Lab Exercise 7: Controlling Elli

Goal: To control the interactive nature of the Elli worm.

Array Review

Recall that Elli uses a seven element array to store the x-coordinate value for each segment (elliX[ ]), a seven element array to store the y-coordinate value for each segment (elliY[ ]), and a seven element array to store the color of each segment (possibly named, elliColor[ ]). These fully describe the rendering – display on the screen – of the worm.

Recall also that to move Elli, the segment positions are shifted left in the elliX[ ] and elliY[ ] arrays one element, and a new position is created for the head. That means that the segments all follow the head, each one occupying its position. Our interest this time is in controlling the position of the head.

Motion Settings In HW 11

Recall that Elli's new position was specified in HW 11 by two lines of code,

```
elliX[6] = elliX[6]+20;          // a new x position for head to move right
elliY[6] = elliY[6]+20;          // move Elli down screen
```

The first of these advanced the worm right, the second moved the worm down; together they created a diagonal motion to the lower right corner of the screen.

Adaptive Motion

Our goal now is to allow general motions of the worm around the screen. So, we generalize these two lines; the first will control horizontal motion, the second will control vertical motion. That means the "new head position" in your code needs to be edited to be

```
elliX[6] = elliX[6] + horiz;     // move horizontally
elliY[6] = elliY[6] + vert;      // move vertically
```

The two integer variables, horiz and vert, need to be declared at the top of the program, and they need to be initialized to 20. At this point, Elli should behave just as it did before you edited the two lines. Check it!

Controlling Elli with Cursor Keys

The next step is to set up an interface to the cursor keys on the lower right of the keyboard. Recall that the use of these keys is explained in References > Keyboard > keyCode. READ THIS – IT TAKES 1 MINUTE.

Set up a keyPressed( ) function along the lines given in the Reference. If the key is CODED, and the user has pressed the right pointing arrow, then you need the code

```
if (keyCode == RIGHT) {
    horiz = 20;                          // set right motion
}
```

to set the amount to move the head right. (Of course, you already initialized horiz to 20, so the first time, this won't matter; but later it will.)

☛ 'his requires ome thought

You need **four tests** of this form that check the four directions of keyCode, and change the correct direction variable (horiz or vert) and change it by the correct amount, either 20, -20, to achieve the correct motion. Ask, for example, "What should happen when the user clicks UP?" Answer: Vertical motion must become negative. Make the four if-statements in the keyPressed( ) function, if the key is CODED. Note that if the key is not CODED, do nothing.

Try out the program. It should do everything that the original program did, plus it is capable of going diagonally in other directions.

Sharp Turns

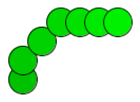
Next we want to change from always going diagonally, to also being able to go vertically and horizontally. The reason Elli doesn't go vertically and horizontally now, is that we keep adding (or subtracting) 20 each time from both coordinates. What we need to do is set one of them to 0, which prevents it from moving further in that direction. So, we modify the logic we already have for the cursor keys. We change the if-statement we wrote earlier

```
if (keyCode == RIGHT) {
    horiz = 20;                  // set right motion
}
```

to become

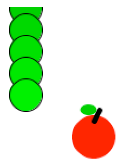
```
if (keyCode == RIGHT) {          // handle diagonal & sharp right turns
    if (horiz == 20) {           // did we previously ask to go right?
       vert = 0;                 // yes, it's 2nd time, quit vertical motion
    } else {
       horiz = 20;               // no, 1st time, set right motion
    }
}
```

The new code works like this: If Elli is not going right at all, then pressing RIGHT causes the worm to begin going right diagonally. But if the worm's already going (diagonally) right, another press of RIGHT means, go straight right, with no vertical motion.



Make the similar changes to the other directions, being sure to get the tests, the variable names and the signs on the constants perfect. Test your code – play with it!

Create An Apple



Write a function, apple( ), to draw a 40x40 red ellipse at the integer locations, applX, applY. The integers must be declared at the top of the program, and assigned a random position on your canvas. Call the apple( ) function in draw( ). A fancy apple is not necessary; a simple ellipse is enough.

Elli Meets The Apple

Write a final function called meet( ). In it you will test to see if Elli's head has reached the apple. How do you do this? You check to see whether the two coordinates of Elli's head are close to the two coordinates of the apple. "Close" is tested by subtracting the items and seeing if their absolute difference is small. Here is the test:

```
If ( abs( elliX[6] – applX ) < 25 &&          // Are the x values close?
     abs( elliY[6] – applY ) < 25 )           // Are the y values close?
```

Notice that both dimensions are tested (see the &&) by subtracting the two positions, computing their absolute value with the abs( ) function, which makes any number it gets positive, and then checking if that number is less than 25. If so, then the meet( ) function should pick a new random position for the apple; otherwise do nothing. Call the meet( ) test in draw( ). If Elli meets the apple, on the next redraw background( ) will eliminate the old apple, and the apple( ) function will draw the new one. Try it!

Elli is a little weird because the positions of the ellipses are figured from the upper left corner of their bounding box, not the center; read about it under ellipseMode( ) in the references. By adding ellipseMode( ) and correcting the size of the segments and apple to be radii rather than diameters (that is, divide them in half), Elli will be less weird.

Wrap-Up You have used that fact that the three arrays, elliX[ ], elliY[ ] and elliColor[ ] completely describe the worm. The motion of the head defines where the worm is going because the segments simply follow along by "shifting". We set up the cursor keys so that one press results in a change of motion diagonally in that direction, and a second press results in a sharp motion in that direction. We added an apple, and a test to see if the worm had reached the apple.

Turn-In After completing all steps, including giving Elli some facial features, submit your .pde file to the class dropbox.