## [Pygame tutorial #3: mouse events](http://lorenzod8n.wordpress.com/2007/05/30/pygame-tutorial-3-mouse-events/)

In the last tutorial, you learned how to draw lines. This time we will deal with mouse events. As usual we will keep it simple.

Already in the first tutorial you learned about the QUIT event. Now let’s add a little bit of code to track the position of the mouse on our window. This is done by the MOUSEMOTIONevent. The code looks as follows:

 1 #! /usr/bin/env python

 2

 3 import pygame

 4

 5 x = y = 0

 6 running = 1

 7 screen = pygame.display.set\_mode((640, 400))

 8

 9 **while** running:

10 event = pygame.event.poll()

11 **if** event.type == pygame.QUIT:

12 running = 0

13 **elif** event.type == pygame.MOUSEMOTION:

14 **print** ("mouse at (%d, %d)" % event.pos )

15

16 screen.fill((0, 0, 0))

17 pygame.display.flip()

Most of the code should look familiar by now. What is new is the check to see if we have an event of the type MOUSEMOTION (line 13). Also notice that if the event is a mouse motion event we can get a little bit more information out of the Event object. In the next line (14), we use call event.pos to get the current coordinates of the mouse pointer. This method returns a pair of values representing the x-position and y-position of the mouse pointer.

Note that the values returned by event.pos are relative to the top left-hand corner of the window, not the entire screen (unless, of course, the window covers the entire screen). You probably already know that the top-left hand corner is (0, 0) in screen coordinates.

**Exercises**

1. Write a program that prints the position of the mouse pointer using Cartesian coordinates. To begin with assume that the x-axis is at half the height of the window and the y-axis is located at half the width of the window. You already know how to draw lines, so you might as well draw the axes. Take care with the difference in direction along the y-axis between screen coordinates and Cartesian coordinates.
2. Replace MOUSEMOTION in the example above with MOUSEBUTTONDOWN. Run the program. Move the mouse over the window and press any mouse button. What happens?
3. Imagine that your window consists of tiles, each one 32×32 pixels. Write a program that detects the current mouse position, translates the screen coordinates into tile coordinates and prints this.
4. Write a program that calculates the distance of the mouse pointer from the centre of the window. Remember that there is an imaginary straight line from the centre point to the point where the mouse pointer is located. Just calculate the length of the straight line.

Seeing screen coordinates dumped onto you terminal window is exciting for just about 0.02 seconds. Let’s do something else. Since what we have learned so far is drawing lines we will stick to that. Here is a program that draws lines that cut the mouse pointer’s coordinates.

 1 #! /usr/bin/env python

 2

 3 import pygame

 4

 5 bgcolor = 0, 0, 0

 6 linecolor = 255, 255, 255

 7 x = y = 0

 8 running = 1

 9 screen = pygame.display.set\_mode((640, 400))

10

11 **while** running:

12 event = pygame.event.poll()

13 **if** event.type == pygame.QUIT:

14 running = 0

15 **elif** event.type == pygame.MOUSEMOTION:

16 x, y = event.pos

17

18 screen.fill(bgcolor)

19 pygame.draw.line(screen, linecolor, (x, 0), (x, 399))

20 pygame.draw.line(screen, linecolor, (0, y), (639, y))

21 pygame.display.flip()

There is really nothing new in this code so I don’t think you will have any difficult understanding it. We have already learned how to set the color of the line we are drawing, so let’s extend the program slightly:

 1 #! /usr/bin/env python

 2

 3 import pygame

 4

 5 bgcolor = 0, 0, 0

 6 blueval = 0

 7 bluedir = 1

 8 x = y = 0

 9 running = 1

10 screen = pygame.display.set\_mode((640, 400))

11

12 **while** running:

13 event = pygame.event.poll()

14 **if** event.type == pygame.QUIT:

15 running = 0

16 **elif** event.type == pygame.MOUSEMOTION:

17 x, y = event.pos

18

19 screen.fill(bgcolor)

20 pygame.draw.line(screen, (0, 0, blueval), (x, 0), (x, 399))

21 pygame.draw.line(screen, (0, 0, blueval), (0, y), (639, y))

22 blueval += bluedir

23 **if** blueval == 255 **or** blueval == 0: bluedir \*= -1

24 pygame.display.flip()

**Exercises**

1. In the examples above, the program draws lines that extend the full width and the height of the window. Create a program that draws a much smaller ‘+’, say from 10 pixels to the left/top of the mouse pointer to 10 pixels to the right/below the pointer.
2. Write a program that draws a horizontal line at the y-coordinate of the mouse pointer. The color of the line should vary according to the following: divide the window into four quadrants. Check which of the quadrants the pointer is in and set the line color to red if it is in the first quadrant, green if it is in the second quadrant, blue if in the third quadrant or white if the mouse pointer is in the last quadrant.
3. Write a program that tracks the positon of the mouse pointer and draws one line from the bottom right right-hand corner of the window to the current mouse position as well as one line from the bottom left-hand corner of the window to the position of the mouse pointer.

**Mouse buttons**

If you have been doing the exercises so far then you already know how to deal with a mouse button being pressed. Let’s look at an example.

 1 import pygame

 2

 3 LEFT = 1

 4

 5 running = 1

 6 screen = pygame.display.set\_mode((320, 200))

 7

 8 **while** running:

 9 event = pygame.event.poll()

10 **if** event.type == pygame.QUIT:

11 running = 0

12 **elif** event.type == pygame.MOUSEBUTTONDOWN **and** event.button == LEFT:

13 **print** ("You pressed the left mouse button at (%d, %d)" % event.pos)

14 **elif** event.type == pygame.MOUSEBUTTONUP **and** event.button == LEFT:

15 **print (** "You released the left mouse button at (%d, %d)" % event.pos)

16

17 screen.fill((0, 0, 0))

18 pygame.display.flip()

In this example we handle the two events MOUSEBUTTONDOWN and MOUSEBUTTONUP. Run the program and try this. Press the left mouse button down. While holding the button down, move it to a different position and then release the mouse button. Good, now you know how that works.

**Exercises**

1. Write a program that draws a horizontal line and a vertical line that both intersect the position of the mouse pointer. The color of the lines should change every time the left button is clicked. The vertical line should be set to a new random color. After generating a random color value, make sure that it is different from the current color. The horizontal line should be set to the previous color of the vertical line.
2. Write a program that draws a ‘+’ surrounding the mouse pointer when it is pressed down. The size of the cross should be 20 pixels in each direction. When the mouse button is released the cross should disappear.

**Conclusion**

In this tutorial you have learned how to deal with mouse motion and mouse button state. I think you are now comfortable enough with what events that we can look into them into some detail. Up to now we have been using poll() to get an event off the event queue, for the sake of simplicity. In the next tutorial you will see why that is not really a good idea and what you should do instead. Don’t worry; it won’t really become more complicated.

We will also learn a few more drawing methods.