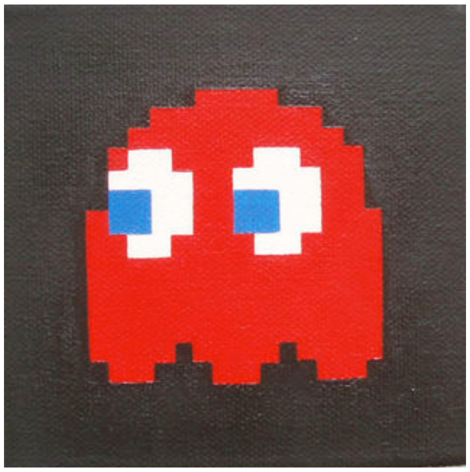
Homework 7: Blinky Stars

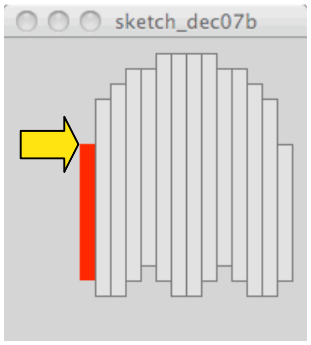
Goal: The purpose of this assignment is to give you experience programming in an interactive development environment by just drawing Blinky's picture using code. At the end we will produce Blinky looking in different directions.



In the ancient game of Pacman one of the characters was a red ghost called Blinky. Blinky is made up of rectangles of three colors, and the easiest way to draw it is to do one color at a time, beginning with red.

The effort begins as usual, so please follow these steps.

Step 1. Open Processing and set up a 200x200 size canvas. Also, you don't want your rectangles outlined, so specify `noStroke();` and, you will want the fill color to be red.



plan for drawing Blinky

Step 2. We take each "square" region in the Blinky picture above as covering 10x10 pixels on the screen – so it's blue irises are 20x20. The easiest way to draw the ghost is to draw vertical strips each of 10 pixels wide. So, we begin at some arbitrary position on the canvas, say, 50, 70, and begin drawing; the spot corresponds to the yellow arrow marked in the figure. Recall that this position is 50 pixels across and 70 pixels down from the upper left corner. Then we use the rectangle drawing function

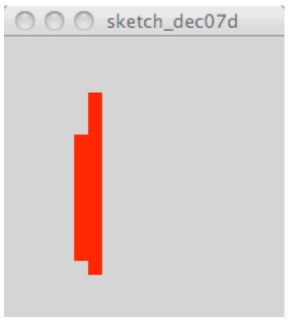
```
rect( 50, 70, 10, 90);  //From left, bar 1
```

remembering to end the command with a semicolon. From the book (*GSwP* p. 17), we recall that the parameters are

rect(<*x*>, <*y*>, <*width*>, <*height*>)

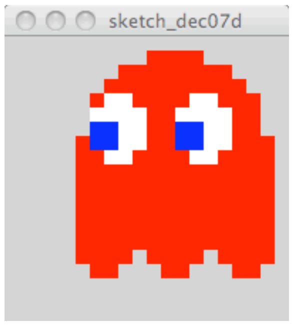
So this is a 10 wide by 90 tall rectangle, positioned so it's upper left corner is at 50,70 on the canvas. It will be the left side of the ghost's body. Notice the comment.

Steps 3-n. To complete the red, you will add 10 wide vertical bars properly positioned so the figure comes out looking like Blinky, and comment them. I want you to do it using this protocol: Figure out (in your head, probably) where the corner of the next bar should be, type in the rectangle function with that position as the first two numbers, give a width of 10, and then figure out what the new height must be. Comment. Then check it, that is, run it. If you are like me, you'll mess them up at least half the time, but that's OK, the computer doesn't care if you make mistakes. (If you don't want to figure out the positions in your head, print off some more graph paper like you used in the Extra Credit assignment.)



So, the next bar must be at 60 pixels over (the last one was at 50 pixels over, and the bar we drew was 10 wide), and it must be 40 pixels down (the last one was 70 down, but the second bar needs to start 30 pixels higher up, right?). It will be 10 wide, of course, and how long? It's longer than the last one – it was already 90 long – by an additional 30 at the top and 10 at the bottom. So the new one must be 130 pixels total. See the figure.

After typing in the command, check the drawing to make sure it is coming out right, that is, run the program. Smart programmers check very often (after each bar in this case!) so that they continue to produce good code, and minimize the number of errors.



left

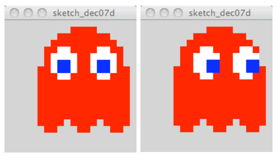
You should feel like you are developing a "rhythm" creating these red bands, adding comments, checking them, correcting them, over and over. Occasionally, saving your file.

Googly Eyes

Next, place the white for the eyes and then the blue iris, as shown in the figure above. You will need more fill commands as well as more bars. At this point, eyes will be a piece of cake! The final result is shown in the accompanying figure.

Assignment Part 1. Save the `.pde` file and name it <*your name*>`LEFT.pde`; you will turn this one in.

Looking Around



center *right*

Next we notice that there are actually three positions for the eyes, which we're calling *left*, *center* and *right*. And we notice the following properties of these:

left: has the form as drawn
center: whites moved right 10 pixels from *left* and irises moved right 20 pixels from *left*
right: whites moved right 10 more pixels and irises moved right 20 more pixels from *center*

What changes would be required in the program to change the current drawing into looking a different way? Obviously, the changes have to be given in terms of positioning the rectangles. Let's look at the white rectangles first. To look left there is 0 shift in each white rectangle because we drew it looking left. To look to the center there is a shift of 10 in each of the white rectangles, and to look right, each one has to be shifted 20 from the base design. (See the accompanying figures.) So, we have

```
left:      rect( 60 +  0, 50, 10, 30);
center:    rect( 60 + 10, 50, 10, 30);
right:     rect( 60 + 20, 50, 10, 30);
```

for the white bar on the left side of the left eye; all of the other white bars are adjusted by the same amount.

The blue irises work similarly, except that the amount they have to move is not 10 with each change, but 20. Check to see that. (And remember this fact for later!)

Shifting Blinky's Eyes

We want to draw Blinky looking in one of the three ways. To do that, we will declare an integer variable called look at the top of our program. We will use look to represent the direction Blinky is looking, coded as

0 means looking *left*
1 means looking *center*
2 means looking *right*

So, to make Blinky look left we give look the initial value 0 in the declaration line at the top of the program.

Now, here is the clever part. Why did I choose 0, 1 and 2 to code the three different looks? Because if I multiply look times 10, I get

look * 10 is 0 when look means *left*, that is, look has value 0
look * 10 is 10 when look means *center*, that is, look has value 1
look * 10 is 20 when look means *right*, that is, look has value 2

So, look * 10 is exactly the amount we need to add to the <*width*> parameter of all of the white rectangles to shift them properly to change the eyes, as in

```
rect(60 + look*10, 50, 10, 30);
```

for the first rectangle on the left side of the left eye.

What must we add to all of the blue rectangle's <*width*> parameter to move the irises?

Assignment Part 2. Change the program as you wrote it for Part 1 so that it includes a new integer variable, look, which when initialized to 0, 1 or 2, draws Blinky looking left, center or right. (Your program is not active, so you need to set the value in the declaration each time and run it to see the result. You can do only one look at a time; it won't move (yet).) Save the .pde file, naming it <*your name*>LOOK.pde.

Summary

You have written another Processing program and commented it; this one parameterizes rectangle positions.

Turn In

Submit your two programs at the course drop box.