GUIs and Event-Driven Programs

Final project proposal due tomorrow at midnight.

Physical simulation project due Wednesday at midnight.

You are not required to write more than an quick outline of your project. However:

1. Writing things down in detail acts as a focusing lens for your thoughts.
2. If you write down some implementation ideas, we can evaluate them and provide feedback.

Is there a time limit for the final presentation?

This will depend on group sizes in the class. We might be looking at anything from a very strict 5 minute time limit to a setting where almost nobody needs to watch time very closely.

What info can go in the slides if we're doing a demo?

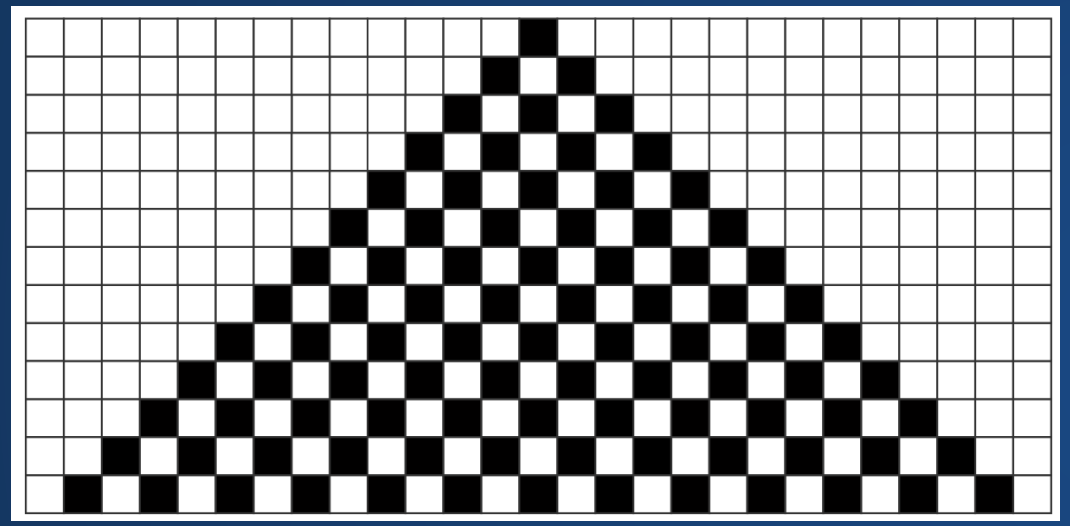
Explain what you did! There are tons of small decisions and other things to explain to someone who's never looked at your project before.

For reference, I've seen engaging, interesting 7 minute presentations on a 2D "throw a ball" simulator.

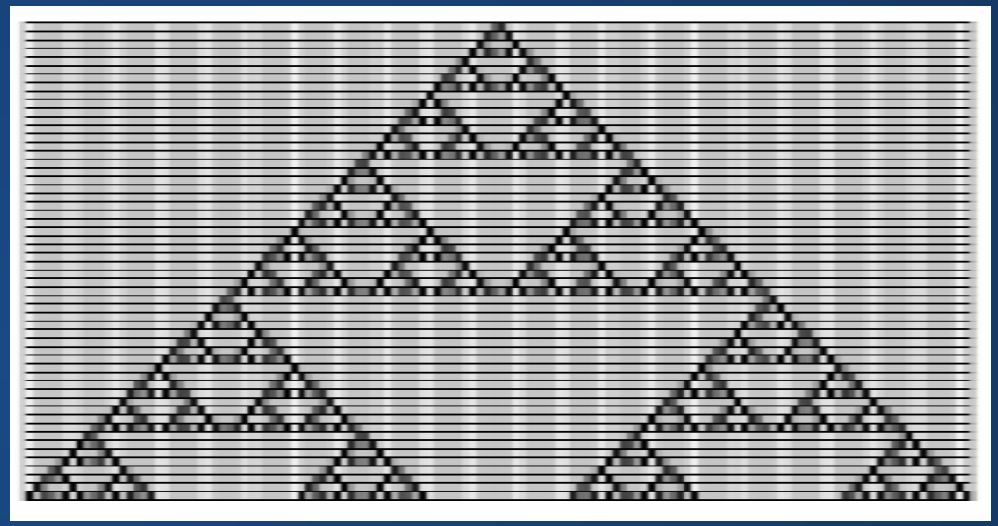
Also, look on Canvas for other requirements.

What other cellular automata exist?

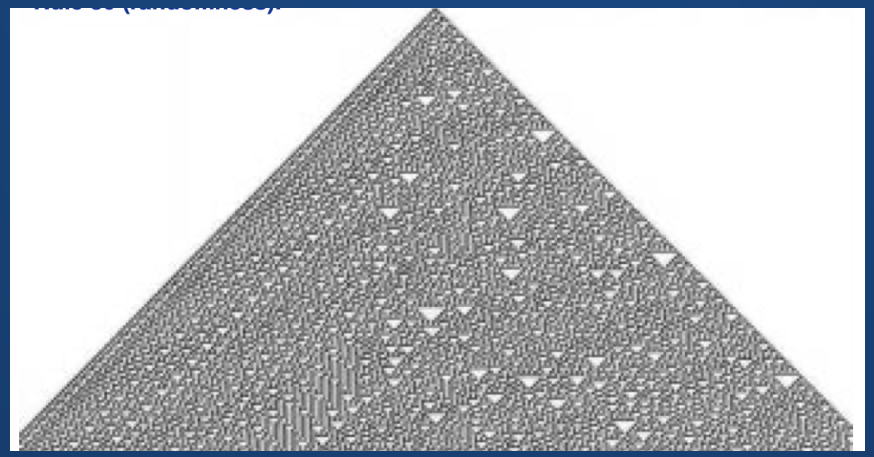
In 1D, there are exactly 256 possible CAs that use neighbor information. Stephen Wolfram has catalogued them in his book, "A New Kind of Science."



Rule 250



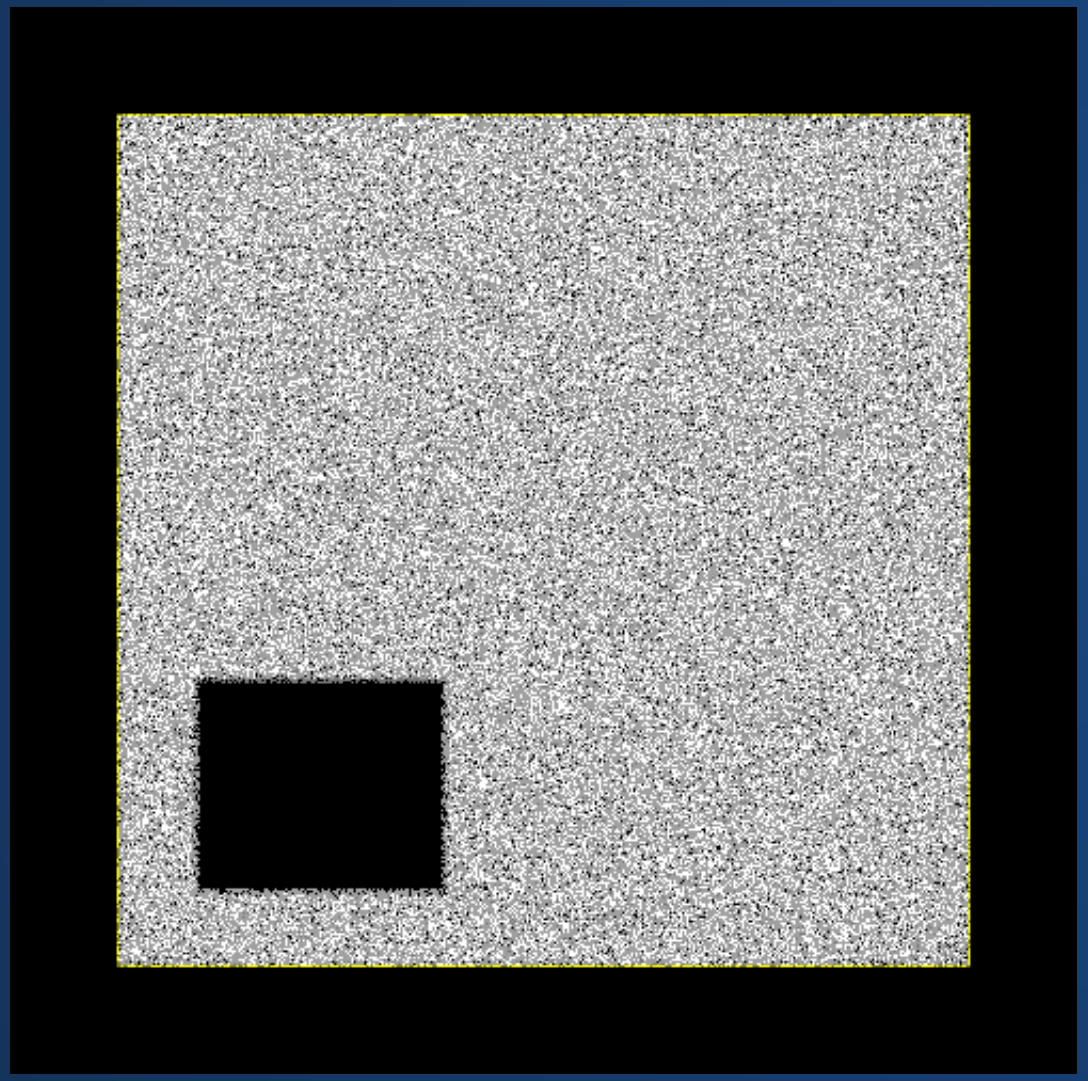
Rule 90

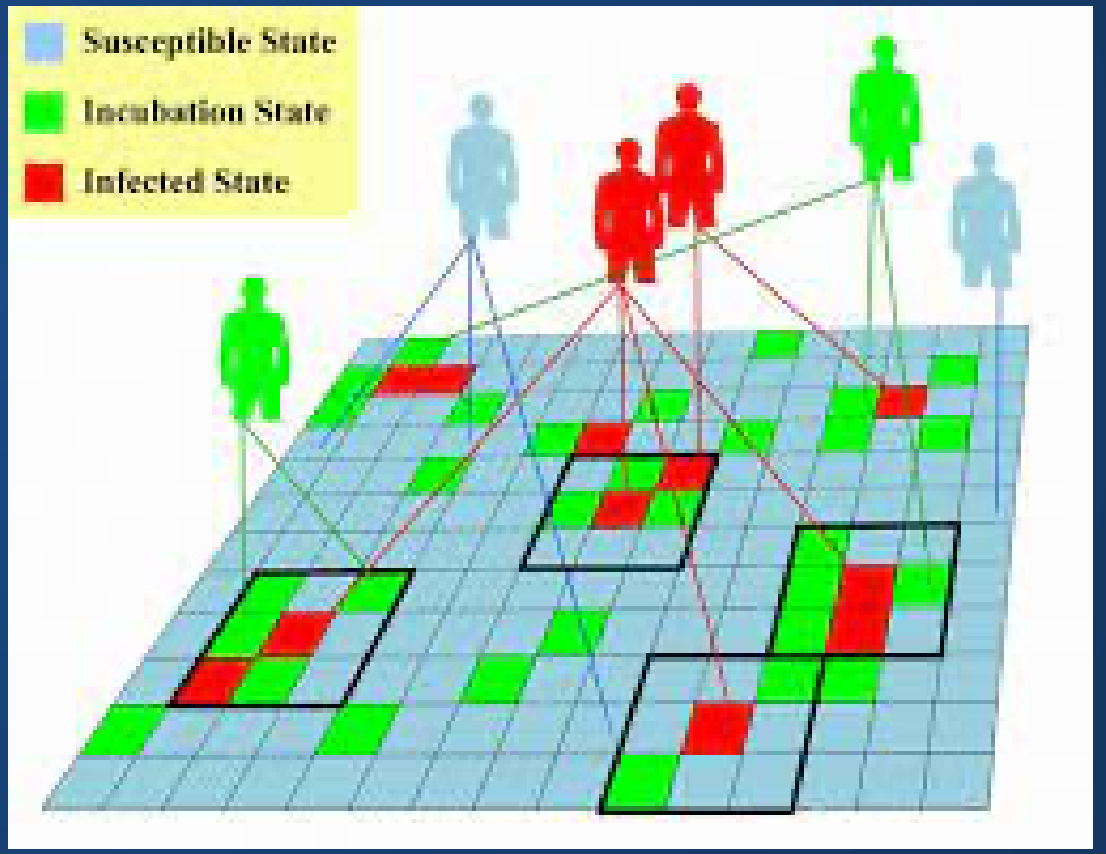


Rule 30

In 2D, we go from 256 possible automata to 1.3×10^{154}.

Very hard to categorize all of these, but many interesting behaviors.





Is "The Sims" just a giant Game of Life?

No, but....

What is the connection between Cellular Automata and AIs?

Automata (not cellular in nature) power most rule-based systems.

AIs in games tend to be one of two systems, both discrete automata:

- Behavior trees
- Finite state machines

https://www.youtube.com/embed/6VBCXvfNlCM?enablejsapi=1

For more general AIs, before the neural network + backpropogation paradigm was shown to be effective (ca. 2010), the primary AI tools were automata-based.

Can CAs display fractals?

Not with infinite resolution, but yes.

In fact, some of the earliest graphics programs were generated by slightly more general CAs!

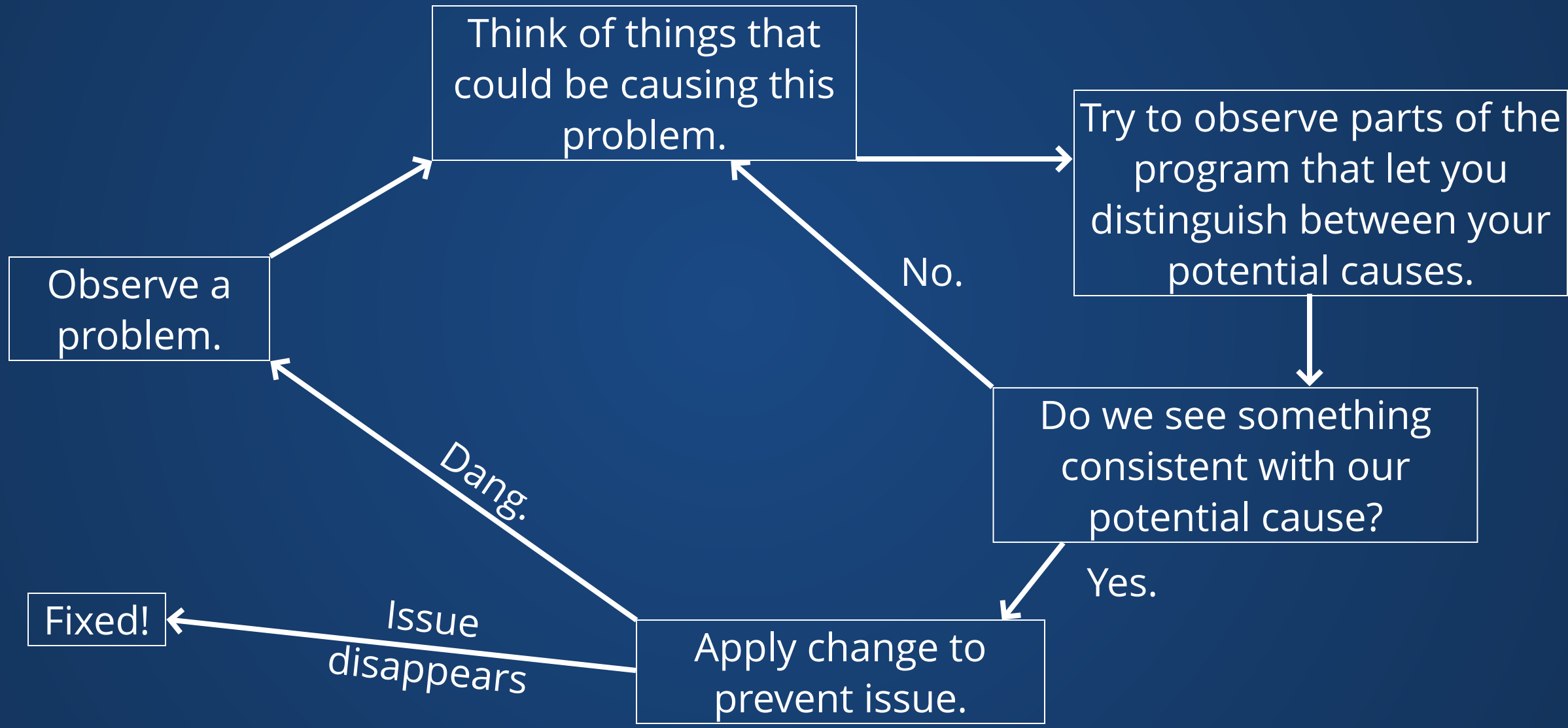
Debugging is Hard

Yes.

Debugging broken code is **significantly** harder than writing new code. I have **never** heard a professional developer express an opinion contrary to this.

It's even harder in graphics because certain parts of the "regular" debugging cycle are much harder or even impossible to do, so debugging broken graphics code can become very interesting.

Debugging


Think of things that could be causing this problem.
Try to observe parts of the program that let you distinguish between your potential causes.
Observe a problem.
No.
Do we see something consistent with our potential cause?
Dang.
Yes.
Fixed!
Issue disappears
Apply change to prevent issue.

How not to Debug*

1. See a problem.
2. "I wonder if it might be X."
3. Write a bunch of code that would fix X.
4. It doesn't work.
5. "I wonder if it might be Y..."

Do this too many times, and your code is undebuggable!

* You can do this a few times as a sort of quick-and-dirty debugging path. If it doesn't work 2 or 3 times, then STOP! Back up, take a breath, and take it slow and steady.

Example

My Game of Life was not showing any change in squares.

Other Tricks

```python
def calculate(lst):
    total_i = 0
    total_e = 0
    for t in lst:
        if t['t'] == 'i':
            total_i += t['a']
        else:
            total_e -= t['a']
    return total_i - total_e
```

```python
def total_income(transactions):
    income = 0
    for t in transactions:
        if t['type'] == 'income':
            income += t['amount']
    return income

def total_expenses(transactions):
    expenses = 0
    for t in transactions:
        if t['type'] == 'expense':
            expenses -= t['amount']
    return expenses

def calculate_net(transactions):
    income = total_income(transactions)
    expenses = total_expenses(transactions)
    income - expenses
```

Other Tricks

Good naming and commenting in your functions can save you a lot of grief!

```
1  void main(){
2    // ...
3
4    for(int i = 0; i < 100; i++){
5      for(int j = 0; j < 100; j++){
6        array[i * width + j] = array[i+1 * width + j] + 10;
7      }
8    }
9
10   // ...
11 }
```

Structuring code into functions can allow you to test things without suffering printspam.

```
1  void main(){
2    // ...
3
4    for(int i = 0; i < 100; i++){
5      for(int j = 0; j < 100; j++){
6        float val = func(array, i+1, j);
7        array[i * width + j] = val;
8      }
9    }
10 }
```

Other Tricks

When writing your code, do it as simply as you can.

If debugging is harder than writing, and you're using 100% of your cleverness when writing the code, how are you ever going to debug it when things inevitably go wrong?

NOTE: When you're still learning to program (as you are now), there is a tendency to favor cool tricks over boring code. This is **fine** (I would even argue that it is **good**) but remember that this does make it harder to debug later.

But mostly...

It takes a lot of time and experience.

The Challenges of User-Facing Programs

"Typical" Program

- Do computation
- **When results are ready**, show them to the user
- User may optionally enter more input
- Program processes user input **when it is ready**, then carries out more computation

Program does things on its schedule!

User-Facing Programs

- Program usually sits around and does nothing until it needs to respond to something.
- Once external event occurs, program should respond to it as quickly as possible.
- Might need to process multiple events quickly

Typical Program

Interactive Program

Events

Events

The things we need to respond to in our program are termed "events."

Events can be system-generated or user-generated, but are almost always external to the program---we can't control when they happen.

When events occur, we execute code which alters the program state. This is termed "event-driven programming."

Taken to the extreme, this creates a paradigm known as "reactive programming."

Examples of Events

System-Generated

- Hardware timer fires
- File load completes
- Network message arrives

User-Generated

- Keyboard press
- Mouse movement/click
- Shutdown initiated

Toolkits

Unless you plan to write your own user interfaces from scratch (not recommended!), you'll likely be programming interfaces using an existing toolkit.

These toolkits already know how to recognize most common events, like mouse click, mouse drag, window close, etc.

The programmer is responsible for writing functions that will be called when these events occur, and telling the toolkit which functions to call on which events.

Callbacks

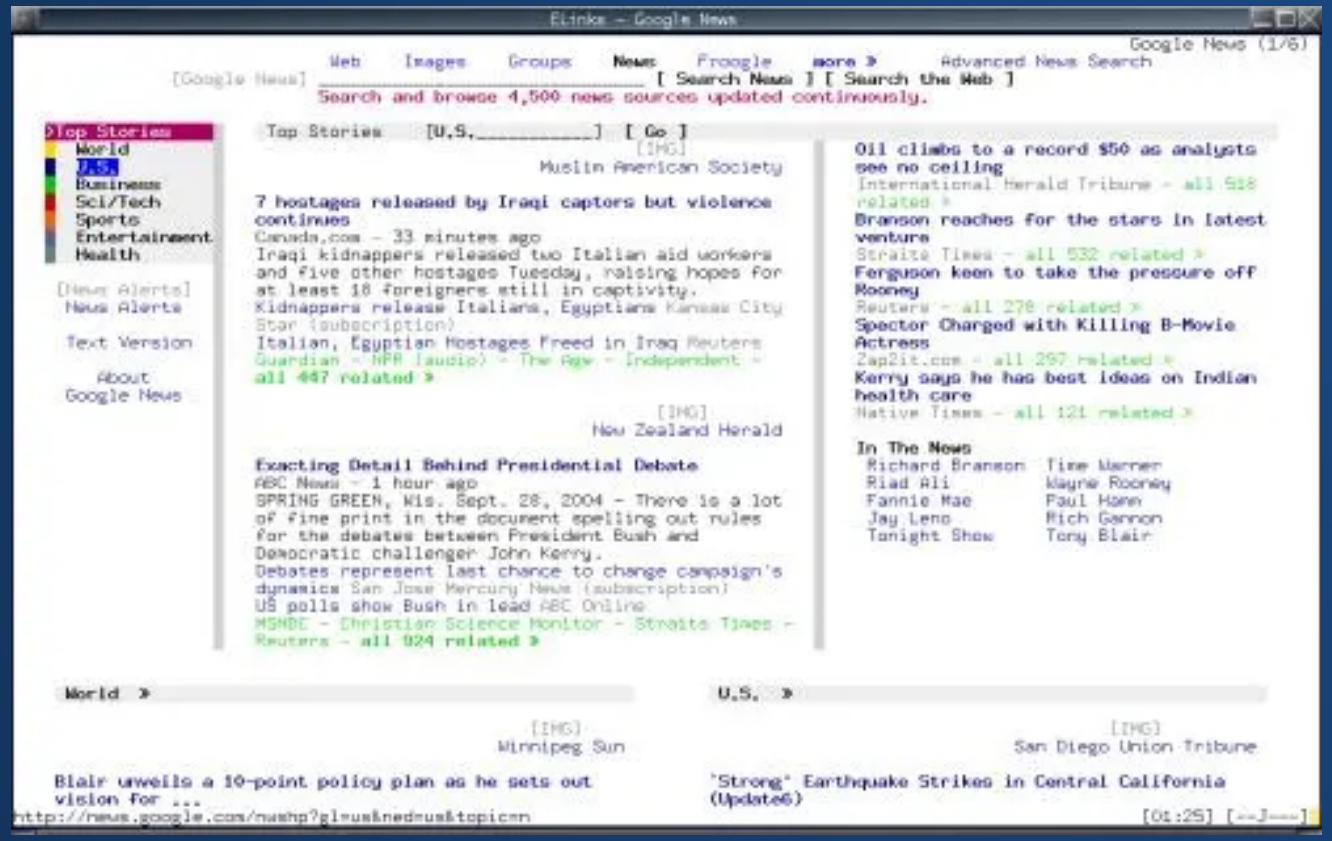
A common programming technique: we register a function to be called when something happens. Example in fake-Processing:

```
1  void mouseClick(){
2    playPauseSound();
3    displayMenu();
4    pauseGame();
5  }
6
7  // Without being told to, the program does
8  // not know it needs to run mouseClick() on
9  // mouse click, so we tell it this in main()
10 void main(){
11   GUI.registerHandler(ON_CLICK, mouseClick);
12 }
```

GUIs

Short for "Graphical User Interface". An interface with a visual component.

Designed for easier, more intuitive experience. Typically based on event-driven programming.



Did you know you can browse the web using only the keyboard? To the left is the eLinks browser.

I'm glad I have Firefox.

Uses

I mean, honestly. What programs don't have graphical interfaces these days?

Question: what are common interactable elements of GUIs?

Widgets

A general name given to interactable elements within a GUI, including things like:

- Buttons
- Checkboxes
- Sliders
- Radio buttons

Give us different ways of interacting with program behavior and configuring the program.

Buttons

Simplest form of widget. Allows for functionality on mouse click.

Buttons need to be aware of the mouse position and the button boundary.

- Circles check whether mouse is within button based on distance from center.
- Rectangles check based on width/height from corner or center.

Hands-On: Buttons

1. Implement a `Button` class. This class should have a method that takes screen coordinates (as separate x, y parameters) and checks whether the coordinates are within the button or not.
2. Create both rectangular and circular buttons, and place them in nonoverlapping positions on the screen.
3. Add functionality so that the color of a button changes when it is clicked.

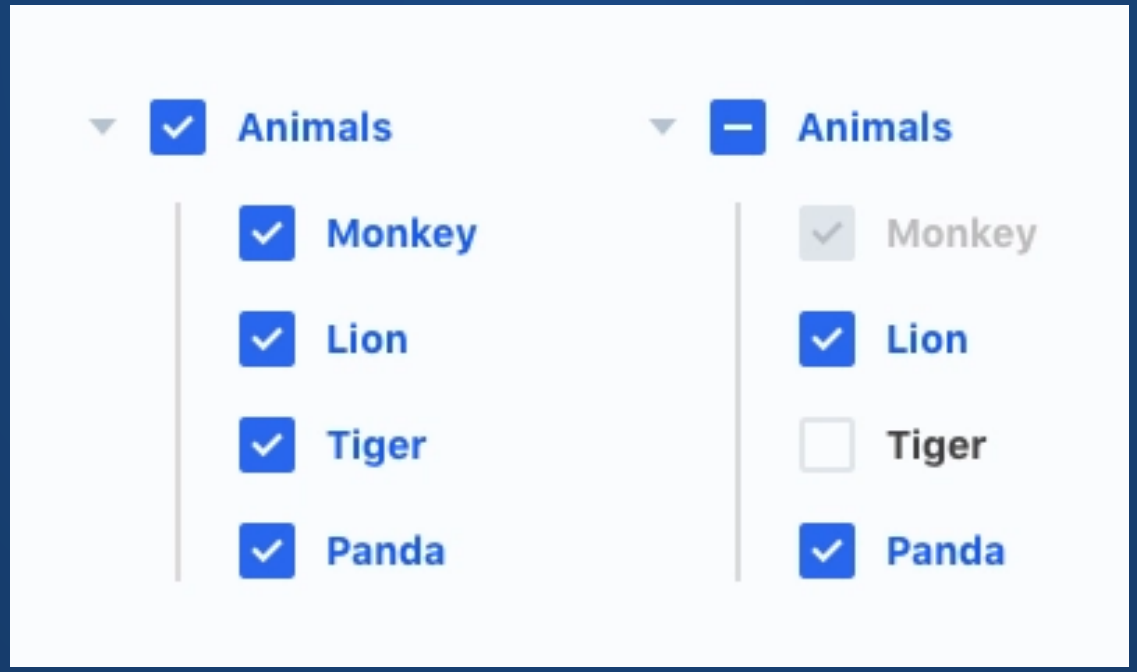
You will extend the Button for today's second Hands-On.

Build Buttons

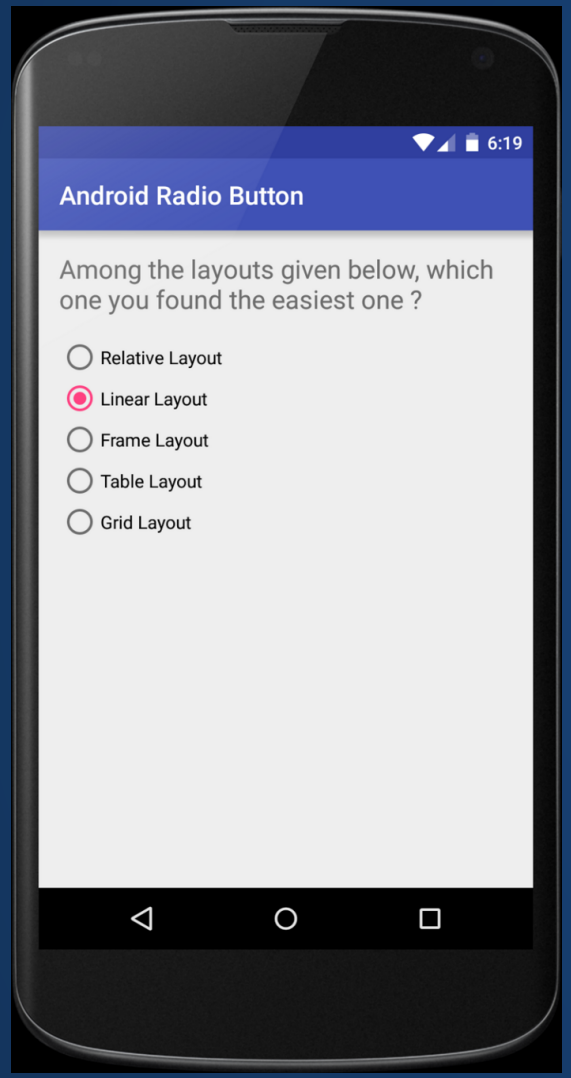
Check Boxes

A specialized button with an "on" and "off" state.

What do we need to store to track a checkbox?



Radio Buttons



A specialized version of checkboxes. Only one radio button can be on at any given time---when one is set to on, all others must be set to off.

Questions

- What other data do radio buttons have to be aware of?
- What data structures can help us organize the radio buttons?

Scrollbars

- Users can select based on a range of values
- Minimum and maximum values correspond to the ends of the slider
- Thumb, or current position, controls the assigned value
- Allows for a “continuous” range of values

What value does this correspond to on a slider?

Does this seem like something else we've studied in this class?

How about this one?

How about this one?

```
0.0 * SLIDER_MAX
```

Slider's value is determined by a *linear interpolation* between the end position of the slider.

```
0.75 * SLIDER_MAX
```

This also works for non-linear sliders (like knobs) with some modification.

```
1.0 * SLIDER_MAX
```

Hands-On: More Widgets

Pick two of the following to implement:

1. Extend your button class to create checkboxes
2. Extend your button class to create radio buttons
3. Create a scrollbar that either updates the x/y position along an image or updates the background color of a sketch.

Make sure you demo them in your sketch.

Other GUI Libraries

Existing libraries for Processing can simplify the GUI creation process.

- Guido: framework for GUI component creation
- controlP5: provides GUI components
- G4P: provides GUI components and builder

Sketches -> Import Library -> Add Library -> [name]

Index Cards!

1. Your name and EID.

2. One thing that you learned from class today. You are allowed to say "nothing" if you didn't learn anything.

3. One question you have about something covered in class today. You *may not* respond "nothing".

4. (Optional) Any other comments/questions/thoughts about today's class.