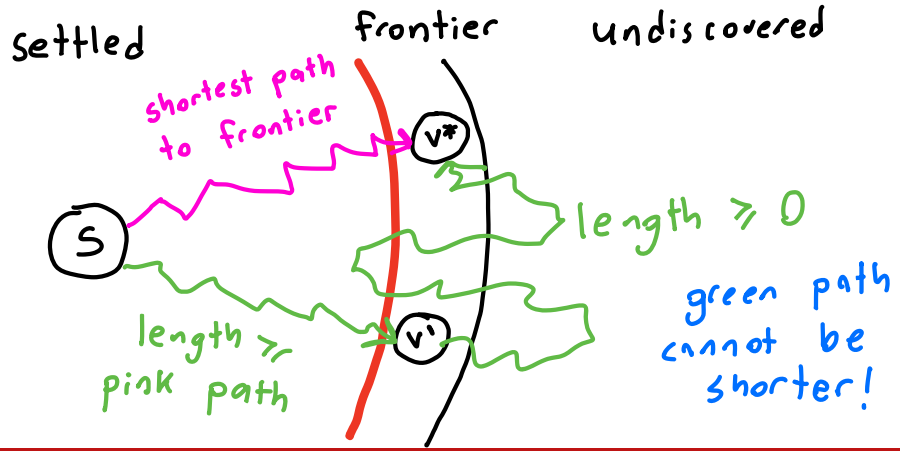
Dijkstra's Invariant

At the start of each main loop iteration:

1. We've located the shortest path to all *settled* vertices.
2. The shortest *known* path to the *closest* `frontier` vertex v^* is guaranteed to be the true shortest $s \rightsquigarrow v^*$ path.

Proof Sketch:

- consider any alternate path to v^*
- at some point, this path must cross boundary to some frontier vertex v'



Properties of Dijkstra's Algorithm

Vertices are visited/settled in increasing *distance* order (just like BFS).

- Shortest path to any unsettled vertex passes through frontier, so has distance ≥ any settled vertex.

The best *known* distance to a vertex v can only decrease as the algorithm runs (and we locate new paths to v)

For any discovered but unsettled vertex v,

≤ length of shortest s⇝v path ≤

Dijkstra's Algorithm Runtime Analysis

```
while(!frontier.isEmpty()) {
  V v = frontier.remove();
  for (WeightedEdge<V> edge : v.outgoingEdges()) {
    V neighbor = edge.head();
    double dist = discovered.get(v).distance() + edge.weight();
    if (!discovered.containsKey(neighbor)) {
      discovered.put(neighbor, new PathInfo<>(dist, v));
      frontier.add(neighbor, dist);
    } else if (discovered.get(neighbor).distance() > dist)  {
      discovered.put(neighbor, new PathInfo<>(dist, v));
      frontier.updatePriority(neighbor, dist);
}}}
```



Lecture 24: Graphical User Interfaces

CS 2110

November 17, 2026

Today's Learning Outcomes

86. Categorize aspects of a GUI application as part of its *model*, *view*, or *controller*.

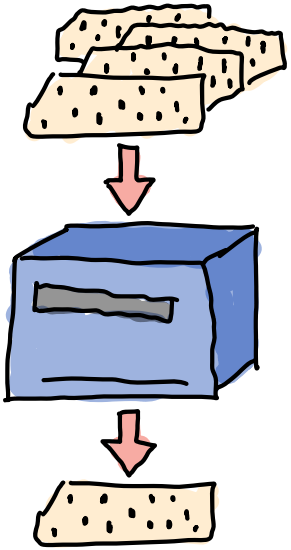
87. Write code to set up the component hierarchy of a graphical application given a mock-up of its design.

88. Describe the work performed by a layout manager and its interaction with elements in the component hierarchy.

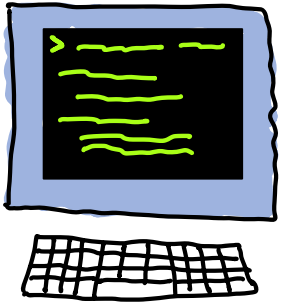
90. Write code that performs custom painting on graphical components.

Application Paradigms

Batch Applications



Command-Line Interfaces (CLI)



Enter Guess: 1122

○ ○ ● ●

A1

Enter Guess: 3311

● ○ ● ●

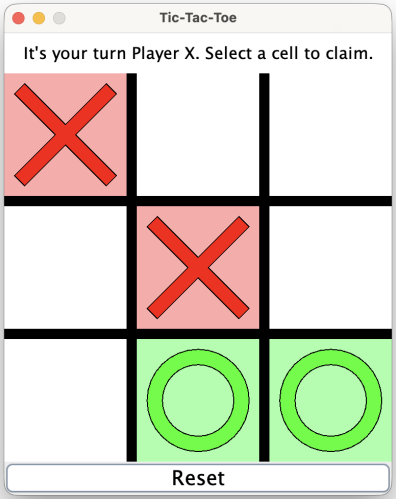
Graphical User Interfaces (GUI)



A6



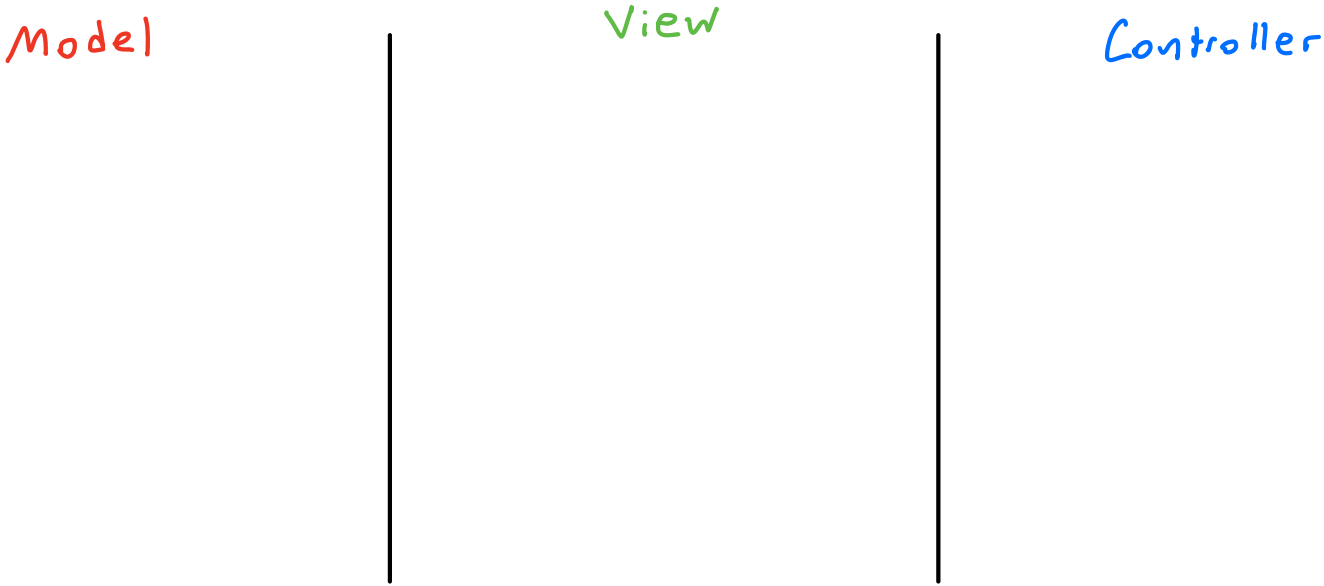
GUI Applications



Brainstorm: What new considerations are there to make a graphical application like this?

MVC Separation

Since GUI applications are more involved, it helps to organize their code into three different aspects:





Coding Demo: TicTacToe Model



GUI Frameworks

GUI applications are way more complicated than other programs we've written.

- Interactions with hardware (keyboard, mouse, etc.) and the operating system.
- Good views incorporate many interoperable and customizable widgets.
- Many layers of abstraction.
- High *coupling* between parts of the view and controller.

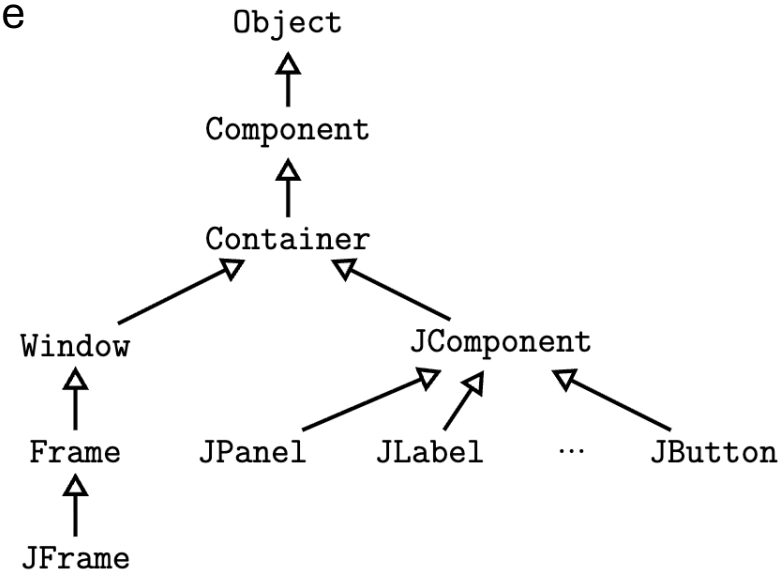


Coding Demo: The `JFrame` Class



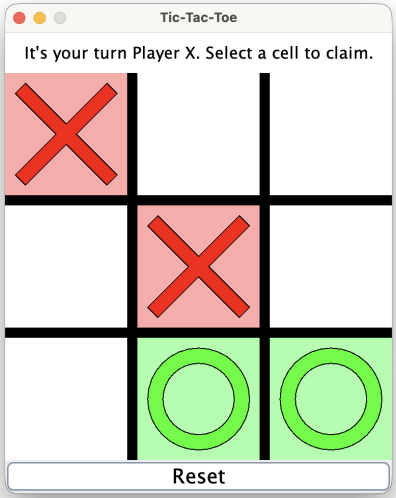
Swing Widgets

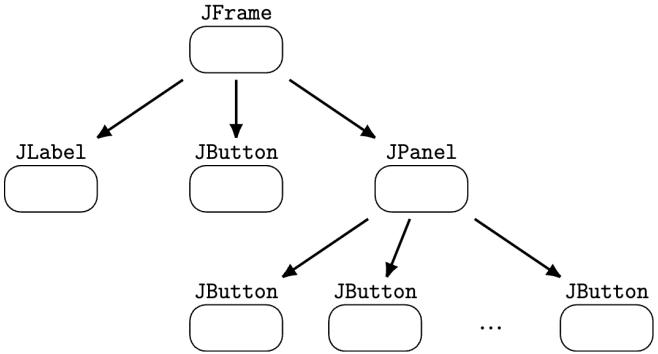
The `JComponent` class serves as the parent of all Swing widgets.



The Component Hierarchy

= A tree that describes all the widgets in a GUI application window.
Parent-child relationships in this tree represent *containment*.

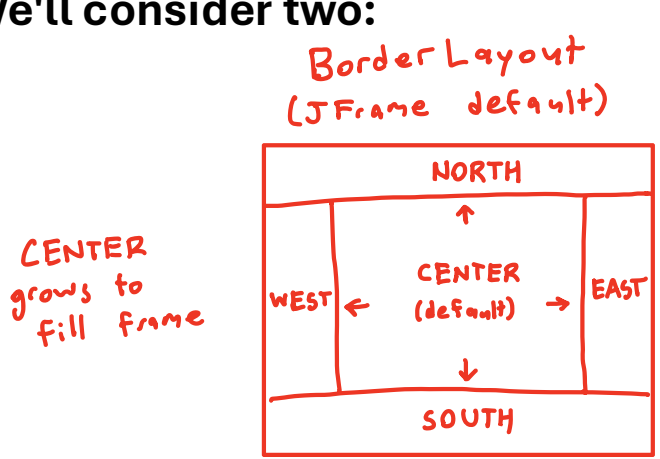


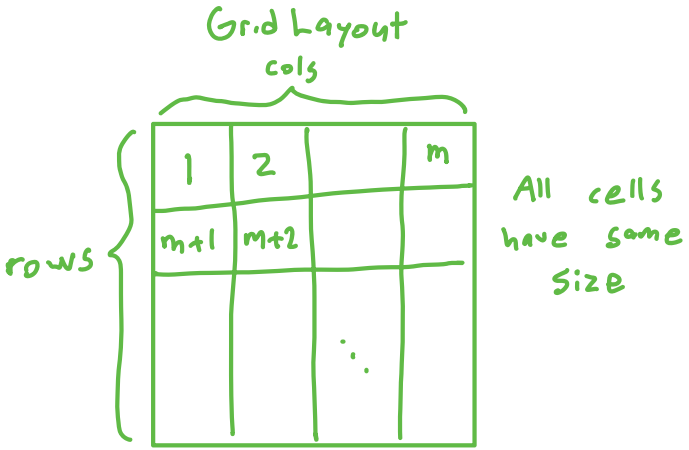


Layout Managers

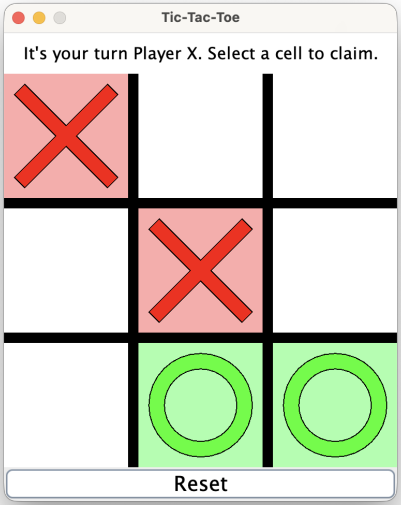
= Separate objects associated with a `Container` that determine how to arrange its child components.

We'll consider two:





Coding Demo: Building out the View



Key Ideas:

- Custom `JComponent` subclasses
- Set `LayoutManagers` and use their constants to assign positioning in `add()`
- Call `pack()` at the end of the frame constructor

Connecting the View and Model

Multiple Approaches:

Maintain the model in a separate class (or classes), accessible to all view components through alias referencing.

Store relevant bits of the model within the relevant view classes.

* we'll demonstrate a bit of both

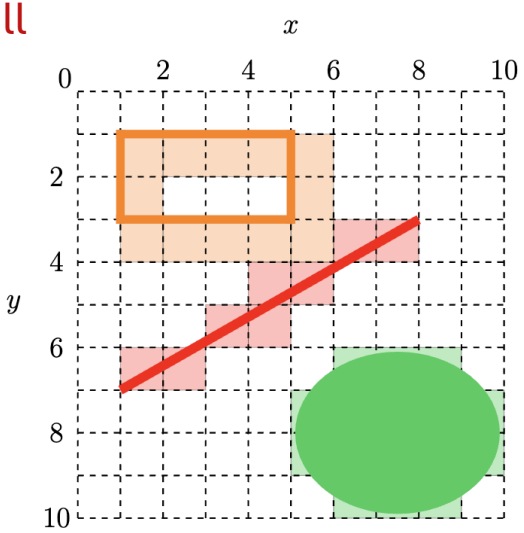
Custom Painting

Override the `paintComponent()` method.

* First line must be `super.paintComponent()` call

`Graphics` object works like a paint brush:

- `setColor()` "dips" into one color of paint
- `draw____()` and `fill___()` methods add lines/shapes in that color



* Never call `paintComponent()` directly, instead call `repaint()`



Coding Demo: Painting Demo

