Mouse Control

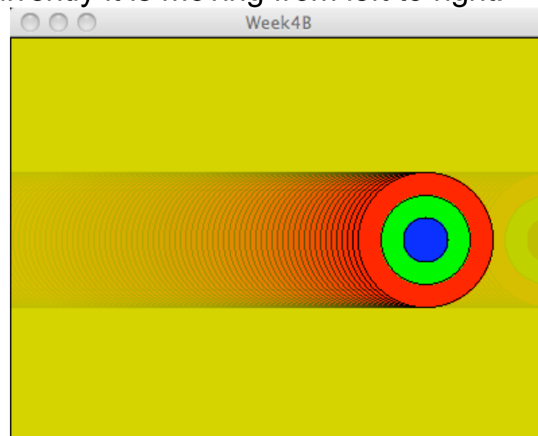
In previous classes and assignments we have used the mouse to position a figure. We can use the mouse in other ways that might prove interesting. Later we will clicks on buttons to tell the program what to do. For this set of notes we will look at one way of using the mouse to “influence” the motion of a figure rather than directly dictate where the figure must be.

Our past efforts looked something like this;

```
drawTarget( mouseX, mouseY, td, td );
```

where td was the target’s diameter. In this example the target was placed exactly on the location of the mouse’s pointer.

Let’s change courses. Suppose we have the target moving on the screen in a wrapping motion. Currently it is moving from left to right.



What we want to do is to be able to speed up and slow down the movement using the mouse instead of the keyboard. How could we do that?

First, let’s take a look at the mouse functions in the API:

Mouse

mouseButton
mouseClicked()
mouseDragged()
mouseMoved()
mousePressed()
mousePressed
mouseReleased()
mouseX
mouseY
pmouseX
pmouseY

Notice that there are some functions (they end with two parentheses) and variables (no parentheses) with the same name. Some of you have used Processing in other classes. And some of you used the variables. My sincere advice is to use the function form and NOT the variable. There are subtle differences in the behaviors of the variables and functions that may well cause

Copyright Jim Roberts September 2012 Pittsburgh, PA 15221 All Rights Reserved

you problems in future work.You can read the API to see what Processing has to say about how these work.

We are going to look at mouseMoved( ) and mouseDragged( ).

We can use either to solve our task of changing the speed with the mouse. The difference between these two is that the mouseDragged( )requires that the button be pressed down when the mouse is moved in the graphics window while any mouse movement with or without the button being down will cause Processing to call mouseMoved( ). It will probably be better to use an intentional mouse movement signaled by the user by pressing the mouse button. Otherwise, any random mouse movement can affect the speed.

So we are going to write this:

```
 void mouseDragged( )
 {

 }
```

But how are we going to use this?

Here is one strategy:
If the mouse is dragged from left to right, we speed up the movement. If the mouse is dragged from right to left , we slow down the movement. Might work.

To do this, we how do we figure out which direction the mouse is being dragged; we have to know where the mouse was in the previous frame so we can compare that location to where it is now. Processing has two variables that we can use: pmouesX and pmouseY. These variables have the location of the mouse in the previous frame. If we compare pmouseX to mouseX, we can determine which way the mouse is moving:

- if mouseX is bigger than pmouseX, movement is to the right so we have to increase the speed.
- if mouseX is smaller than pmouseX, movement is to the left so we have to decrease the speed.

This can be coded like this:

```
void mouseDragged( )
{
   if ( mouseX > pmouseX )
   {
      txSpeed = txSpeed + 1;
   }
   else if ( mouseX < pmouseX )
   {
      txSpeed = txSpeed - 1;
   }
```

Copyright Jim Roberts September 2012 Pittsburgh, PA 15221 All Rights Reserved

```
}
```

If we type this, it will work but we have created a new problem. Can you figure out what it is?

If we drag the mouse from right to left, we slow down the movement. If we do this several times we can stop the movement. We can not only stop the movement but we can start moving the target to the left. Great,!! But if this happens, the target will disappear off the left side of the window.

So we have to edit moveTarget( ) to wrap the target back to the right side when this happens.
Here is moveTarget( ) as we coded it:

```
void moveTarget( )
{
    tx = tx + txSpeed; // target too far right
    if (tx > width )
    {
       tx = 0;  // move it back to the left side
    }

}
```

What do we have to add to th code to keep the taget from moving off the left side of the window?

```
void moveTarget( )
{
    tx = tx + txSpeed;
    if (tx > width )  // target to far right
    {
       tx = 0;  // move it back to the left side
    }
     ? ? ?
}
```

We leave that and the vertical control of movement to you…

Easing

In previous classes and assignments we have used the mouse and variables to locate positions exactly. When we have moved a figure, the figure instantly reached its moving speed and instantly stopped when it got to its final destination. There was no acceleration at the beginning and deceleration at the end. If we want our figures to move “more naturally” we need to try to code some acceleration/deceleration into our code.

One strategy to do this is called easing. John will cover this in class. Jim will write up the notes after his presentation and put John’s code on line.

Copyright Jim Roberts September 2012 Pittsburgh, PA 15221 All Rights Reserved

Shortcuts (a word from our sponsor)

Are you tired of typing all this stuff?
Would you like a few shortcuts?
The cost very little – in fact so little that we cannot charge for them.
Here are a few – take them out for a test drive.

The Long Way	The Shortcut
`x = x + 1;`	`x++;`
`x = x - 1;`	`x--;`
`x = x + xSpeed;`	`x += xSpeed;`
`x = x - xSpeed;`	`x -= xSpeed;`

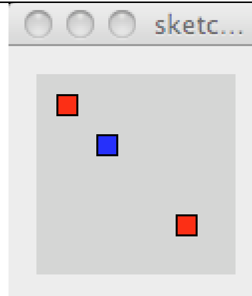
==

Everything is red but one rect which is blue.
Do I have to go back and add
fill(255, 0, 0);
everywhere I draw anything else?

We have another shortcut that you might like to try out. Let's assume that you have code that is drawing everything in red but one shape and it has to be blue. If you change the fill to blue, the fill is set to blue forever (which is defined as "until you set it back to red"). This could mean searching through lots of code to find where you might have to reset the fill color.

Here is some code that will let you temporarily set the fill to blue and then literally throw it away:

```
fill( 255, 0, 0);
rect( 10, 10, 10, 10 );
pushStyle( );
  fill( 0, 0, 255 );
  rect( 30, 30, 10, 10 );
popStyle( );
rect( 70, 70, 10, 10 );
```



The pushStyle( ) …popStyle( ) functions let you temporarily set a style like fill( ) and then throw it away returning your settings back to what they were before the pushStyle( ) function was called. Check the API for what else can be temporarily set between these two functions. It might save you some coding.

Copyright Jim Roberts September 2012 Pittsburgh, PA 15221 All Rights Reserved