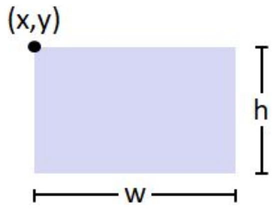
Section 15: Boards and Buttons

Introduction: Defining clickable regions is a commonly-desired feature in programs. These usually come in the form of game boards and buttons. We can implement them in Processing using some neat mathematical tricks. Note that these aren't new constructs in Processing, but just combining techniques we've learned about previously!

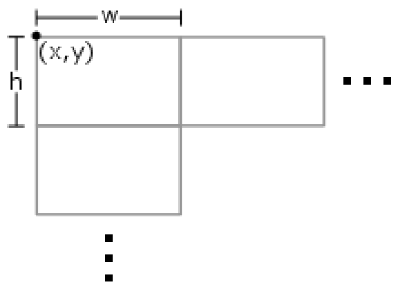
Buttons: The simplest way to define an on-screen button will be a rectangular region of your drawing canvas, as shown in the image to the right. If the upper-left corner is at position `(x,y)` and the dimensions are width `w` and height `h`, then we can say that the mouse is *within* the button if `mouseX` is between `x` and `x+w` and `mouseY` is between `y` and `y+h`. This translates very naturally into a conditional statement.



- This can be used in a hover sense (is the mouse currently over the button?) within the normal execution of **`draw`**`()`, or it can used in the clickable sense (did the user click on the button?) within the `mousePressed()` or `mouseReleased()` functions.

Note that a circular button can also be created/detected using a method similar to how we detected Elli's proximity to apples in the Controlling Elli assignment.

Boards: Game boards are often created from regular grids of rectangles. Instead of drawing out each rectangle individually, typically we can use nested for-loops to condense the drawing statements. Let's assume that we want a grid of `rows` rows and `cols` columns, with the upper-left corner of the grid at `(x,y)` and each grid cell of width `w` and height `h` (see image on the right).



Notice that the upper-left most cell will be a rectangle drawn at (`x,y`). The cells to the right of it will be drawn at (`x+w,y`), (`x+2*w,y`), and so forth. The cells below will be drawn at (`x,y+h`), (`x,y+2*h`), and so on. Taking advantage of this pattern, we can use a nested for-loop similar to the one shown below:

Example:

```
int i = 0;
int j = 0;
while (i < rows) {      // move to next row row
   while (j < cols) {   // move across this row
      rect(x + w * j, y + h * i, w, h);             // draw cell
      j = j + 1;
   }
   i = i + 1;
}
```

Detecting a click on a grid is trickier. We could write an individual conditional for *each* cell, similar to a button, but that'd be super tedious and repetitive for larger grids. Instead, we'll use some math to help us out:

If the mouse is currently over a grid cell, then `(mouseX-x)/w` will tell us how many cell widths over we are and `(mouseY-y)/h` will tell us how many cell heights down we are. Both of these quantities are fractions and thus would normally be stored in `float` variables. However, we don't really care about the fractional part. If we are more than 1 cell width and less than 2 cell widths over, then we are in column 1 (counting from 0). It turns out that if we convert a `float` to an `int`, then we truncate the fractional part by just dropping it (instead of rounding). This allows us to easily get the row and column number in the grid of our current mouse position!

Exercises:

1) You should review the posted code (`keypad.pde`) from the “Buttons & Boards” lecture. Take the time to talk through the code with your partner and the TAs to make sure you understand every piece of it.

2) Find a partner then go to the course website and start working on the lab titled “Tic-Tac-Toe.”