

UI Design Essentials for Developers

- previously
 - how to build an interface in JavaFX
- today
 - how to make an interface that works for the people using it – in a nutshell

You Are Not the User

- you built the interface, so you know –
 - what every control does and what order to use them in
 - which inputs are valid
 - where to click, and that you *can* drag that card
- your users know none of this
- your users may also differ from you –
 - using a screen reader
 - no mouse
 - color vision differences
 - a small screen
 - a first-time player

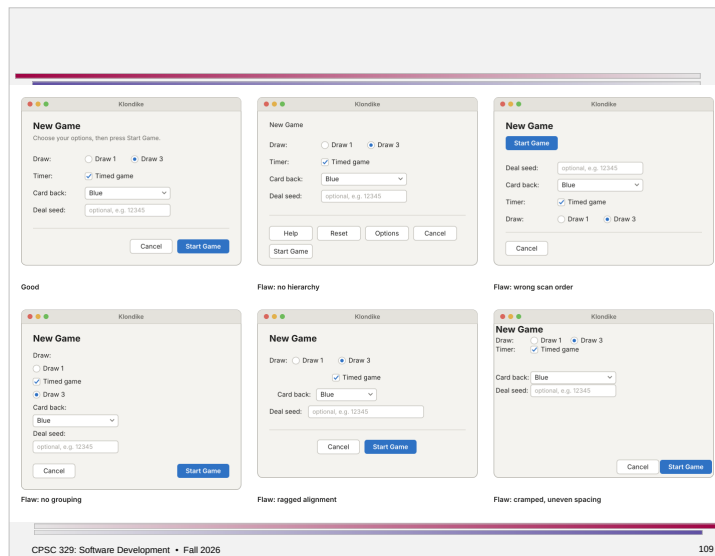
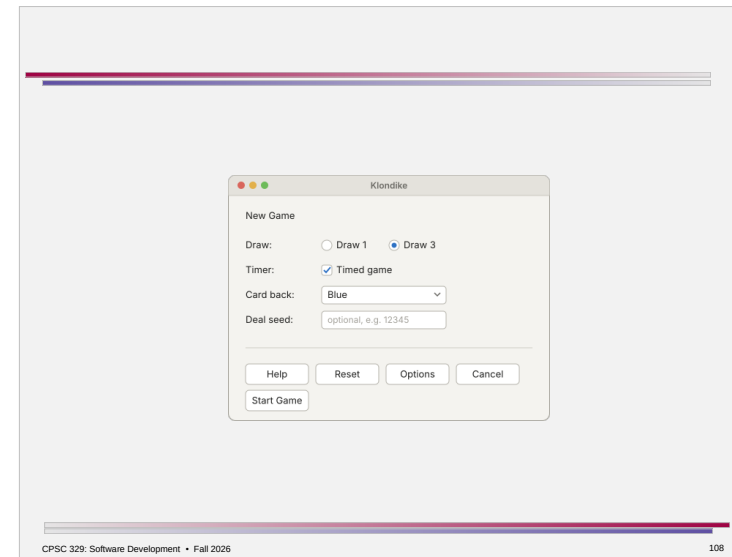
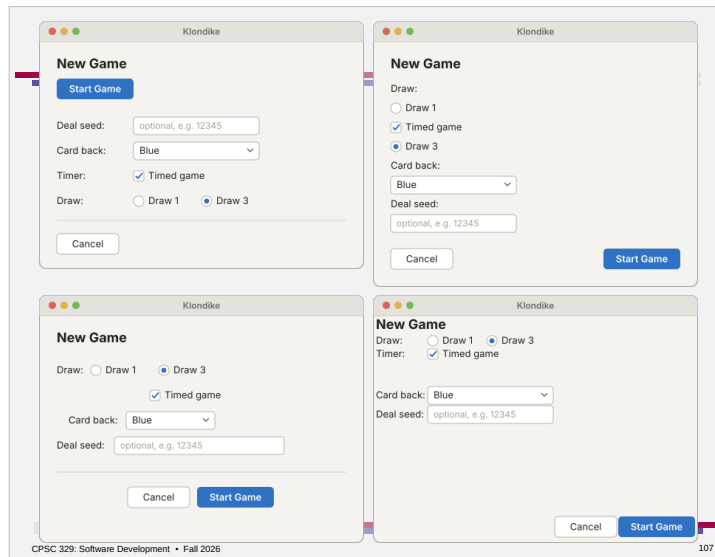
Hallway Test

A mini usability test –

- ask a friend to perform a specific task using your program
 - have the friend think out loud as they work
- watch silently – don't help, explain, or point
 - note every hesitation, wrong click, or question

Design Principles

- visual hierarchy and grouping
- consistency
- feedback and visible state
- errors: prevent first, then explain
- affordances and signifiers

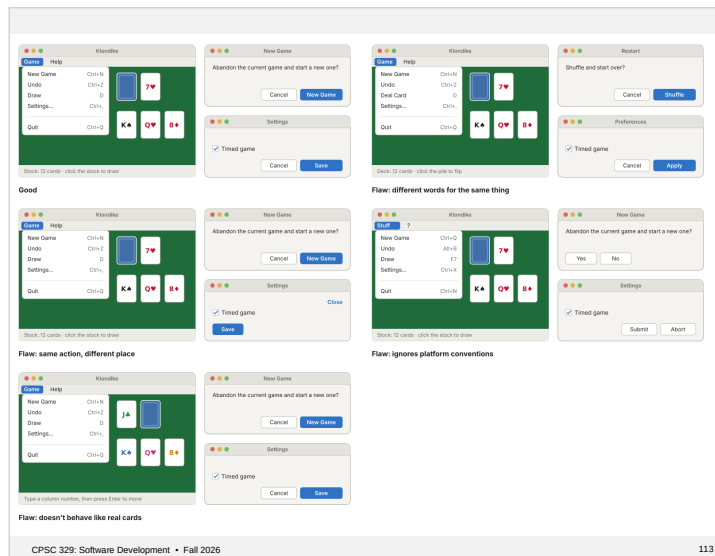
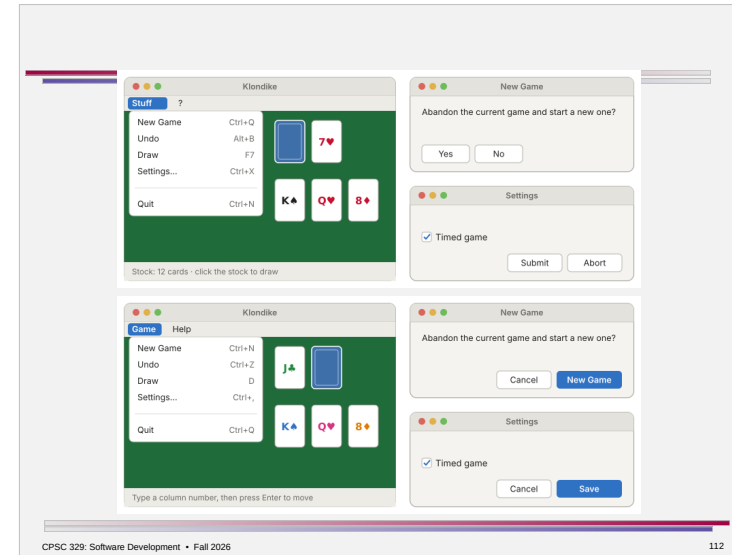
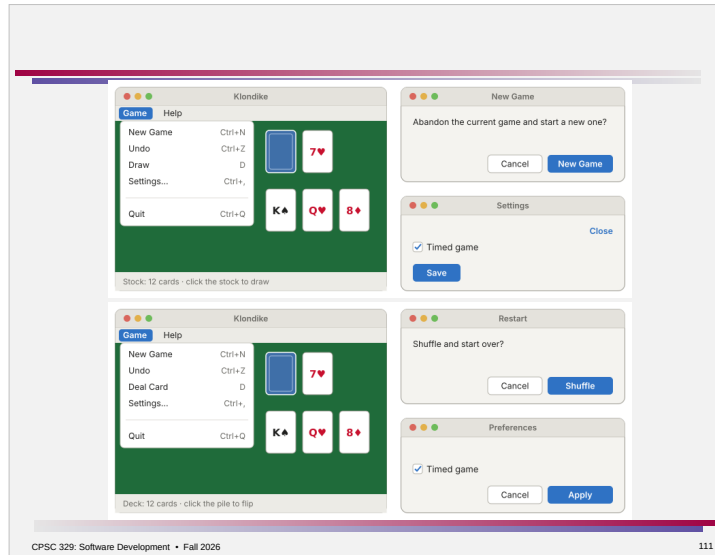


Visual Hierarchy and Grouping

- the most important thing should be the most noticeable
 - use size, weight, position, contrast
- users scan top-left to bottom-right
 - put the start of the task in the top left
 - put the finishing action (start, save) in the bottom right
- group related controls
 - separate groups with space or a Separator
- align labels and fields to a common edge
 - a GridPane with a label column and a field column does this for free
- leverage spacing
 - consistent padding and spacing reads as "designed"

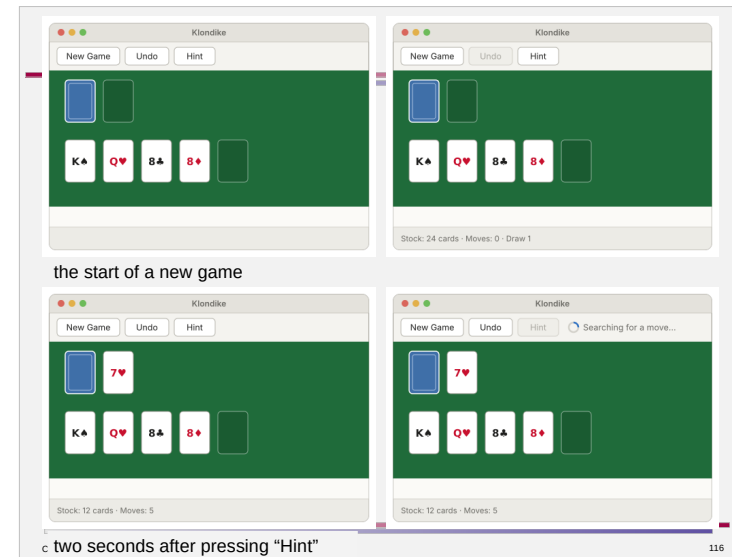
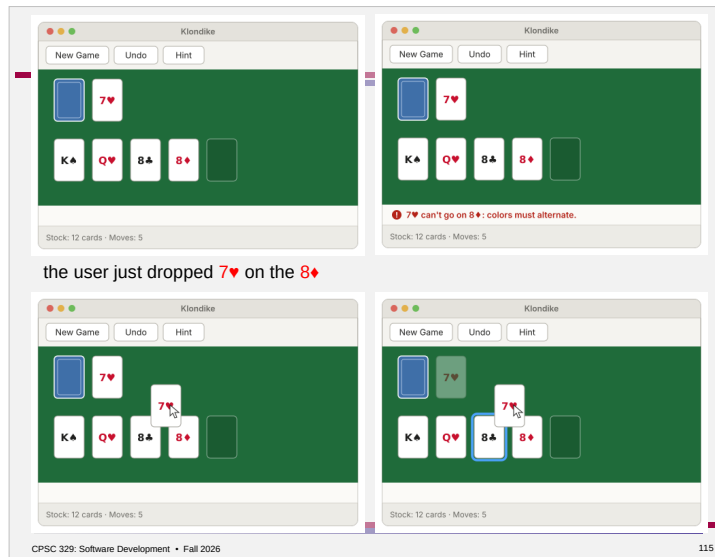
in left-to-right languages –
know your users and follow
cultural conventions





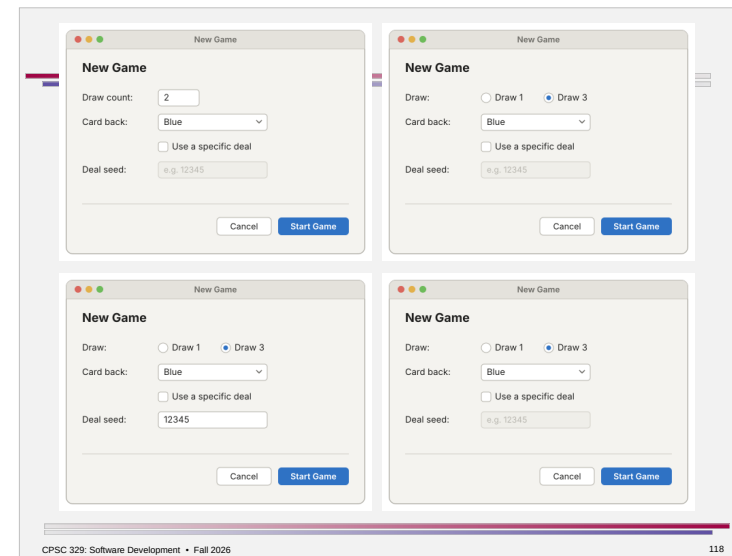
Consistency

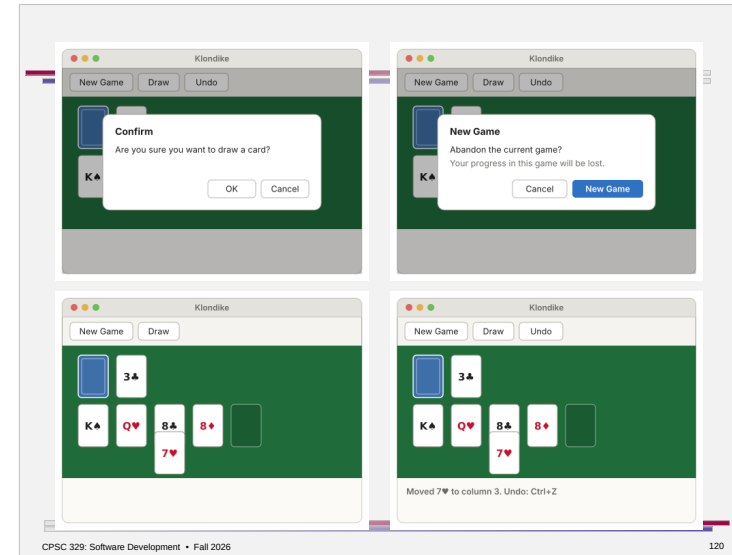
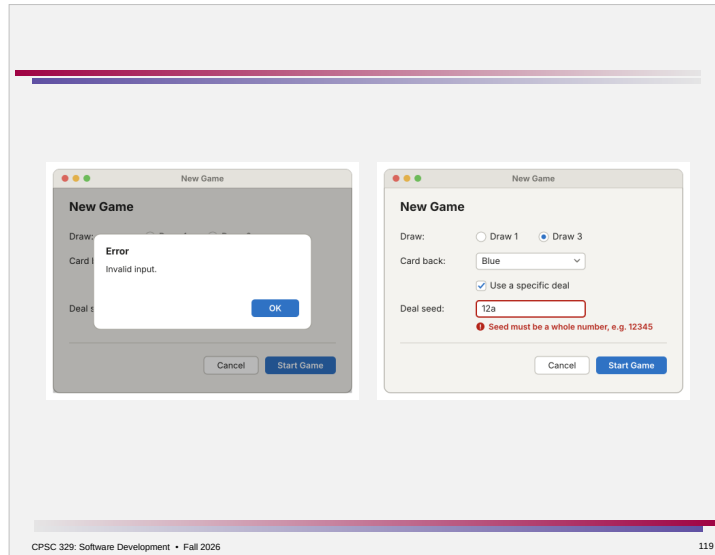
- the same thing looks and behaves the same way everywhere in the application
 - one word per concept – not “draw” in one place and “deal” in another
 - the same action is always in the same place
- follow platform conventions that users already know
 - buttons labeled with verbs
 - Enter = default button, Esc = cancel
 - standard menu names (File/Game, Help) and shortcuts
- maintain consistency with the real world
 - e.g. in Klondike, drag a card to move it – matches picking a card up and putting it down somewhere else
 - e.g. a calendar that looks like a calendar grid
 - e.g. a trash can holds deleted files until it is emptied



Feedback and Visible State

- every action gets a visible response, quickly
 - e.g. hover or drag-enter highlight
 - e.g. a message when a move is refused, saying why
 - e.g. a disabled button when an action isn't available
- show the current state so the user never has to remember it – recognition rather than recall
 - e.g. a label "draw 3 – 12 cards left" showing what is left in the stock pile
 - e.g. what's selected
 - e.g. whose turn it is
- never fail silently
- long operations show progress

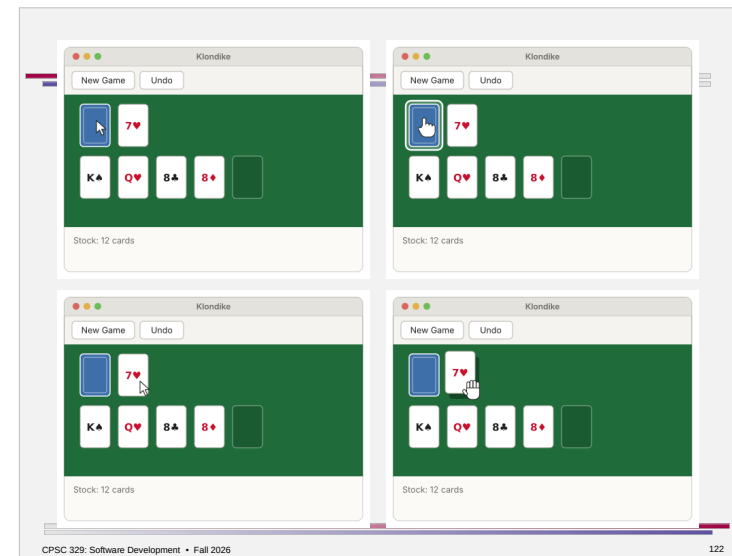


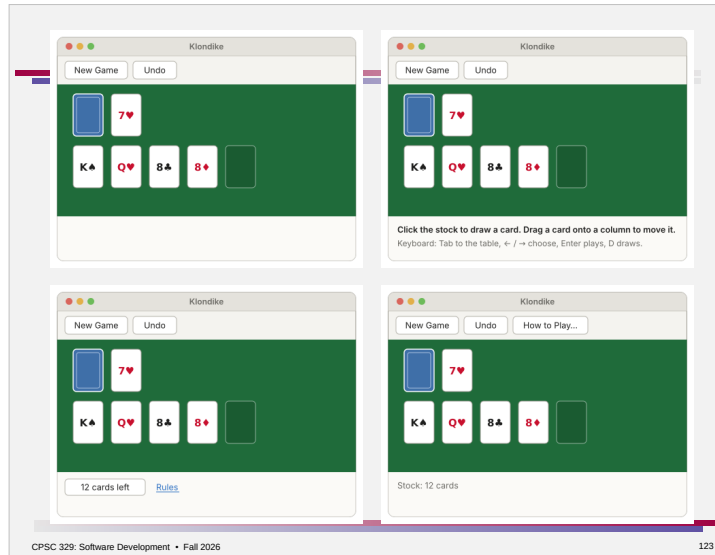


Errors: Prevent First, Then Explain

- prevent
 - constrain input so mistakes can't happen
 - e.g. Spinner, ComboBox instead of TextField
 - disable what doesn't apply
- explain
 - when an error happens, say what went wrong and how to fix it
 - in words (not only with a red border)
 - next to where it happened
 - bad: "invalid input"
 - good: "seed must be a whole number e.g. 12345"
- confirm only destructive or irreversible actions
 - "are you sure?" on everything is annoying and trains people to click Yes without reading
- offer undo where you can
 - makes mistakes cheap
 - allows for fewer confirmations

1





Affordances and Signifiers

- an *affordance* is what a user can do with something
 - e.g. a card can be dragged
- a *signifier* is a visible cue that shows the affordance
 - e.g. a hand cursor, a lift on drag
- signifiers help mouse users discover actions
 - an affordance without a signifier is *hidden* – users won't find it
- avoid *false signifiers* – something that looks like it does something but doesn't
 - e.g. underlined text that isn't a link or a button-shaped label

Affordances and Signifiers

- built-in controls come with signifiers
 - e.g. a button looks pressable, a text field looks typeable
- custom components have none by default
 - e.g. a card drawn from shape doesn't say "drag me"
- adding signifiers – examples

```
// hand cursor: "you can click or drag this"
stock.setCursor(Cursor.HAND);

// hover highlight: shows it responds before you commit
waste.setOnMouseEntered(e -> waste.setHighlighted(true));
waste.setOnMouseExited(e -> waste.setHighlighted(false));

// "lifted" look while dragging
waste.setOpacity(0.4);           // in DRAG_DETECTED; restore in DRAG_DONE

// an instruction line for anything a cue can't show
Label hint = new Label("Click the stock to draw; " +
    "drag the waste card onto a column");
```

Accessibility

- keyboard access: focus
- keyboard shortcuts: mnemonics and accelerators
- screen readers
- color and contrast
- custom components
- keyboard alternative to dragging

Accessibility Is for Everyone

- designing for accessibility includes
 - people who can't see the screen – screen reader compatibility
 - people with low vision – high contrast, large text, magnification
 - people who can't use a mouse – keyboard only
 - people with color vision differences – about 1 in 12 men and 1 in 200 women
- accessibility helps everyone
 - the curb cut idea – ramps built for wheelchairs help people with strollers and suitcases too
 - e.g. keyboard access helps power users
 - e.g. good contrast helps everyone when things are displayed in lower-contrast environments (such as projectors)
- accessibility is a professional expectation
 - designing for inclusion is part of the ACM Code of Ethics
 - laws and regulations often require compliance with W3C's WCAG (Web Content Accessibility Guidelines)
 - e.g. Section 508 for US federal agencies and the ADA (Americans with Disabilities Act) for state and local government websites and applications

7

Keyboard Access: Focus

- one node at a time has *keyboard focus*
 - key presses go to it
- tab / shift-tab move focus through focusable nodes in scene graph order
 - tab moves forward, shift-tab moves backward
 - to adjust –
 - add nodes to the scene graph in the desired order
 - GridPane gotcha – tab follows the order of the add(...) calls, not row and column numbers
 - use containers to group – a container's children are visited together
 - wrap related controls in a VBox or HBox to keep them consecutive
 - skip a node with `setFocusTraversable(false)`
- space presses the focused button, enter presses the default button
- `node.requestFocus()` moves focus programmatically
 - e.g. to a field with an error

8

Keyboard Access: Focus

- controls are focusable and show a *focus ring* when they have focus
 - labels, panes, shapes are/do not
- 
- `setFocusTraversable(true)` to add a node to the tab order
 - needed for custom components
 - also need to add a key handler (so having the focus does something), a visible indicator, and (typically) `requestFocus()` on click
 - usability test: put the mouse away and do every task with the keyboard alone
 - can you do it?
 - is the tab order sensible?

Keyboard Shortcuts

- a *mnemonic* lets you navigate an interface from the keyboard
- an *accelerator* invokes an action directly, skipping the navigation entirely

	mnemonic	accelerator
key combination	Alt + a letter from the label	usually Ctrl/Cmd + a key, e.g. Ctrl+N
shown as	an underlined letter in the label	the shortcut printed next to a menu item
works when	the control is on screen; in menus, only while that menu is open, one level at a time	anywhere in the window, without opening a menu
purpose	reach any visible control without the mouse	a fast path to a common command
set with	an underscore before the letter in the text: " _Draw"	<code>setAccelerator(KeyCombination.keyCombination("Shortcut+N"))</code>
platforms	Windows and Linux convention	all platforms (Shortcut = Ctrl on Windows/Linux or Cmd on Mac)

Keyboard Shortcuts

- buttons, checkboxes, menu items parse mnemonics by default
- labels can give a mnemonic to a field and tell screen readers what a field is for

```
TextField seed = new TextField();
Label seedLabel = new Label("_Seed:");
seedLabel.setMnemonicParsing(true); // labels don't parse by default
seedLabel.setLabelFor(seed);       // Alt+S focuses the field
```

- specifying an accelerator

```
newGame.setAccelerator(KeyCombination.keyCombination("Shortcut+N"));
```

Screen Readers

- a screen reader speaks each focused node – its role (e.g. button, checkbox), its text, and its state (checked, disabled)
- built-in controls provide all three automatically
- images and shapes are silent by default
 - `setAccessibleText(...)` defines the primary label – describes what the element is (read first)

```
dieImage.setAccessibleText("Die showing 4");
cardView.setAccessibleText("7 of hearts");
```
 - `setAccessibleHelp(...)` defines a secondary description – describes how to interact with the element (read second, after a pause)
 - write accessible text the way you'd say it aloud – "7 of hearts", not "7H"
 - describe meaning, not appearance – "column 3, selected" rather than "blue rectangle"
- tooltips don't count – keyboard users and screen readers usually never see them

Color and Contrast

- contrast – text needs enough contrast with its background
 - WCAG 2.0 AA – at least 4.5:1 for normal text, 3:1 for large text and for the edges of controls and important graphics
 - check with a contrast checker – don't trust your eyes or your own monitor
 - e.g. <https://accessibleweb.com/color-contrast-checker/>
- never use color alone – pair every color cue with a second one
 - e.g. an error – red border + a message in words
 - e.g. card suits – red and black + the suit symbol
 - e.g. players – a color + a letter or name
 - e.g. drop ring – color + a thick outline
 - quick test – take a screenshot and convert to grayscale
 - anything ambiguous in the grayscale version is relying on color alone
 - check with a color blindness simulator
 - e.g. <https://daltonlens.org/colorblindness-simulator>

Custom Components

- shapes and panes don't get any of the built-ins
 - not focusable, so tab skips them
 - no keyboard handling
 - nothing for a screen reader to say
 - no focus ring
- for a game board, create one focusable board with a keyboard cursor
 - one tab stop for the board
 - if the board is made up of many components, making everyone focusable becomes tedious
 - use arrow keys to move a visible cursor
 - Enter acts on the cursor's position
 - the board's accessible text describes where the cursor is
 - showing the cursor only while the board has focus needs a focus listener – a future topic

Dragging With the Keyboard

- every drag needs a non-drag way to do the same thing
 - e.g. for Klondike, tab to shift focus to the board and left/right arrows to choose a tableau column, then Enter to play the waste card there and D to draw from the stock

```
tableauLayer_.setFocusTraversable(true);
// show the cursor ring whenever the board gains or loses focus
tableauLayer_.focusedProperty().addListener((obs, wasFocused, isFocused) →
    showCursor());
tableauLayer_.setOnMouseClicked(e → tableauLayer_.requestFocus());
tableauLayer_.setOnKeyPressed(e → {
    if (e.getCode() == KeyCode.LEFT) {
        if (cursor_ > 0) { cursor_--; }
    } else if (e.getCode() == KeyCode.RIGHT) {
        if (cursor_ < MiniGame.COLUMNS - 1) { cursor_++; }
    } else if (e.getCode() == KeyCode.ENTER) { playWasteTo(cursor_); }
    } else if (e.getCode() == KeyCode.D) { drawFromStock(); }
    } else { return; } // leave other keys alone so tab still works
    showCursor();
    e.consume();
});
```

- setOnMouseClicked to request focus serves mixed mouse/keyboard users – clicking doesn't move focus to a custom node by default
- also add an instruction label so users know the keys exist

15

UI Checklist

- *clear* – the main action stands out; related things are grouped and aligned
- *consistent* – one word per concept; follows platform conventions; Enter and Esc work
- *responsive* – every action gets visible feedback; state is shown, not remembered; nothing fails silently
- *forgiving* – invalid input prevented where possible; errors explained in words next to the problem; destructive actions confirmed
 - undo if possible
- *discoverable* – clickable and draggable things look it; a short instruction where needed
- *keyboard* – every task works without a mouse; sensible tab order; mnemonics or accelerators; no keyboard traps
- *screen reader* – fields have labels (setLabelFor); images and shapes have accessible text
- *color* – enough contrast; nothing relies on color alone

16