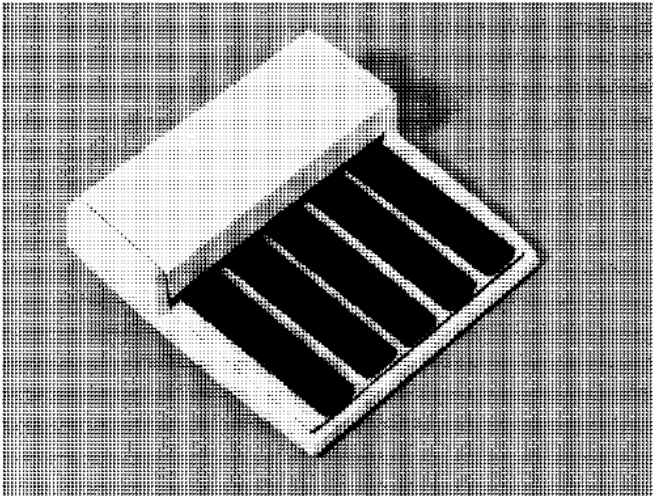
Chording Keyboards

Englebart's NLS Keyboard

- Multiple keys together produce letter
- No hand “targeting”, potentially very fast
- Can be small and portable
- One handed

Thad Starner's Twiddler

- for wearable computing input



NLS Keyboard



Twiddler

Text Recognition and Gestures

Gestural strokes for letter (e.g., Graffiti / Unistroke Gestures)

- Map single strokes to characters



Natural Handwriting recognition (e.g., iPad Scribble, Microsoft Ink to Text)

- dictio



Text Input Expert-User Input Rates

Device	Input Rates
Qwerty Desktop	80+ WPM proficient 150 WPM record (sustained for 50 minutes)
Qwerty Thumb	60 WPM typical with training
Soft Keyboards	45 WPM
T9	45 WPM possible for experts
Gestural	~ ShapeWriter claims 80 WPM (expert)
Handwriting	33 WPM
Graffiti 2	9 WPM

ACII & Unicode

ASCII is a 1-byte encoding of the Latin alphabet.

Unicode is a *superset* of ASCII, that has replaced it in common use

- Values 0-127 have the same meaning in both (e.g., 'A' \Leftrightarrow 65)
- Uses multiple bytes to store character information, which greatly increases the range of values
- Denoted as UTF-xx where xx is the minimum number of bits.

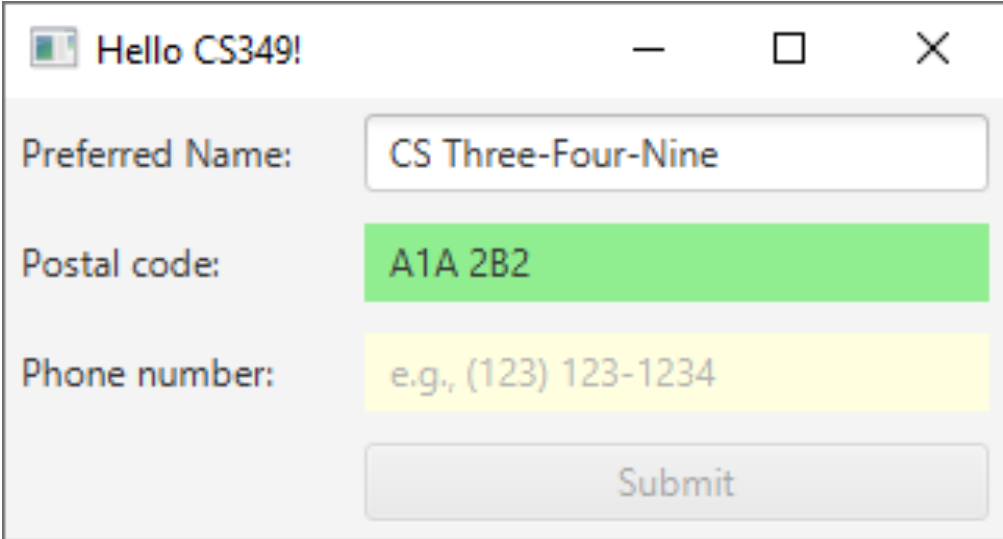
UTF-8 is the standard method of encoding characters

- Minimum 8 bits
- Capable of encoding all 1,112,064 code points in Unicode (characters, control codes, other meaningful characters)

Text Validation

Interfaces often need to check text input typed by the user

- a required field (e.g., credit card number)
- a certain format (e.g., numeric, postal code, phone number)
- within a certain range (e.g., number between 0 and 100)
- unique (e.g., choose an unused username)



A screenshot of a web form titled "Hello CS349!". The form contains three input fields and a "Submit" button. The "Preferred Name" field contains the text "CS Three-Four-Nine". The "Postal code" field contains the text "A1A 2B2" and is highlighted with a green background. The "Phone number" field contains the text "e.g., (123) 123-1234" and is highlighted with a yellow background. The "Submit" button is located at the bottom of the form.

Guidelines for Text Validation

1. Prevent invalid input through constant validation.
2. Accept data formatted in different ways
3. Have different levels of text validation:
 - Basic, in the View
 - Intermediate, in the Model
 - Thorough, in the backend

When input is invalid:

- Place error messages close to the source of the error
- Use colour to differentiate valid from invalid input

Regular Expressions (regex)

A sequence of characters that specifies a search pattern in text

- developed from language theory and theoretical computer science
- a regex pattern describes a deterministic finite automaton (DFA)

Please refer to

- <https://regexone.com> (Regex Tutorial)
- <https://regex101.com> (Regex Testing, Explanation, Reference)

Regular Expressions (regex)

Used in form validation to “test” if string can is correct format:

- Postal Code (upper case only, with space in between the two 3-tuples):

```
[A-Z][0-9][A-Z] [0-9][A-Z][0-9]
```

- Number (North American decimal separators required):

```
^-?[0-9]{1,3}(,[0-9]{3})*(\.[0-9]{1,2})?$$
```

- Phone Number (10 digit North American, with some formatting options):

```
\((?[0-9]{3}\)\)?[ .-]?[0-9]{3}[ .-]?[0-9]{4}
```

Text Validation

Input validation that blocks illegal inputs (via `pcType`) and performs a final check, including colour highlighting (via `pcFinal`):

```
val textInput = TextField().apply {
    promptText = "e.g., A1A 1A1"
    textFormatter = TextFormatter<String> {
        it.text = it.text.uppercase()
        val pcType = Regex("[A-Z]?[0-9]?[A-Z]? ?[0-9]?[A-Z]?[0-9]?")
        val pcFinal = Regex("[A-Z][0-9][A-Z] [0-9][A-Z][0-9]")
        if (pcType.matches(it.controlNewText)) {
            background = Background(BackgroundFill(
                if (pCodeFinal.matches(it.controlNewText))
                    Color.GREEN else Color.YELLOW, null, null))
            it
        } else {
            null
        }
    }
}
```



Positional Input

U

CS 349

Properties of Positional Input Devices

Displacement vs. Force Control

- Mouse = displacement
- (Analog) joystick = force



Properties of Positional Input Devices

Absolute vs. Relative Positioning

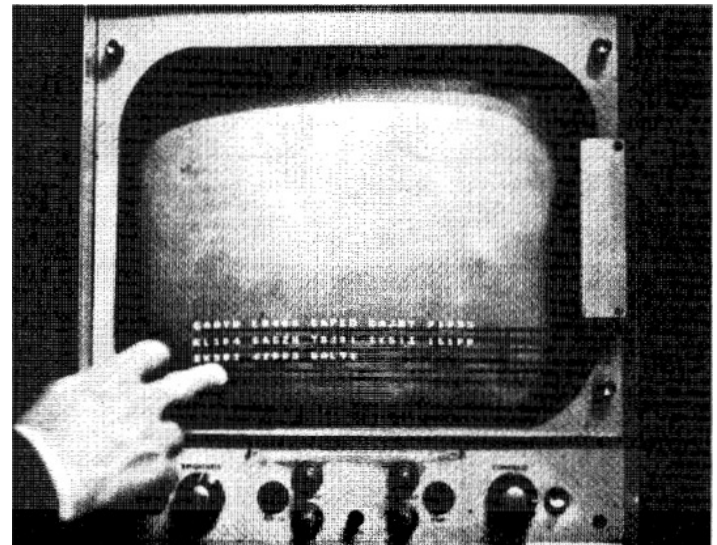
- Mouse = relative
- Touchscreen = absolute

Direct vs. Indirect Contact

- Mouse: indirect
- Touchscreen = direct

Degrees of Freedom (DOF): Number of (continuous) dimensions sensed:

- Mouse: 2 (x,y), 3 (x,y,scroll)
- Touchscreen: 2 (x,y), 3 (x,y,force)
- Joysticks: 2+ (x,y,z,w, ...)



Displacement vs. Force Sensing

Isotonic devices measure displacement, e.g.,

- Mouse: optical sensor (x,y)
- Mouse: scroll wheel (scroll)

Isometric devices measure force, e.g.,

- Mouse: scroll “bar”, scroll “point” (scroll)
- Gamepad: left, right sticks



Direct vs. Indirect Input

Indirect: input position is controlled by a cursor which is controlled by a device away from the display



Direct: input position is controlled by direct contact with the corresponding display position



Absolute vs. Relative Position

Absolute position is a direct mapping of input device position to a display input position



Relative position maps changes in input device position to changes in cursor position



Direct vs. Indirect vs. Absolute vs. Relative Position

Indirect

Direct

Absolute



Relative



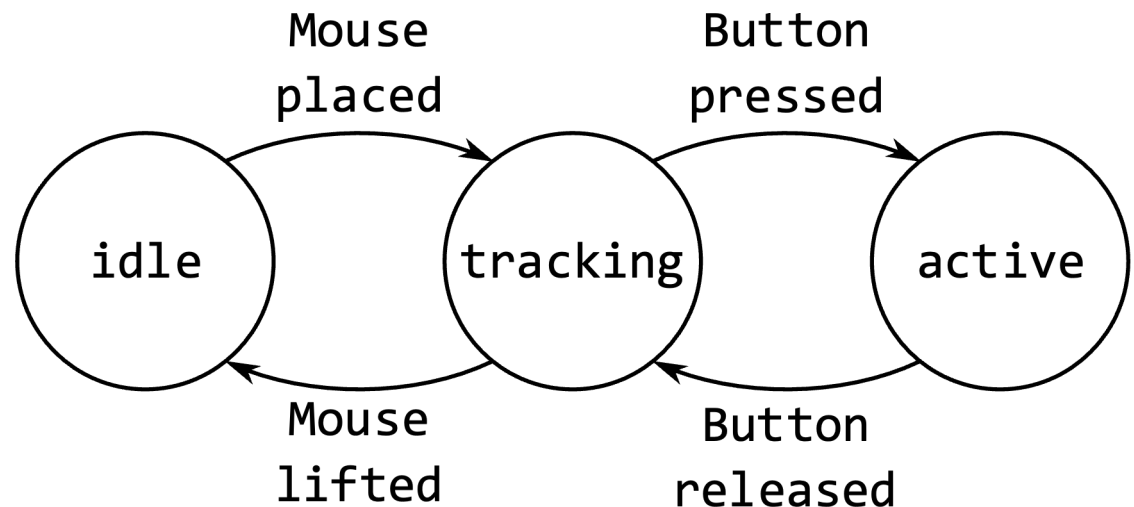
???

Relative Positional Devices: Clutching

Clutching is a method to temporarily disconnect an input device from controlling cursor position, for example, by lifting a mouse and moving it to another location.

Relative positional movements will cause the device to drift and either run out of space or become out of reach.

Clutching is necessary for any kind of relative position control device.

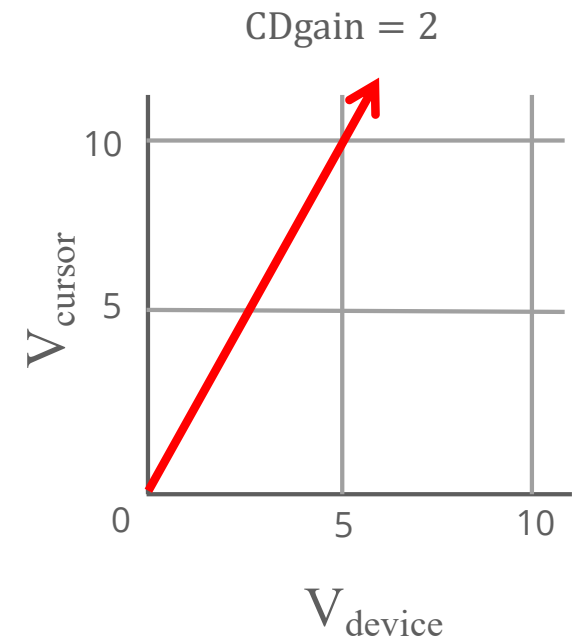
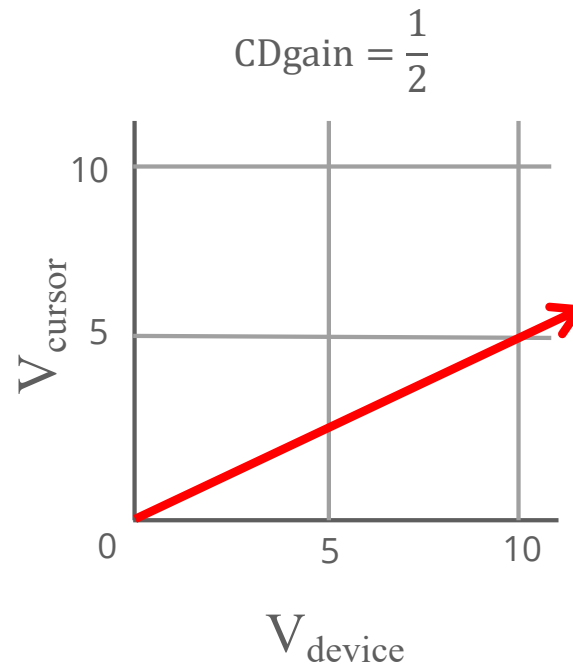
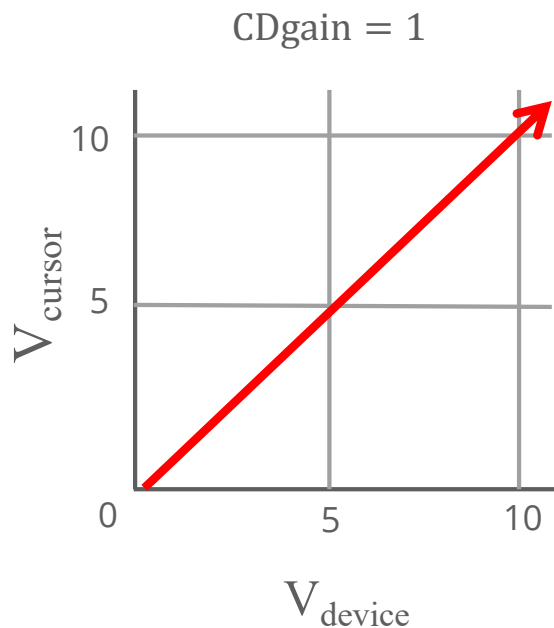


Relative Positional Devices: Control-Display Gain

Control-Display Gain (or CD Gain) is the ratio of **display cursor** movement to **device control** movement

- The ratio is a scale factor (i.e., “gain”)
- Works for relative devices only

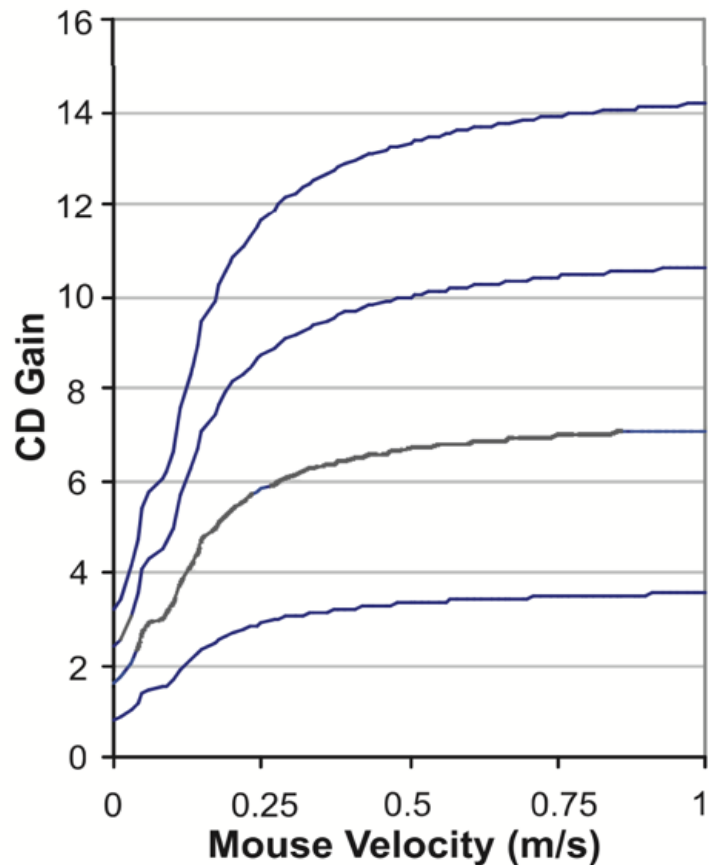
$$CDgain = \frac{v_{cursor}}{v_{device}}$$



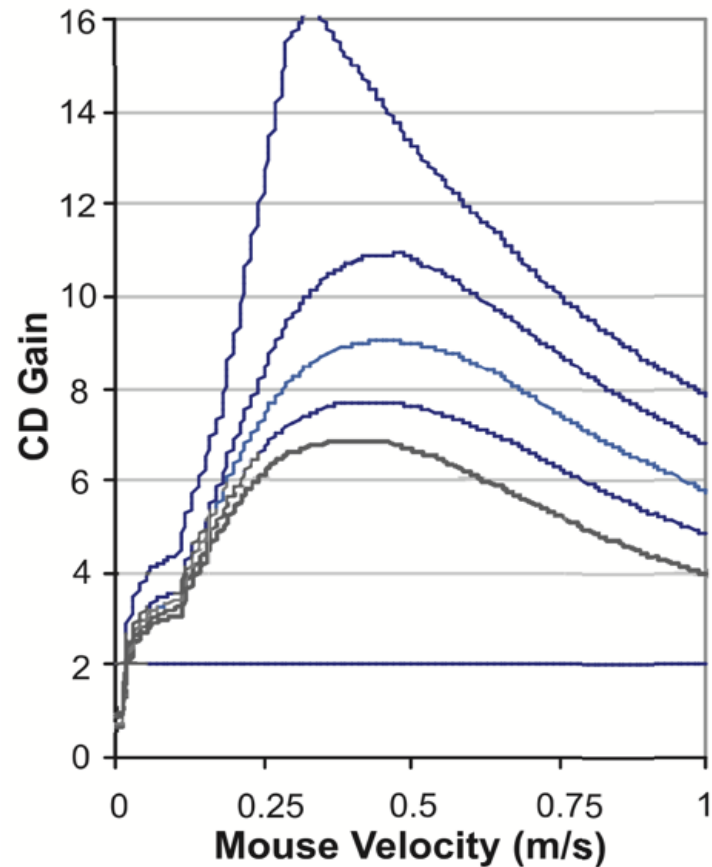
Relative Positional Devices: Control-Display Gain

Dynamically change CD Gain based on device velocity

- can reduce the amount of clutching



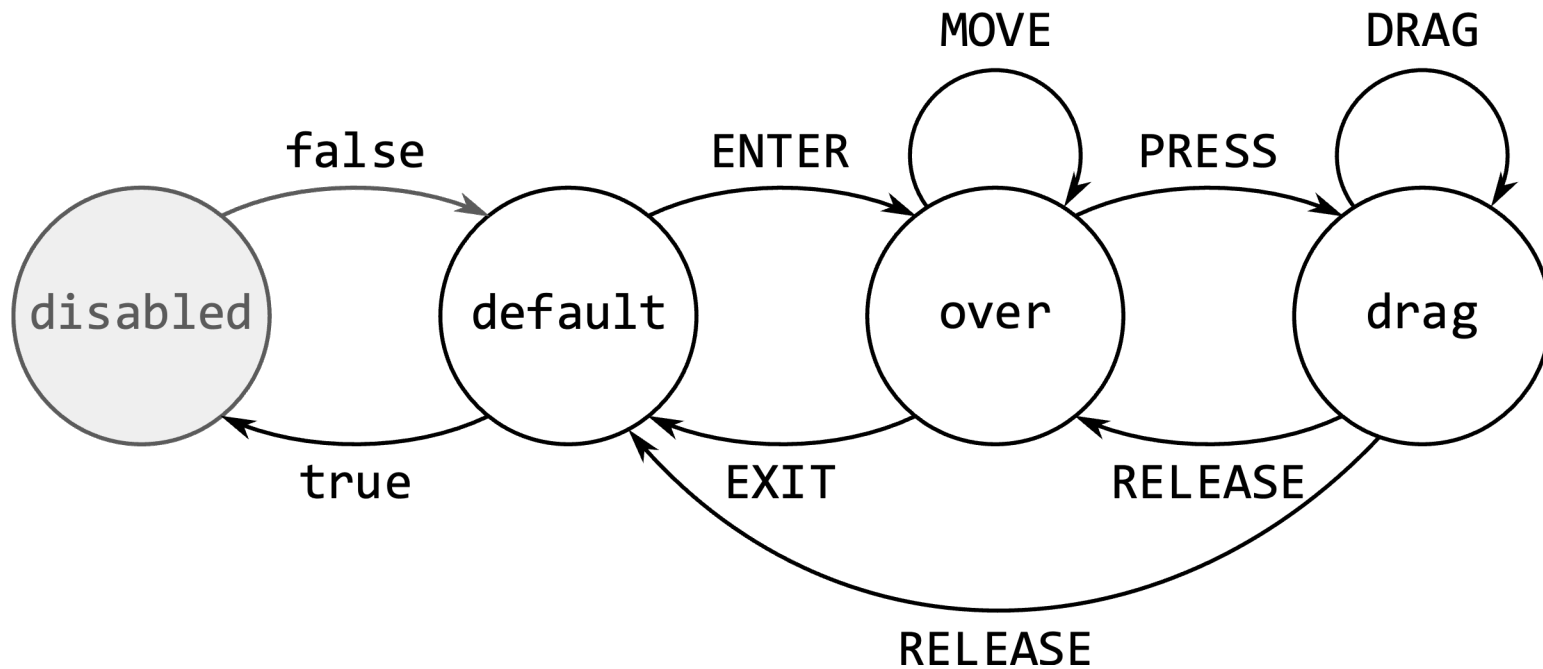
(b) Windows XP/Vista



(c) Mac OSX

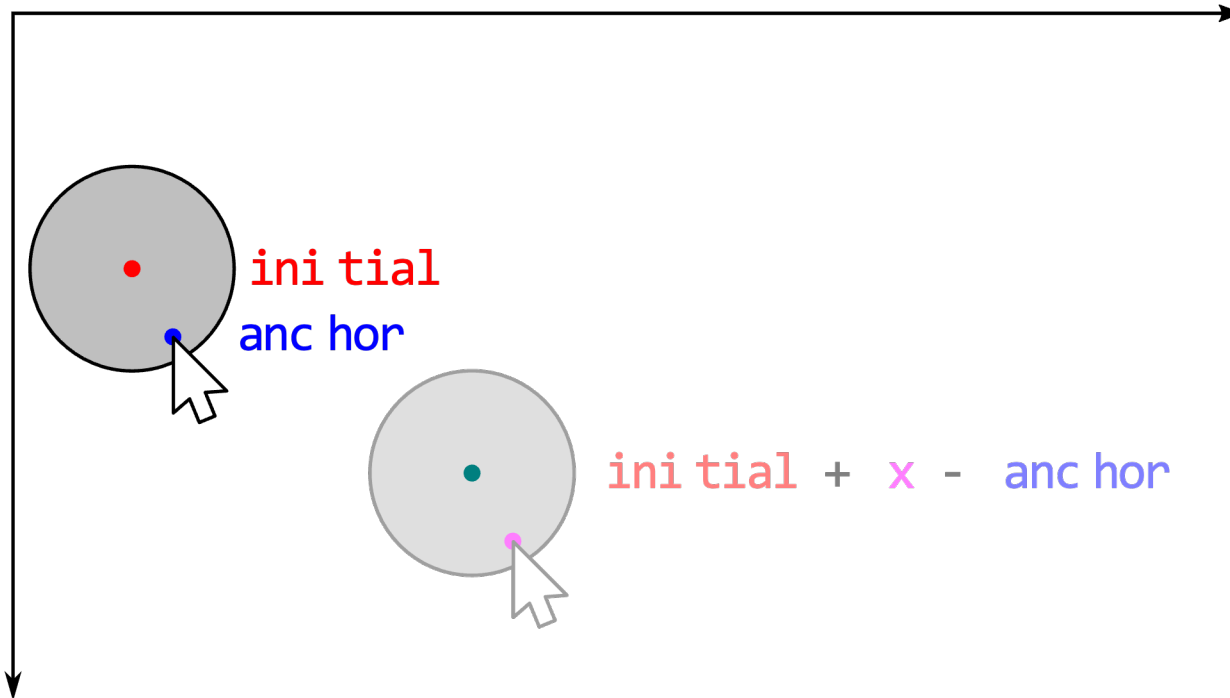
Widget States for Positional Input

- default
- over (hovering)
- down (dragging)
- disabled



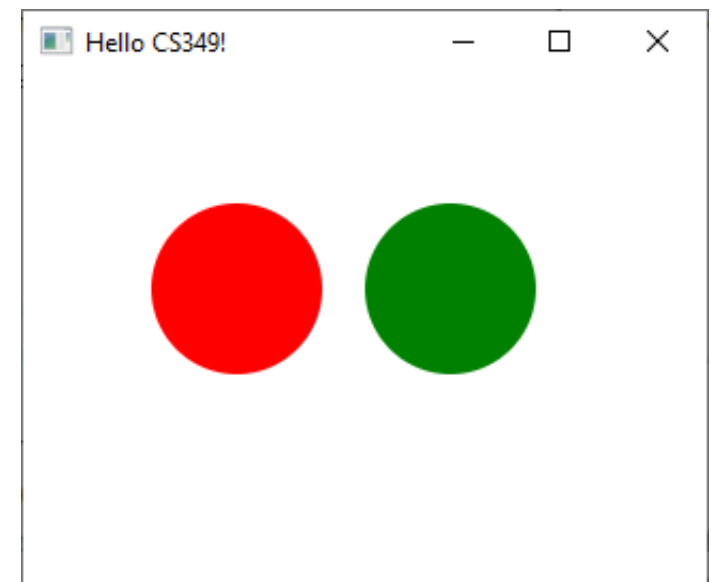
Implementing Dragging

```
data class DragInfo(var target : Shape? = null,  
                    var anchorX: Double = 0.0,  
                    var anchorY: Double = 0.0,  
                    var initialX: Double = 0.0,  
                    var initialY: Double = 0.0)  
  
var dragInfo = DragInfo()
```



Implementing Dragging

```
val makeCircle = { x: Double, y: Double, r: Double, col: Color ->
    Circle(x, y, r, col).apply {
        addEventFilter(MouseEvent.MOUSE_PRESSED) {
            dragInfo = DragInfo(this, it.sceneX, it.sceneY,
                                translateX, translateY)
            viewOrder -= 1000.0
        }
        addEventFilter(MouseEvent.MOUSE_DRAGGED) {
        }
        addEventFilter(MouseEvent.MOUSE_RELEASED) {
            dragInfo = DragInfo()
            viewOrder += 1000.0
        }
    }
}
```



Implementing Dragging

Here is an example on how to use the bubble phase of the parent to modify its child after it has processed an event:

```
val circ1 = makeCircle(100.0, 100.0, 40.0, Color.RED)
val circ2 = makeCircle(200.0, 100.0, 40.0, Color.GREEN)

val pane = Pane(circ1, circ2).apply {
    addEventHandler(MouseEvent.MOUSE_DRAGGED) {
        dragInfo.target?.translateX = dragInfo.target!!.translateX -
            (dragInfo.target!!.translateX % 10)
        dragInfo.target?.translateY = dragInfo.target!!.translateY -
            (dragInfo.target!!.translateY % 10)
    }
}
```

End of the Chapter



- Remember the levels of text validation



Any further questions?