

Buttons & Boards

CSE 120 Winter 2020

Instructor:

Sam Wolfson

Teaching Assistants:

Yae Kubota

Eunia Lee

Erika Wolfe

MIT Wizards Invent Tech That Sees Around Corners

Let's pretend you're standing in an L-shaped hallway, looking where the inside corner of the hallway meets the floor. You can't see what's around the corner, but you can see light emitting from the other side, splashed onto the floor at that right angle. So long as what's over there isn't a single light source, like a flashlight, you won't see one hard line of shadow. You'll see a sort of gradient of not-quite-shadow—kind of a blurry shadow. This is known as the penumbra.

Your eyes can't see it, but there's a lot going on in this penumbra: It's a reflection—a real-time, low-res view of the scene around the corner. This happens outdoors, too, thanks to light from the sun. Train a camera on this spot and magnify the color, and you can start to pick out different-colored pixels that correspond to objects otherwise obscured by the wall.

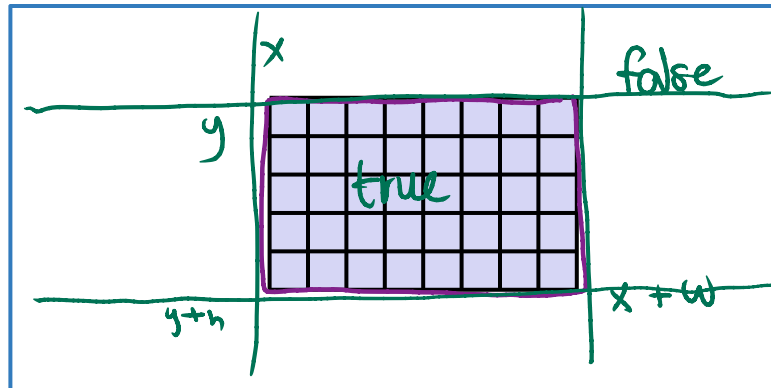
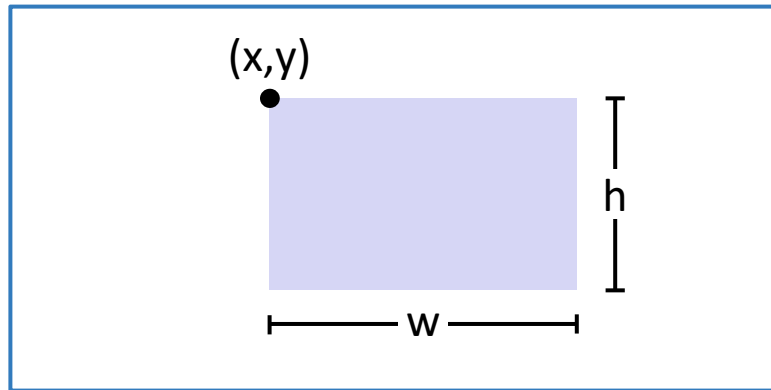
- <https://www.wired.com/story/mit-wizards-invent-tech-that-sees-around-corners/>

Administrivia

- ❖ Assignments (**changes!!!**)
 - Reading Check 6 due @ 3:30 pm tomorrow (2/13)
 - Color Filters [checkoff] now due tomorrow (2/13)
 - We'll help you out in section tomorrow
 - Arrays & Elli [checkoff] due Friday (2/14)

- ❖ “Big Ideas” lecture: Machine Learning

Review: Rectangle Detection



```
if( (mouseX >= x)      &&  
    (mouseX <= x + w)  &&  
    (mouseY >= y)      &&  
    (mouseY <= y + h) )
```

Review: Rectangle Detection

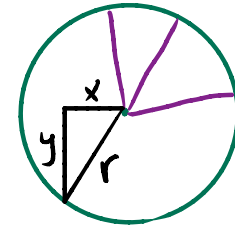
```
if( (mouseX >= x) && (mouseX <= x + w) &&
    (mouseY >= y) && (mouseY <= y + h) ) {
    // do something
}
```

❖ Potential Uses:

- To detect on *every frame*, place in **draw()** or function called by **draw()**
 - e.g. hover detection – change color when mouse is over rectangle
- To detect on a mouse click, place in **mousePressed()**
 - e.g. a button that the user can click on ☆ today

Circle Detection

on edge: $x^2 + y^2 = r^2$
within circle: $x^2 + y^2 < r^2$



- ❖ A circle is defined as all points in a 2-D plane that are equidistant from a center point
 - In mathematical terms, the set of all points (x, y) that satisfy:
 $(x - \text{centerX})^2 + (y - \text{centerY})^2 = \text{radius}^2$
- ❖ To detect the mouse being *inside* the circle, this becomes an inequality
 - $(\text{mouseX} - \text{centerX})^2 + (\text{mouseY} - \text{centerY})^2 \leq \text{radius}^2$
- ❖ In Processing:


```
if( (mouseX-x)*(mouseX-x)+(mouseY-y)*(mouseY-y) <= r*r ) {  
    // do something  
}
```

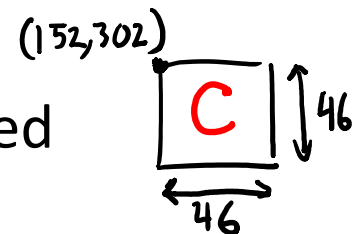
Creating a Button

- ❖ The button needs to be *visible* to the user
 - Use `rect()` or `ellipse()` to draw on your canvas
- ❖ Generally, the user should know what the button is for
 - Use `text()` to either label the button or put directions somewhere else on screen
 - Often, `textAlign(CENTER)` makes finding appropriate coordinates easier

`text("some text", x, y)`

Button Demo

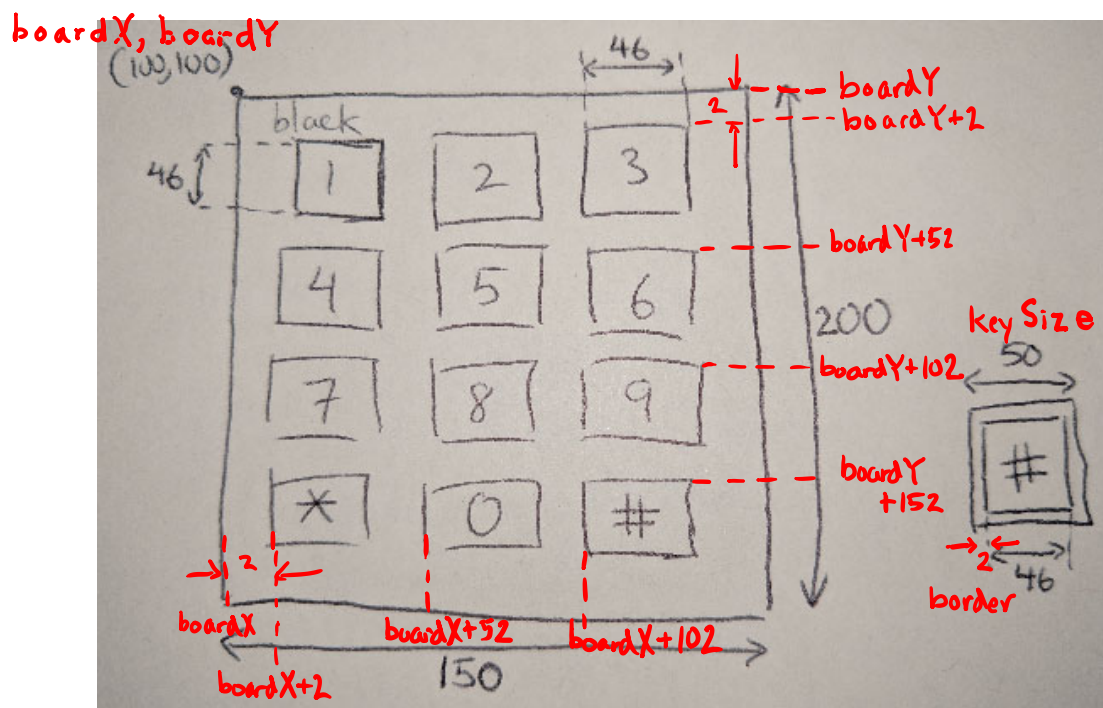
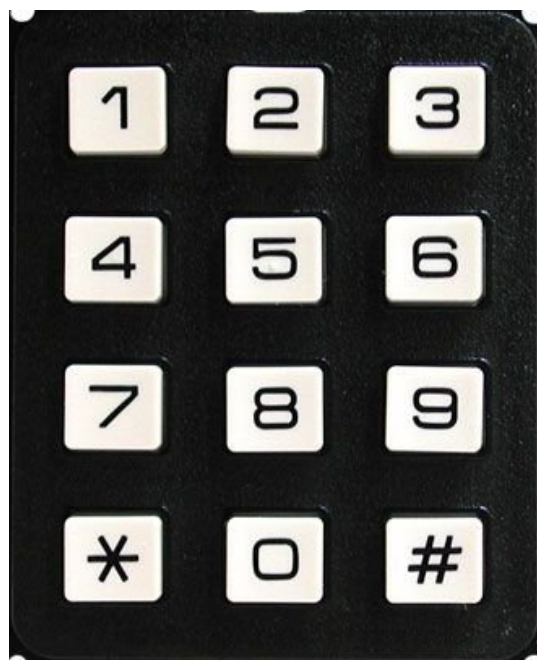
- ❖ Create a “Clear” button for phone or calculator
 - Use a 500 x 500 canvas
 - The button should be of size 46 x 46 and white at position (152, 302)
 - Labeled button with a red “C” text roughly centered
 - Hints: use `textSize(40)` and `textAlign(CENTER)`
 - When the mouse is hovering over the button, it should turn yellow: `color(255, 255, 98)`
 - Requires Active Mode
 - When the button is clicked, it should print “Cleared!” to the console  new ∇ ₆



Grids and Boards

Similar to loops lecture!

- ❖ Grids can be created using nested loops
- ❖ Example: numeric keypad



Grid Demo

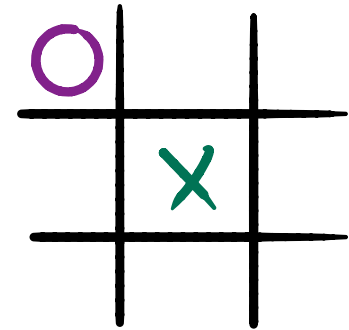
- ❖ Grids can be created using nested loops

```
int i=0;
while (i < 4) {
  int j=0;
  while (j < 3) {
    rect(...);
    j=j+1;
  }
  i=i+1;
}
```

Your Board “State”

```
char [] state = {'O', ' ', ' ', ' ', ' ', 'X', ...}
```

space
character

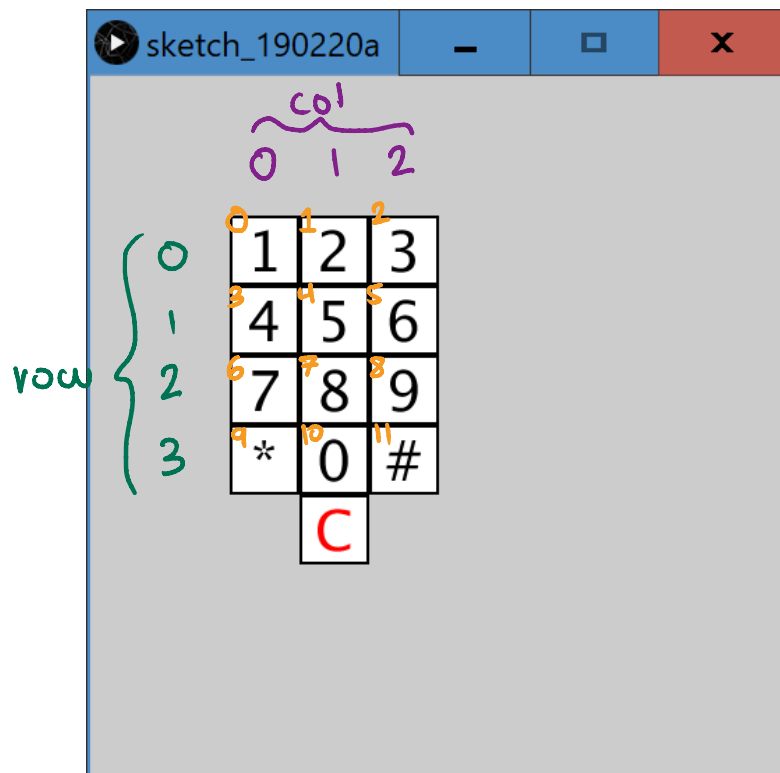


- ❖ The **state** of your board indicates its current configuration
 - In some applications, this never changes
 - e.g. numeric keypad
 - In other applications, this will change over time
 - e.g. tic-tac-toe
- ❖ Board state is typically represented via an **array**
 - Naturally ties a numeric location on your grid to the symbol/value currently associated with that cell
 - Similar to **pixels** [] holding the color “state” of your drawing canvas

Labeling Our Grid

- ❖ For the numeric keypad, the board state is the set of (ordered) key labels:

```
char[] keypad = {'1', '2', '3', ..., '*', '0', '#'};
```



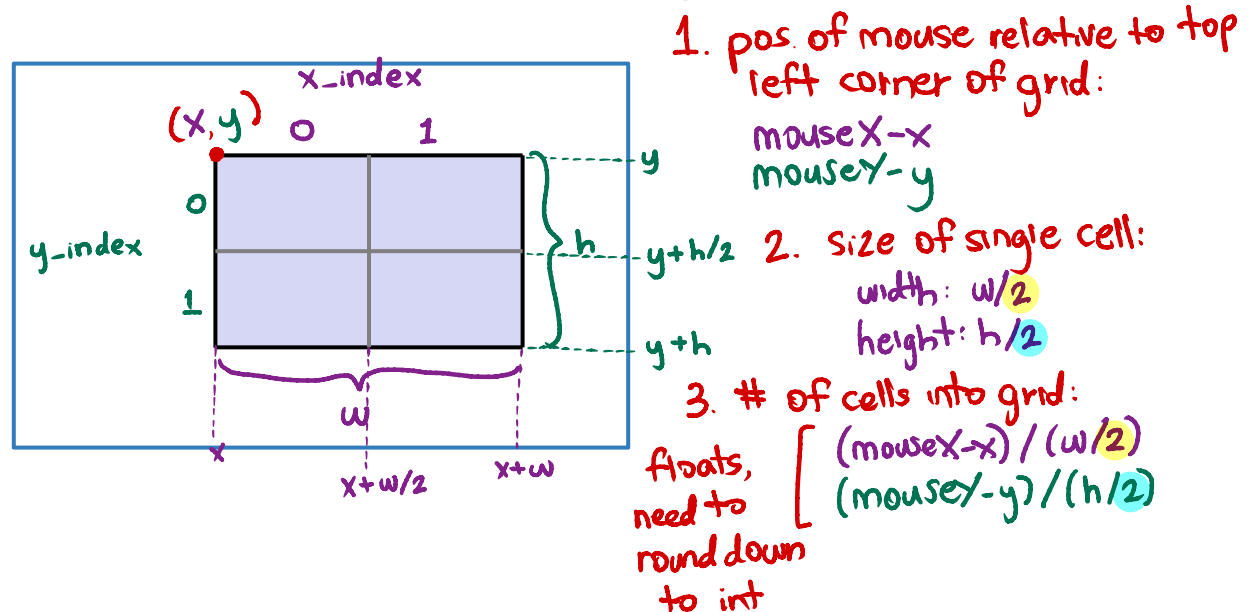
Knowing col and row, how to find index of keypad array?
 ↳ 3 columns total

col	row	index
0	0	0
1	0	1
2	0	2
0	1	3
1	1	4
2	1	5
0	2	6
1	2	7
2	2	8
0	3	9
1	3	10
2	3	11

total # of cols
 index = $3 * \text{row} + \text{col}$
 beginning of correct row
 slide across to the correct col

Grid Detection

- ❖ Detection of mouse location within a *grid*



```

int x_index, y_index;
if( (mouseX >= x) && (mouseX <= x+w) &&
    (mouseY >= y) && (mouseY <= y + h) ) {
    x_index = int( (mouseX-x)/(w/2) );
    y_index = int( (mouseY-y)/(h/2) );
}

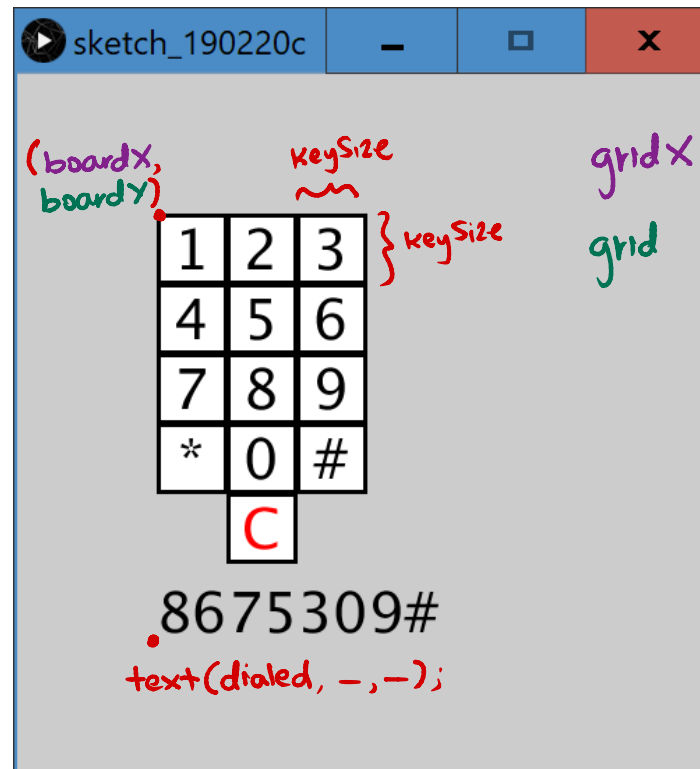
```

round down to int
(drop fractional part)

Keypad Grid Click Detection

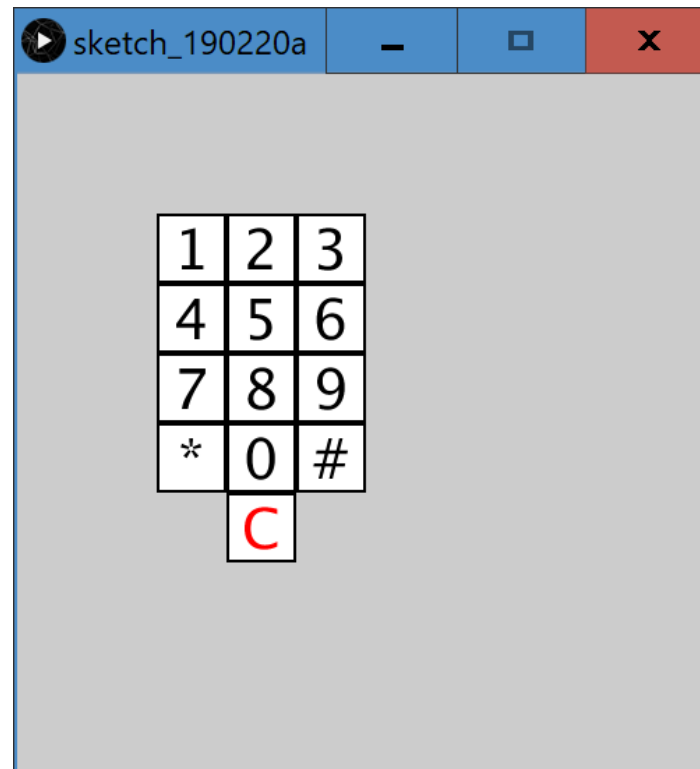
- ❖ Clicking on the keypad should add to the phone number you are trying to dial

- Use a **String** to store and display on the canvas *String dialed = "";*



Clear Functionality

- ❖ Our phone number should “reset” or “clear” when we click the clear button
 - Currently, it prints "Cleared!" to the console



*instead, set
dialed = "";*

Summary

- ❖ Sketched the idea on paper
- ❖ Planned out coding representations
- ❖ Built on previous work by adding one function or idea at a time
- ❖ Ran the program after *every* improvement to make sure that it worked correctly
 - Unit and integration testing!!!