Lab Exercise 6: The Mice And The Owl

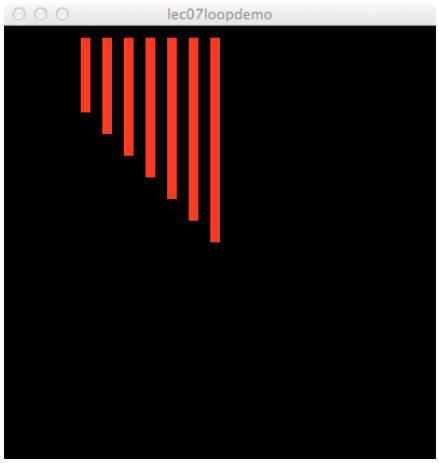
Goal

The objective is to get more practice controlling action on the screen. The programming in this exercise is the kind you've seen before. The goal is to work out the logic of the computation.

for-loop practice

Create a practice program using this code from lecture 07:

```
void setup( ) {
    size(800, 400);
    background(0);
    fill(255,0,0);
}
void draw ( ) {
    for (int i = 0; i < 5; i = i + 1) {
        rect(10+20*i,10,10, 10);
    }
}
```



Spend a few minutes playing with the for-loop: the limit value (5), the starting value (0), the increment (1) and the parameters of the `rect( )` function. Make the image at right.

Base Program

Open a new Processing program, and copy/paste the mouse.txt code linked to this lab on the Calendar page. Notice that this is actually five functions: `setup( )`, `draw( )`, `pick( )`, `owl( )`, and `mouse( )`. Run the program and notice the mouse.

for-loop programming

Create six rows of mice, each row containing 8 mice. This will require two functions. The first function, `mouseRow( )`, will draw a row of 8 mice with one statement, which is a loop of the form

```
for (int i = <starting value here>; i < <limit value here>; i = i +1) {
    <a call to the mouse function goes here>
}
```

Notice several features of the loop. First, the loop variable, `i` , is declared to be an `int` at the start of the control portion of the loop. You need to fill in the starting value of `i`, the limit value of `i`, and the call in the body of the loop; that is, the code that is repeated. The value of `i` will be tested to be less than (<) the limit value, meaning that this is the first value that would be too big. So, for example, if the starting value is 0 and you want to

make eight mice, then the limit must be 8, because you think about the mice as being numbered, 0, 1, 2, 3, 4, 5, 6 and 7. Consider what parameter(s) you will need – for example to position the row – and then write `mouseRow()`, so that mice are placed every 50 pixels.

The second function, `mouseArray( )`, will draw six rows of mice using the `mouseRow( )` function. Its only instruction will also be a for-loop having the very same structure as the last one, except that it calls a different function in its body. Write `mouseArray( )`, so that mouse rows are placed every 100 pixels.

When the `mouseArray( )` computation is done, replace `setup( )`'s `noLoop( );` line with the call to `mouseArray( )`. It won't flicker that way.

The owl appears

In `draw( )`, call the `owl( )` function so it appears at the right end of the first row after the last mouse. Recall that the owl is 70x100, and that the 0x0 point is at bottom center. The owls y-position should be offset by an integer called `oWhy` that is initialized to 0. The top of the canvas might look like this:

if-statement

Now we will use an if-statement to make a choice. Each time that `draw( )` is called it will either erase the owl, or redraw it. For the purpose of controlling the two cases, we declare a variable of datatype Boolean and initialized it to `false`. (Recall that Boolean variables are either `true` or `false`, and that the words 'true' and 'false' are constants … Processing will color them.) So, we write

```
boolean erase = false;
...
draw( ) {
   noStroke( );
   if (erase) {                          //check if erase is T or F
      <overwrite the owl with a white square>  //it's true, wipe out owl
  } else {
      <draw the owl at position oWhy+100>  //it's false, show owl
  }
}
```

Now the only question is how to set the erase variable. That's easy, we just use our friend the keyPressed( ) interrupt function (last lab), and assign erase = `true`. Try it out!

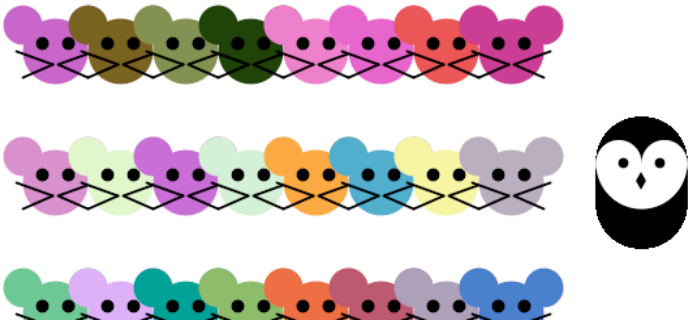
The Owl Choose

At the moment the owl should disappear when a key is pressed. After we have erased it we will cancel the erase request (erase = false) and we will advance the `oWhy` variable to move the owl to the next row of mice. That operation uses mod

```
oWhy = (oWhy + 100)%600;
```

and causes the owl to drop down to the next row. Add these two assignment statements at the end of the "overwrite" code in the if-statement.

Now, as the keys are pressed the owl should "fly" to different rows of mice. In this figure of the top of the screen, one key press has moved the owl to the second row.



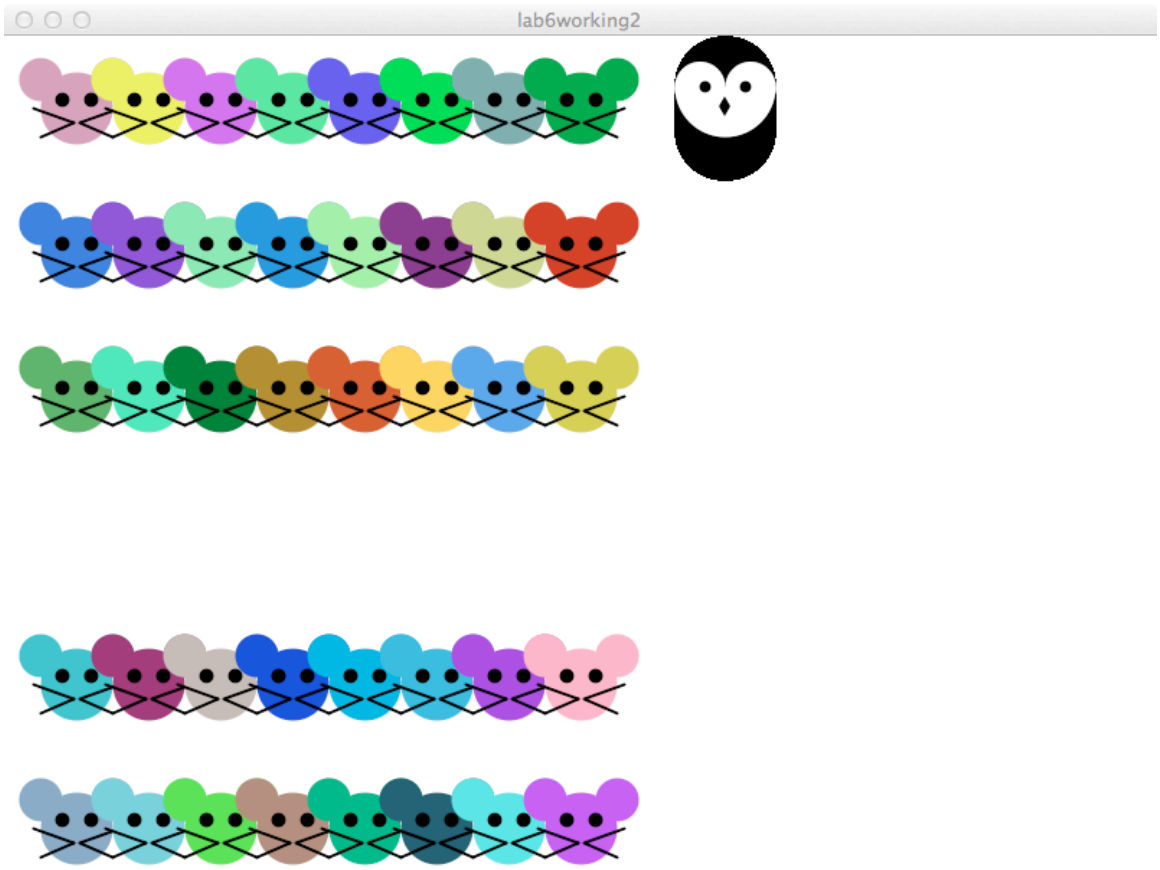
The Mice Panic

To complete the project we add one more if-statement after the one we already have in our `draw( )` function. This one is based on a boolean variable `panic`, which is declared and initialized to `false`. It doesn't make a choice like the last time, but only applies when `true`, i.e. there is no else. We write

```
if (panic) {
    <overwrite all mice with a white rectangle for row oWhy+100>
    <cancel the panic indicator>
}
```

The result is to completely "white out" the row of mice if the owl is at their row, and the mouse is clicked. So, using our old friend `mousePressed( )`, we set the panic variable to `true`.

Try it out! Can you erase several rows? Here is a screenshot after these actions: key, key, key, click, key, key, key. Another click would erase the top row.



Wrap Up

The program draws the mice using for-loops, saving us considerable typing. Using if-statements, we were able to develop some simple logic based on two boolean variables, `erase` and `panic`. We needed to know when to erase and move the owl to a new location, and we did that through the `keyPressed( )` function. We needed to know when to zap a row of mice, and we did that through the `mousePressed( )` function.

Be sure the portion of the program that YOU wrote is commented, and submit it to the class dropbox.