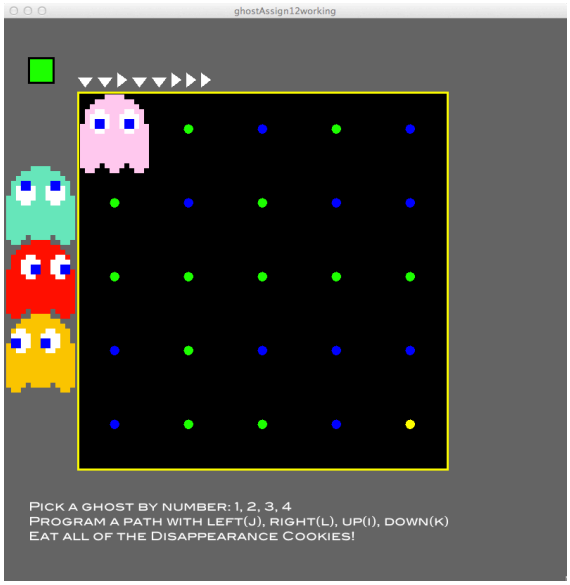
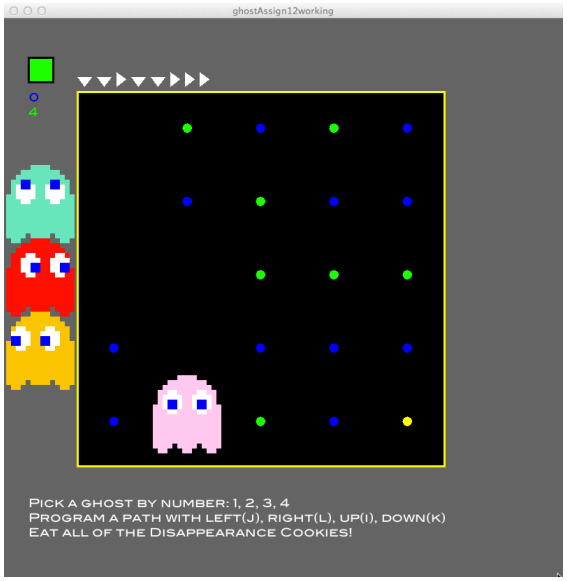
Homework 12: The Ghost Game Project

Goal: The purpose of this exercise is for you to build on your earlier code, and write a more substantial program.

You have nearly a week to work on this program, and so it is somewhat longer than a typical assignment. HOWEVER, it is not overwhelming, because it builds on what you've done before in many aspects. We're just adding a "little bit more" and that will make it much more interesting for you.

My Game: Because each of you will write a somewhat different game, I want to tell you how mine works. Yours will have the same base parts, but you'll extend it to make it better.





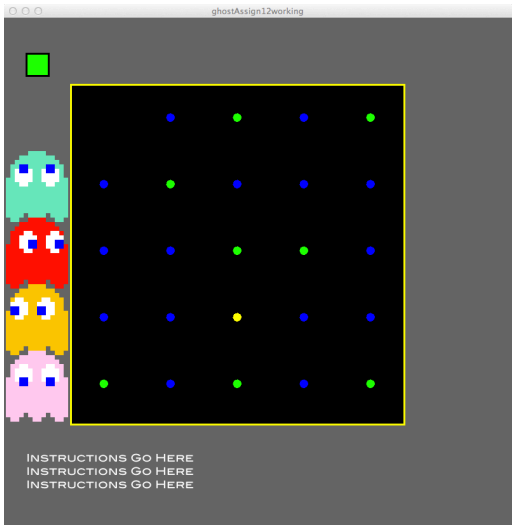
Mine (and yours) uses a layout similar to last assignment. The big difference is that the user programs the motion first, then runs it … like the Lightbot did. Here on the left is my game, programmed and ready to go. My goal is to get to the yellow dot – my transfer into another world, with a minimum of touches of blue dots; green are OK. To run my program, I click the green button at upper left. As the ghost moves around, the green and blue dots are recorded at the upper left.

My game is simple, but it introduces all of the ideas we will use in this project, which are:

- Display set up and initialization
- Programming
- Running the animation
- Protecting against errors
- Recognizing a prize

To these basic parts, you can add your own ideas.

Step 1: The Display. The display is like Assignment 11, but it is probably smarter to begin this assignment with a fresh program, Assignment 12, and rebuild it by copying parts from Assignment 11. (Though the programs are similar, morphing one to another is usually harder than simply stealing the parts you need to write it fresh.) The checklist is:



- Copy declared variables, nextTri, xpos, ypos, xmove, ymove and the color var
- Copy the setup( )
- Use the new board code provided with this assignment
- Copy all code to place the ghosts at left
- Copy if statements for the 4 keys for a new keyPressed( ) function to pick color
- Copy code to redraw ghost in corner when color changes as the draw( ) function

Notice that you are not using two keys that you used last time, animate( ) and maybe other parts. The program should pick the ghost as before, and should be quite clean!

Finally, check out how to load a font (it was in an earlier lecture), pick a font, and print at the bottom of display: "Instructions Go Here." You will change these later. Try it!

Step 2: Programming. Like Lightbot, the ghost is programmed ahead of time, and then run to perform the game. We use triangles as we did in Assignment 11, but this time we need four, one for each direction. Also, *we need to store the code so we can run it later.* So, declare a new global string variable,

```
String moves = "";                    // declare string containing empty string
```

to record the "program" of moves, which we call the **move code**.

While we're declaring variables, I declare two global integers, row and col, each initialized to 0. These are used to keep track of where on the grid of dots the ghost is.

Next pick four keys – I am using j (left), k (down), l (right) and i (up) for this explanation, but you can use any keys you want. As before, add if-statements to keyPressed( ) to recognize each key, and for each key, draw the triangle at the top, and add the "code" to moves. For example, I would write

```
if (key == 'j' ) {
  triangle ( … );
  nextTri = nextTri + 30;
  moves = moves + "L ";
}
```

The two letters 'L' followed by a space, become the "move code" for this move. (The space will be explained below; the '+' is concatenate.) I use 'L ' for *left*, because that is the meaning of the j-key; I use 'D ' for *down*, 'R ' for *right*, and 'U ' for *up* for the other keys. Each is followed by a space.

Finally, at the end of keyPressed( ) I include println(moves); to watch the progress of my program at the bottom of the Processing window. (This can be removed at any time.) Referring to the first picture above, the user will type kklkklll and the moves variable will be D D R D D R R R That's the program! Try it!

Step 3: Running The Animation. Unlike the animation( ) function in Assignment 11 that moved the ghost as soon as the user typed a letter, this animation will read off the operations from the moves variable, that is, the move code. Also, because the user can start and stop the game, we must test before animating to see if the button has been clicked. So, we set that part up first.

Button: Declare a global Boolean variable, go, and initialize it to false. (Recall that boolean is a datatype and that its only possible values are true and false.) Next, define a mousePressed( ) function that (a) flips the meaning of go, and (b) changes the color of the button to the "other" color. Flipping a boolean variable to the other value is easy using the not operator (go = ! go;). Flipping the color between red and green requires the "three statement exchange" that was described in lecture slides in an earlier lecture, and probably, three new integer variables. Try it!

Animate: The new animate( ) function will be called in the draw( ) routine, and has the form

```
void animate( ) {
        … draw the ghost's body at xpos, ypos in black to erase it
        if ( go ) {
            ... the actual motion code goes here
        }
        ... draw the ghost in color at xpos, ypos
}
```

which will stop all animation if go has the value false.

For the *"... actual motion code goes here*" part we need to insert motion instructions like we used in Assignment 11. For example, to move in the x-direction we need

```
        if ( xmove > 0 ) {
            xmove = xmove – 1;
            xpos = xpos + ?? ;
        }
```

but now we could be moving forward or backward in the x-direction. So, although the ?? was a 1 in Assignment 11, it now has to be dir, a new global integer which will be either +1 or -1. (Recall we used the idea of changing forward or backward direction in an earlier lecture.) Obviously, we need an if-statement for ymove, too.

In summary, if go is true, one of two if-statements will cause the ghost to move if it is supposed to move. We get the amount to move by processing the moves variable.

Program Counter. Recall that while the Lightbot walked around, its place in its program was displayed by a yellow highlight on the instruction it was performing. We will need to keep our place in the moves code, too, so we declare a global integer variable place, initialized to 0. We refer to the program instructions stored in moves with

```
moves.charAt(place)
```

which gives the character (letter) in the moves string (the move code) at position place; the first position is 0, so if place is initialized to 0, moves.charAt(place) refers to D in our example code above. Obviously, to refer to the next letter, we increment place, which in the example will be a space. And so on. The only thing we need to avoid is for place to get too large, and "go off the end" of the moves string. We can tell how long moves is using the phrase moves.length( ). By the way, the length is the number of letters in the string, but because they are numbered starting at 0, the last letter of a 10-letter string is in position 9. That is, the last position is one less than the length.

Interpreting the Letters. If go is true, then if the ghost's last movements are over with (xmove == 0 and ymove == 0), we need to run the next instruction … if there is another instruction. So we check,

```
if (xmove == 0 && ymove == 0 && place < moves.length( ) – 1 ) {
  … handle the cases of the five different letters
  place = place + 1;                    // move to the next "instruction"
}
```

where the five letters are D, U, R, L, and space. For each letter we need an if-statement to test to see if moves.charAt(place) equals that character, and we do three things: (a) set xmove or ymove to 150 depending on whether the ghost is going in the x- or y-direction, (b) set dir to be either +1 or -1 depending on whether it is supposed to go down or up, or right or left, and (c) adjust either row or col by +1 or -1 depending on the direction of motion for that letter. (The if-statement for space is explained in the Prize section below.) For example,

```
if (moves.charAt(place) == 'D') {
  ymove = 150;
  dir = +1;
  col = col + 1;
}
```

At this point you have changed from Assignment 11 where the ghost moved from your keystrokes, to a version where the ghost moves under program control. Congratulations. Try it!

Step 4. Check For Out-of-Bounds. This is an easy extension to your program. We want to be sure that the code doesn't tell the ghost to walk out of the board region. So, we write a small check routine, let's call it check( ), that tests to be sure that row is between 0 and 4 (inclusive) and the same with col. If either one is out of bounds, stop (set go to false) and print a warning somewhere on the screen: "Illegal instruction". check( ) should be called in 4 places: right after each change to row and col.

Step 5. Making A Better Game
So far your program solves the last assignment with a stored program rather than using real-time motions. That is the goal of this assignment. But, at this point everything is in place to make an interesting game out of it, and you are free to do that however you like.

Below, I explain how my game works in order to show off two techniques you may want to use.

Prize. The prizes are generated at the time the board is created. They are the colored pills. Some are to be avoided, some are OK, one is special. Like the ghost's path – which is specified at one time and used later – it is necessary to remember what was placed at each position. The information will allow us later to check, when the ghost gets to a square, what was placed there. So, like moves, we add a String variable called prize to record the information,

```
String prize = "";
```

It is initialized to the empty string. Then, as we generate the prize images for display on the board, we record what they are.

```
void prize ( ) {
    int row, col, newworld;
    for (int i = 0; i < 25; i = i +1) { // Need 25 prizes for board
    if (random(0,2) < 1 ) {             //Is it blue or green?
        fill(0,0,255);                  //blue
        prize = prize + 'B';            //record it
    } else {
        fill(0,255,0);                  //green
        prize = prize + 'G';            //record it
    }
        row = i % 5;                    //what row is it in?
        col = int(i/5);                 //what col is it in?
        ellipse(225+row*150, 225+col*150, 20, 20);  //draw it
  }

  newworld = int(random(1,25));         //Where is exit to new world?
  fill(255,255,0);                      //Set color to gold
  row = newworld % 5;                   //Figure it's row
  col = int(newworld/5);                //Figure it's col
  ellipse(225+row*150, 225+col*150, 20,20);  // draw it
    //Now pull apart the price string and stick the y for yellow
    //into it at the right place
  prize = prize.substring(0,newworld) + "y" + prize.substring(newworld+1,prize.length());
  println(prize);                        //Watch it work!
}
```

There are two basic parts to prize( ) creation. The first part places 25 pills that are either blue or green on the board. (Of course, you could place anything else!) After they are set, the special dot (yellow), which moves the ghost into a new world, is placed on the screen and then recorded in the prize string. Notice that to put the 'y' for yellow in its right position, we need to separate the prize string into the part before the 'y' position (from 0 up to, but not including the newworld position where 'y' goes) and the part after the 'y' (from the position after newworld to the end), and then "glue" (concatenate) the three parts together.

Recognizing Cell Contents
We said earlier that in the animate routine, in the portion where we read the operations (L, R, U, D) from the move code, we would process the space (following each letter) later. That time is now. The space is where our ghost-moving computer checks to see what is in the cell that the ghost has just gotten to. How does it find out where it is? By using the row and col values. So, when we find a space (moves.charAt(place) == ' ') in the "go-part" of the animate( ) function, we can look at what the prize is on that position by computing

```
myLoc = row*5 + col;
```

and then looking at the letter in the prize string using

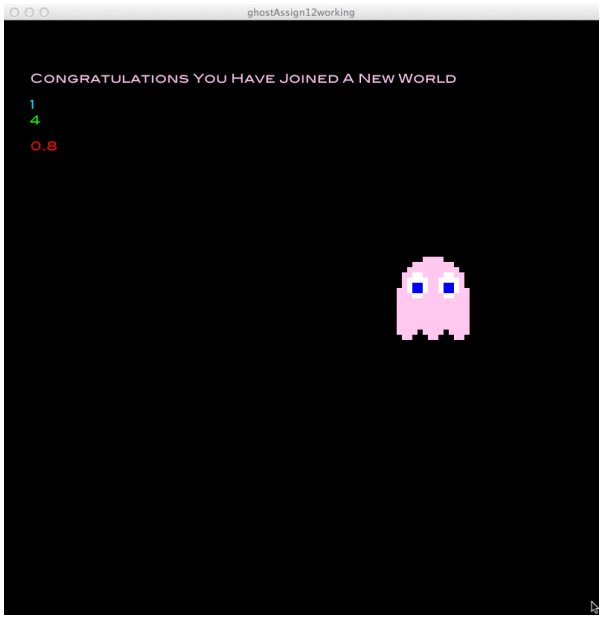
```
prize.charAt(myLoc) == 'y'
```

or whatever the test should be. Obviously, you will need to declare the variable myLoc (but it can be declared in the animate( ) function because it is only used there).

In my program, when I process a space, I check if it is a 'B', indicating a blue pill, and count it in the blue total; if it is a 'G', indicating a green pill, I add one to the green total. And if it is a 'y', I blast the whole page away by drawing the background and adding a greeting. "Congratulations, you have joined a new world"! Notice that I print out the blue and green totals as the ghost is walking around. Again, you can decide to do anything you'd like.

OK, so for the Prize part of this project, you need to make some objective to the game – something for the player to "go for". You could program the game as I've done it, but you can do anything else that seems interesting to you. (This course is about creativity!)

Complete your game by going back to the text(…) commands you put in at the beginning, and say how your game works.



Wrap Up. You have written a very substantial program – mine was more than 220 lines; yours may be more. This is a tremendous accomplishment. You achieved it by using code you wrote before, and using the same ideas over and over again. If you were smart while you programmed, you used copy/paste a lot, taking code from your earlier assignments or lectures. That definitely saves the brain for the few places where actual thinking is required. But, you did that, too!! Congratulations!!

Turn In. Submit your code to the dropbox. If we need special instructions to maximize our enjoyment of your game, please give them. Thanks!