## Pygame tutorial #6: from pixel to worm

In the previous tutorial we learned how to plot individual pixels. We wrote a simple program where you move a pixel around the screen. Today we are going to turn our single pixel into a worm.

Two classic games that could easily be built from what we will look at today are “the worm game” and Tron. In this tutorial we will learn how to create the worm itself, and in the next tutorial we will turn it all into a little game that we can actually play.

**A first try**

Our worm will move across the screen. We can control it with the cursor keys. There are two things are worm must avoid at all costs: the screen borders and itself.

The following code implements just that.

  4 # Move a worm across the screen. Beware of borders and self!
  5
  6 import pygame
  7
  8 **class** Worm:
  9  """ A worm. """
10 **def** \_\_init\_\_(self, surface, x, y, length):
11         self.surface = surface
12         self.x = x
13         self.y = y
14         self.length = length
15         self.dir\_x = 0
16         self.dir\_y = -1
17         self.body = []
18         self.crashed = False
19
20     **def** key\_event(self, event):
21         """ Handle key events that affect the worm. """
22         **if** event.key == pygame.K\_UP:
23             self.dir\_x = 0
24             self.dir\_y = -1
25         **elif** event.key == pygame.K\_DOWN:
26             self.dir\_x = 0
27             self.dir\_y = 1
28         **elif** event.key == pygame.K\_LEFT:
29             self.dir\_x = -1
30             self.dir\_y = 0
31         **elif** event.key == pygame.K\_RIGHT:
32             self.dir\_x = 1
33             self.dir\_y = 0
34
35     **def** move(self):
36         """ Move the worm. """
37         self.x += self.dir\_x
38         self.y += self.dir\_y
39
40         **if** (self.x, self.y) **in** self.body:
41             self.crashed = True
42
43         self.body.insert(0, (self.x, self.y))
44
45         **if** len(self.body) > self.length:
46             self.body.pop()
47
48     **def** draw(self):
49         **for** x, y **in** self.body:
50             self.surface.set\_at((x, y), (255, 255, 255))
51
52 # Dimensions.
53 width = 640
54 height = 400
55
56 screen = pygame.display.set\_mode((width, height))
57 clock = pygame.time.Clock()
58 running = True
59
60 # Our worm.
61 w = Worm(screen, int(width/2), int(height/2), 200)
62
63 **while** running:
64     screen.fill((0, 0, 0))
65     w.move()
66     w.draw()
67
68     **if** w.crashed **or** w.x <= 0 **or** w.x >= width-1 **or** w.y <= 0 **or** w.y >= height-1:
69         **print** ("Crash!")
70         running = False
71
72     **for** event **in** pygame.event.get():
73         **if** event.type == pygame.QUIT:
74             running = False
75         **elif** event.type == pygame.KEYDOWN:
76             w.key\_event(event)
77
78     pygame.display.flip()
79     clock.tick(240)

The first thing we do a define what a worm is. Notice how the functionality of the worm is contained within the class. The worm knows how to move itself, draw itself, and it handles the events that relate to it. (Note: if you have done Java programming before, you may wonder where your accessors and mutators are. Don’t worry about them. You don’t need them).

The body of the worm is stored in a list. Each time we move one step, we insert the new position of the body into the beginning of the list and pop off the last list item. That keeps our worm constant in length. Actually, that is not entirely true. Our worm starts off being only a single pixel long. We have a bit of extra code that allows the worm to grow to its full length. A little bit of code analysis should make that clear.

The move() method also contains a check to see if the worm has crashed onto itself. What we do is see if the position of the worm’s head (self.x, self.y) happens to be at the same place as any part of the worm’s body, by comparing (self.x, self.y) with each position in the body list.

The draw() method simply plots all the points held in the body list. The main program should be fairly straight-forward by now. Notice that in addition to checking if the worm has crashed onto any of the borders, the worm object is also asked if it has crashed with itself.

**Exercises**

1. Let the worm grow over time. You could for instance have a counter that decreases every frame, or every time the worm turns or whatever. When the counter reaches zero, you increase the length of the worm.
2. Create a program that displays two worms. What changes do you need to make to the Worm class in order for this to work?
3. Modify the worm to have a gradient effect. The head of the mouse should be white. Each body segment should gradually become darker and darker shades of gray.

**Collision detection**

In our first example, we checked if the worm had collided with itself by comparing the position of the worm’s head with all the positions of the body. That worked fine, but there are instances where it’s not practical to keep a lot of positions in a list. An example would be the Tron game that I referred to earlier. The more you move, the longer the list would be, and the longer it would take to determine whether or the object has collided with its own body.

We will try a slightly different approach, by using Surface.get\_at, which tells us the color of the pixel at the given position. It actually returns (red, green, blue, alpha), but we won’t worry about the alpha part just yet.

The new code is below:

 3 # Move a worm across the screen. Beware of borders and self!
 4
 5 import pygame
 6
 7 **class** Worm:
 8     **def** \_\_init\_\_(self, surface, x, y, length):
 9         self.surface = surface
10         self.x = x
11         self.y = y
12         self.length = length
13         self.dir\_x = 0
14         self.dir\_y = -1
15         self.body = []
16         self.crashed = False
17
18     **def** key\_event(self, event):
19         """ Handle keyboard events that affect the worm. """
20         **if** event.key == pygame.K\_UP:
21             self.dir\_x = 0
22             self.dir\_y = -1
23         **elif** event.key == pygame.K\_DOWN:
24             self.dir\_x = 0
25             self.dir\_y = 1
26         **elif** event.key == pygame.K\_LEFT:
27             self.dir\_x = -1
28             self.dir\_y = 0
29         **elif** event.key == pygame.K\_RIGHT:
30             self.dir\_x = 1
31             self.dir\_y = 0
32
33     **def** move(self):
34         """ Move the mouse. """
35         self.x += self.dir\_x
36         self.y += self.dir\_y
37
38         r, g, b, a = self.surface.get\_at((self.x, self.y))
39         **if** (r, g, b) != (0, 0, 0):
40             self.crashed = True
41
42         self.body.insert(0, (self.x, self.y))
43
44         **if** len(self.body) > self.length:
45             self.body.pop()
46
47     **def** draw(self):
48         """ Draw the worm. """
49         **for** x, y **in** self.body:
50             self.surface.set\_at((x, y), (255, 255, 255))
51
52 # Window dimensions.
53 width = 640
54 height = 400
55
56 screen = pygame.display.set\_mode((width, height))
57 clock = pygame.time.Clock()
58 running = True
59
60 # Our worm.
61 w = Worm(screen, int(width/2), int(height/2), 200)
62
63 **while** running:
64     screen.fill((0, 0, 0))
65     w.draw()
66     w.move()
67
68     **if** w.crashed **or** w.x <= 0 **or** w.x >= width-1 **or** w.y <= 0 **or** w.y >= height-1:
69         **print (** "Crash!!!")
70         running = False
71
72     **for** event **in** pygame.event.get():
73         **if** event.type == pygame.QUIT:
74             running = False
75         **elif** event.type == pygame.KEYDOWN:
76             w.key\_event(event)
77
78     pygame.display.flip()
79     clock.tick(240)

As you can see, the new version is similar to the previous one for most part. Where it does differ is in the part where we detect whether or not the worm has collided with itself. We no longer need to compare the position of the head with each position of the body. Instead we simply compare the color of the pixel where the head is with the color of the background. If they don’t match then we have a collision, since the worm is the only object on the screen.

There is one other difference between the programs. In the event loop, look at the order in which the w.draw() and w.move() are in each program. Notice the difference? In order for the second program to be able to work at all, it is vital that we draw that worm before we move it. Think about it for a while and you will understand why.

**Exercises**

1. Change the program to compare the pixel color at the worm’s head with the worm’s body color instead of the background color. If the color of the pixel is the same as that of the body it is a crash.
2. Add obstacles around the screen. These should be randomly positioned pixels. Use a different color than the worm’s color. Add collision detection to determine if the worm has crashed onto one of the obstacles. If so, the program should exit.
3. Give the worm food to eat. Food should be a pixel randomly positioned on the screen. Make it a different color from both obstacles and the worm itself. Any time the worm gets the food, the message “yummy” should be printed to the console and a new food should be generated. Keep a counter that increases every time the worm grabs a food item.

**Optimizing**

As a game developer, one thing you will have to get used to is spending hours with your code looking for places to optimize. What I’m going to do here is very trivial, but then again, we are working with a very trivial example.

One thing that we do that is really wasteful is “clear” the surface each frame. This wouldn’t be necessary if we were to instead plot a single pixel, the color of the background, at the very tail of the worm. The following program does that:

  4 # Move a worm around the screen. Beware of borders and self!
  5
  6 import pygame
  7
  8 **class** Worm:
  9     """ A worm. """
10     **def** \_\_init\_\_(self, surface, x, y, length):
11         self.surface = surface
12         self.x = x
13         self.y = y
14         self.dx = 0
15         self.dy = -1
16         self.length = length
17         self.body = []
18         self.last = (0, 0)
19         self.crashed = False
20
21     **def** key\_event(self, event):
22         """ Handle keyboard events that affect the worm. """
23         **if** event.key == pygame.K\_UP:
24             self.dx = 0
25             self.dy = -1
26         **elif** event.key == pygame.K\_DOWN:
27             self.dx = 0
28             self.dy = 1
29         **elif** event.key == pygame.K\_LEFT:
30             self.dx = -1
31             self.dy = 0
32         **elif** event.key == pygame.K\_RIGHT:
33             self.dx = 1
34             self.dy = 0
35
36     **def** move(self):
37         """ Move the mouse. """
38         self.body.insert(0, (self.x, self.y))
39         self.x += self.dx
40         self.y += self.dy
41
42         r, g, b, a = self.surface.get\_at((self.x, self.y))
43
44         **if** (r, g, b) != (0, 0, 0):
45             self.crashed = True
46
47         **if** len(self.body) >= self.length:
48             self.last = self.body.pop()
49         **else**:
50             self.last = self.body[-1]
51
52     **def** draw(self):
53         """ Draw the worm """
54         self.surface.set\_at((self.x, self.y), (255, 255, 255))
55         self.surface.set\_at(self.last, (0, 0, 0))
56
57 # Window dimensions.
58 width = 640
59 height = 400
60
61 screen = pygame.display.set\_mode((width, height))
62 screen.fill((0, 0, 0))
63 pygame.display.flip()
64
65 clock = pygame.time.Clock()
66 running = True
67
68 # Our worm.
69 w = Worm(screen, int(width/2), int(height/2), 200)
70
71 **while** running:
72     w.draw()
73     w.move()
74
75     **if** w.crashed **or** w.x <= 0 **or** w.x >= width-1 **or** w.y <= 0 **or** w.y >= height-1:
76         **print (** "Crashed!")
77         running = False
78
79     **for** event **in** pygame.event.get():
80         **if** event.type == pygame.QUIT:
81             running = False
82         **elif** event.type == pygame.KEYDOWN:
83             w.key\_event(event)
84
85     pygame.display.flip()
86     clock.tick(240)

Study the move() and draw() methods and compare them to the ones in the previous version. Also notice that there is no longer any screen.fill() call inside the event loop.

I have rearranged some of the code in the move() method. Is this necessary? Figure it out. You can always try rearranging lines to see what happens. You will learn much by experimenting with the code.

**Exercises**

1. Analyze the last version of the worm program. How does it differ from the previous one? How does it work? In what ways can this said to be “optimized” as compared to the previous versions?
2. Extend the worm program that draws food for the worm to eat in such a way that each time the worm eats food it grows a little longer.
3. Extend the program in the previous exercise so that there is “poison mushroom” as well as food on the screen. If the worm eats one of these, it reduces in length. If the length goes below a certain minimum, the program should end.

**Conclusion**

That brings to end another pygame tutorial. I hope you enjoyed it. I know that I am moving way too slow for some people. There are loads of other good tutorials out there if you are in a hurry to learn. Many (all?) of them are far superior to my own.

Please do try and solve at least some of the exercises. If you want to send me your solutions or questions, you can find my email if you look around a bit. 