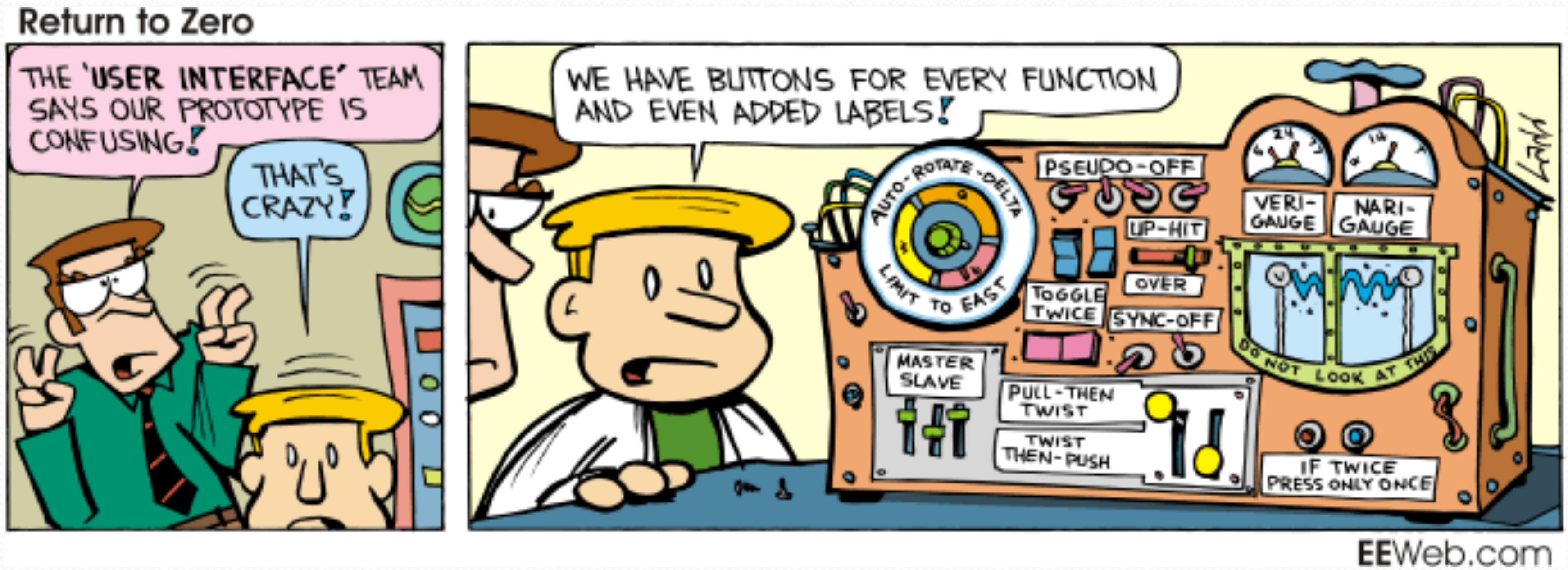
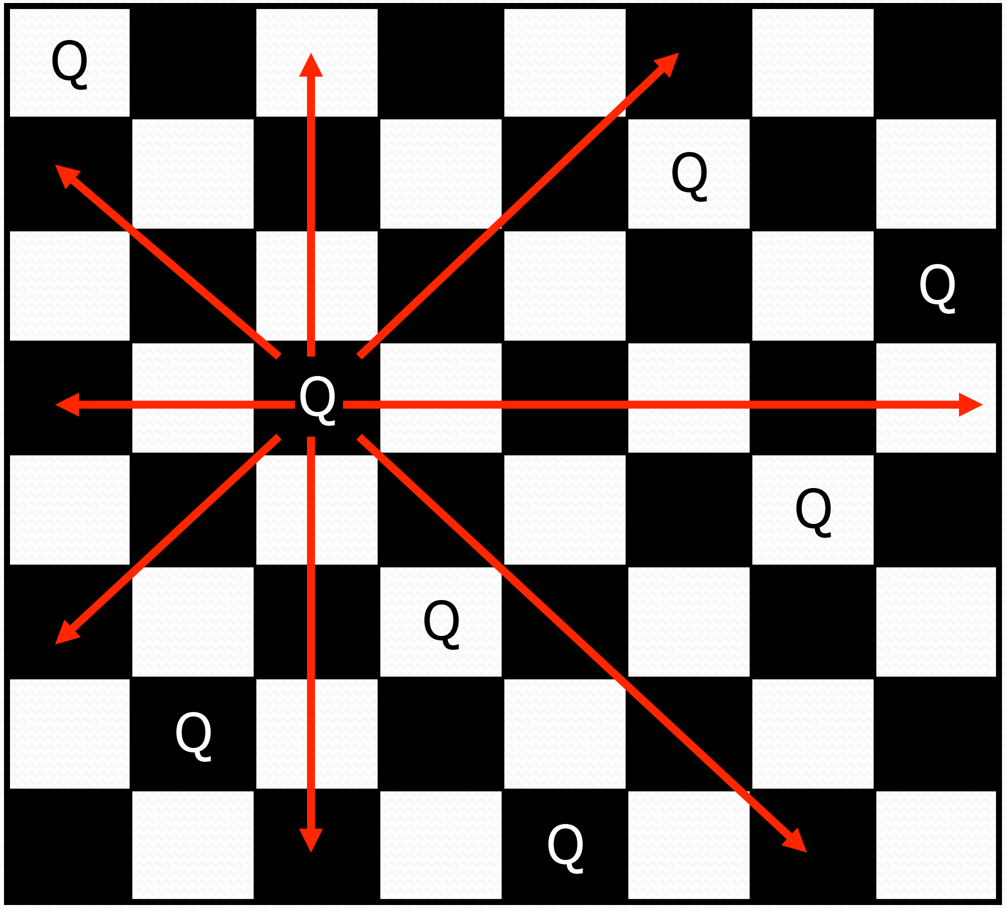

Return to Zero
THE 'USER INTERFACE' TEAM SAYS OUR PROTOTYPE IS CONFUSING!
THAT'S CRAZY!
WE HAVE BUTTONS FOR EVERY FUNCTION AND EVEN ADDED LABELS!
AUTO-ROTATE-DELTA
LIMIT TO EAST
PSEUDO-OFF
UP-HIT
OVER
TOGGLE TWICE
SYNC-OFF
VERI-GAUGE
NARI-GAUGE
DO NOT LOOK AT THIS
MASTER SLAVE
PULL-THEN TWIST
TWIST THEN-PUSH
IF TWICE PRESS ONLY ONCE
EEWeb.com

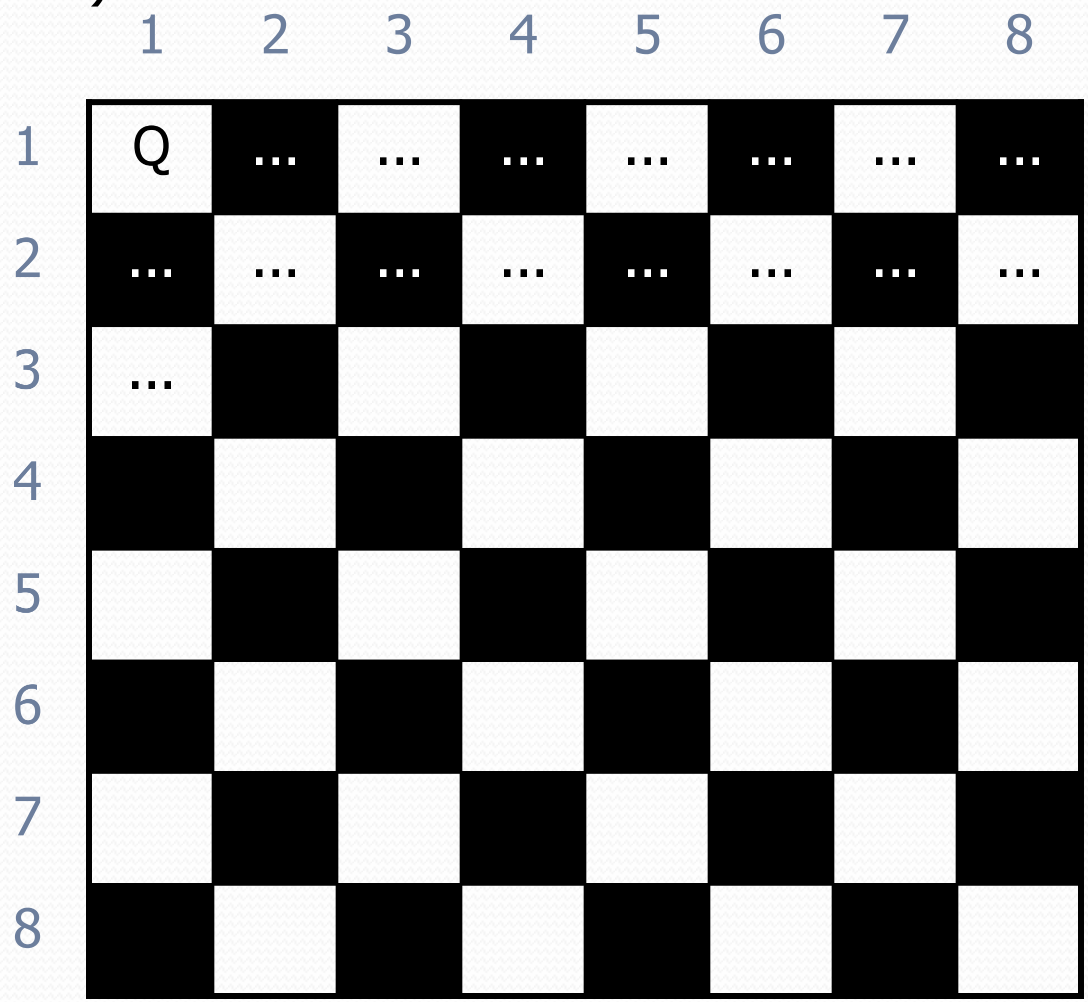
The "8 Queens" problem

- Consider the problem of trying to place 8 queens on a chess board such that no queen can attack another queen.
  - What are the "choices"?
  - How do we "make" or "un-make" a choice?
  - How do we know when to stop?



Naive algorithm

- for (each square on board):
  - Place a queen there.
  - Try to place the rest of the queens.
  - Un-place the queen.
  - How large is the solution space for this algorithm?
    - 64 * 63 * 62 * ...



Better algorithm idea

- Observation: In a working solution, exactly 1 queen must appear in each row and in each column.
  - Redefine a "choice" to be valid placement of a queen in a particular column.
  - How large is the solution space now?
    - 8 * 8 * 8 * ...

	1	2	3	4	5	6	7	8
1	Q	...	...					
2		...	...					
3		Q	...					
4			...					
5			Q					
6								
7								
8								

Recall: Backtracking

A general pseudo-code algorithm for backtracking problems:

Explore(**choices**):

- if there are no more **choices** to make: stop.
- else, for each available choice **C**:
  - Choose **C**.
  - Explore the remaining **choices**.
  - Un-choose **C**, if necessary. (backtrack!)

Exercise

- Suppose we have a `Board` class with these methods:

Method/Constructor	Description
`public` **`Board`**`(int size)`	construct empty board
`public boolean` **`isSafe`**`(int row, int column)`	`true` if queen can be safely placed here
`public void` **`place`**`(int row, int column)`	place queen here
`public void` **`remove`**`(int row, int column)`	remove queen from here
`public String` **`toString`**`()`	text display of board

- Write a method `solveQueens` that accepts a `Board` as a parameter and tries to place 8 queens on it safely.
  - Your method should stop exploring if it finds a solution.

Exercise solution

```
// Searches for a solution to the 8 queens problem
// with this board, reporting the first result found.
public static void solveQueens(Board board) {
    if (solveQueens(board, 1)) {
        System.out.println("One solution is as follows:");
        System.out.println(board);
    } else {
        System.out.println("No solution found.");
    }
}

...
```

Exercise solution, cont'd.

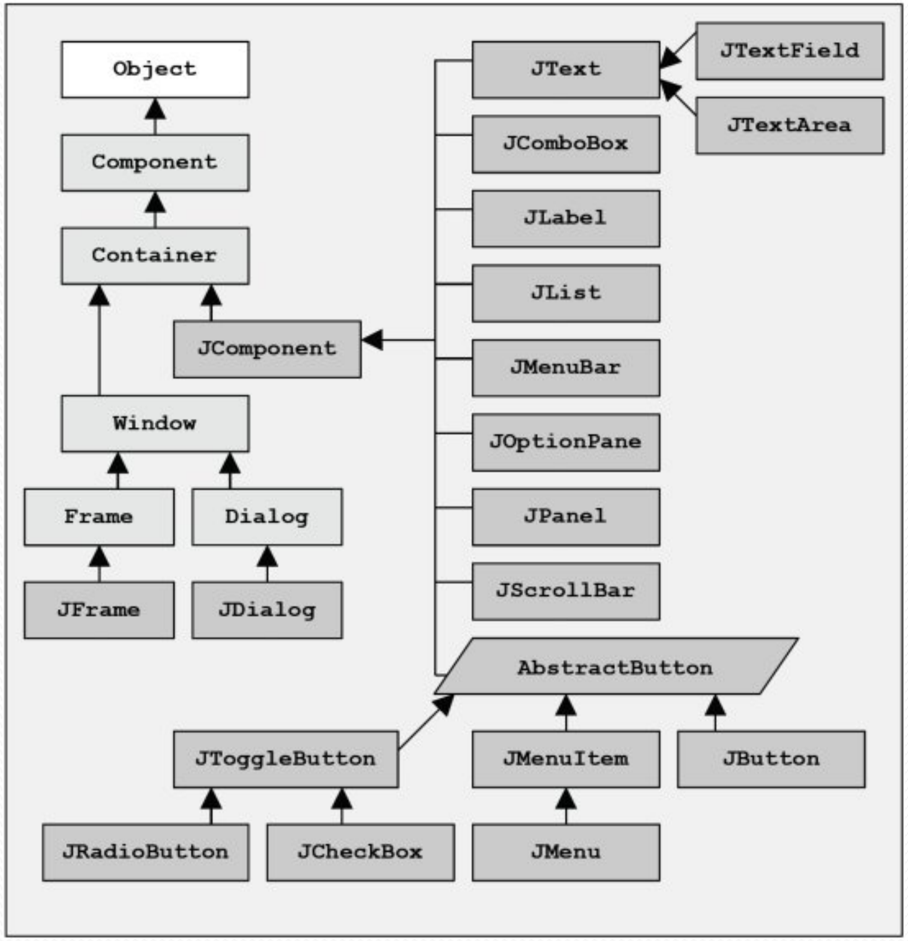
```
// Recursively searches for a solution to 8 queens on this
// board, starting with the given column, returning true if a
// solution is found and storing that solution in the board.
// PRE: queens have been safely placed in columns 1 to (col-1)
public static boolean solveQueens(Board board, int col) {
    if (col > board.size()) {
        return true;   // base case: all columns are placed
    } else {
        // recursive case: place a queen in this column
        for (int row = 1; row <= board.size(); row++) {
            if (board.isSafe(row, col)) {
                board.place(row, col);              // choose
                if (explore(board, col + 1)) {    // explore
                    return true;   // solution found
                }
                b.remove(row, col);                 // un-choose
            }
        }
        return false;   // no solution found
    }
}
```

Graphical User Interfaces

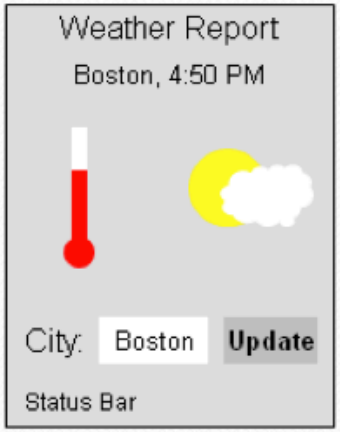
- Involve large numbers of interacting objects and classes
  - Highly framework-dependent
- Path of code execution unknown
  - Users can interact with widgets in any order
  - Event-driven
- In Java, AWT vs. Swing; GUI builders vs. writing by hand

Swing Framework

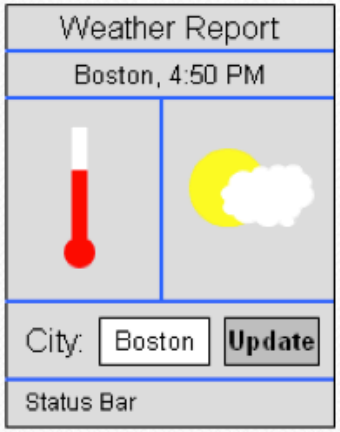
- Great case study in OO design



Composite Layout



Draw out desired result



Divide into regions

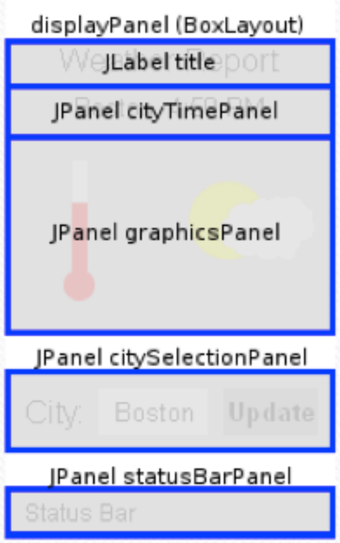


Figure out appropriate layout managers and components