**Keyboard and Mouse Interaction in Processing**

**Carson Foster**

Now we start to get to some of the more interesting material. In this lesson, we’ll be discussing how to interact with the user’s keyboard and mouse. This will allow you to make more complex, more intriguing, and more useful applications in Processing.

Mouse Interaction

We will start off by discussing the ways of monitoring the state of the mouse in the normal **draw** function. We haven’t really used this function, since our programs have been static images that don’t change. Recall that the **draw** function runs once every frame (and the default is 60 frames per second). This section is differentiated from events/listeners, which we will discuss in a later subsection. Let’s look at the different types of information we can extract from the mouse.

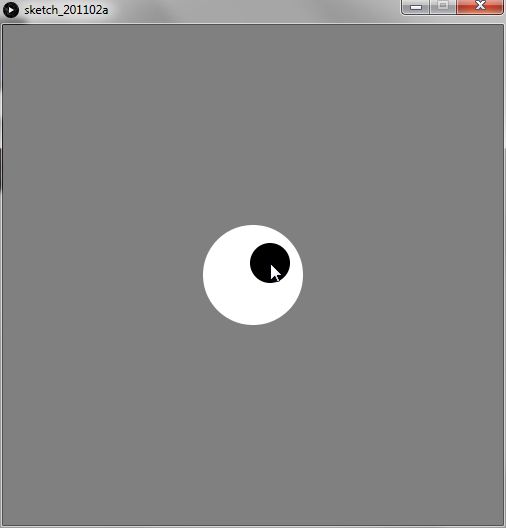
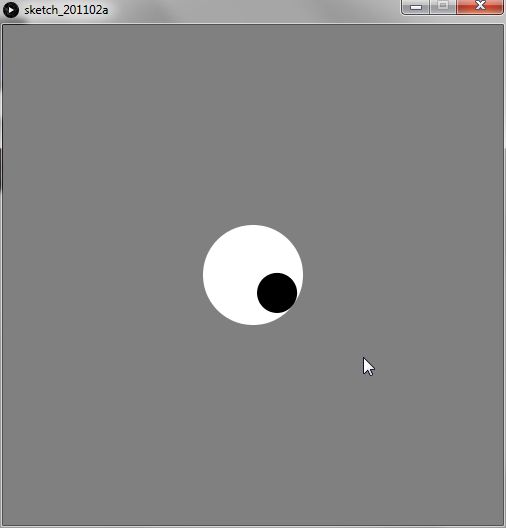
First, we can find the coordinates of the mouse cursor using two special variables:



These variables act like any other float variable, but change after each call to the **draw** function. The coordinate system they use is always the exact same as the default Processing coordinate scale, so they are not affected by transformations.

Next, we can also look at the coordinates of the mouse cursor on the previous frame, which allows us to make comparisons with the coordinates from this frame. These values are stored in the pmouseX and pmouseY variables.

Using just this information, we can make some interesting programs. For example, we could make an Eye class that simply draws an eye that looks wherever the cursor is.



Try this code out for yourself! The code is mostly

self-explanatory, but the atan2 function returns

the angle (in radians) for a given set of (y, x) coordinates: it acts as an inverse tangent function that is given the side lengths of the triangle instead of a value. Here are some example screenshots, but the code can be more fully appreciated when you run it yourself.

Next, we can also poll the mouse for the status of its buttons: whether a button is pressed, and if so, which button is pressed.



The mousePressed variable is a Boolean, so it can either hold true or false. The mouseButton variable can hold one of three values: left, center, or right. The center mouse button is normally pressing down on the scroll wheel, by the way. The mousePressed variable is true only as long as a mouse button is depressed – so it’s false as soon as the button is released. However, mouseButton keeps its value until another button is pressed. So, when, for example the left mouse button is pressed, mouseButton is left even after the mouse button is released, until, for example, the right mouse button is pressed.



This is a simple example, where the color of the rectangle in the center changes based on what mouse button is currently depressed. This one isn’t super exciting, but try it out for yourself!

Keyboard Interaction

From the keyboard we can tell whether a key is pressed, and what that key is. This is similar to mouse interaction, since we’ll just being variables in the draw function to check the status of the keyboard. Remember, we’ll discuss another way of dealing with the keyboard and mouse soon.

So, the variables that we use to interact with the keyboard are the following:



The keyPressed variable works the same way as the mousePressed variable does: it’s a Boolean that is true when any key is depressed, and then false the second all keys are released.

When a key with an ASCII code is pressed, the corresponding character value is stored in the key variable. This means that key is a variable of type char.

When a key that is ‘coded’ is pressed, then we have to look at the keyCoded variable to get its value. You’ll know if a coded key is pressed because key will be equal to the constant coded. The following are the most common values of keyCoded (you can find more through research):



Notice that keys like the enter key or the backspace key are not listed. These keys have ASCII values, and can therefore be tested for using the plain key variable.

A small example of a bouncing ball, which you can control using the arrow keys, is shown on the next page. I encourage you to write a similar program and to explore some of the possibilities now that we can interact with users.

Events

Instead of using variables in the draw function, we can instead use special functions to interact with the user. These functions are called once when a certain input event occurs, at the end of the current draw cycle. In general terms, we call these types of functions event listeners, because they wait and ‘listen’ until a certain event happens. When that event happens, then they are called. The various listeners that are available are as follows:





The mousePressed listener is called only once when a mouse button is pressed down. This differs from how the mousePressed variable works, which is true while a button is pressed down. The mouseReleased listener is called only once when a mouse button is released.

The mouseMoved listener is self-explanatory: it is called once when the mouse is moved. The mouseDragged function is called once when the mouse is moved and a mouse button is pressed down.

The keyPressed and keyReleased functions work the same as mousePressed and mouseReleased, i.e. they are called once when a key is pressed down the first time, or when a key is released (respectively).

This just provides another way of processing interaction, and it is completely up to personal preference. Some might prefer the abstraction of events with event listeners and other might prefer manual input processing.