## Pygame tutorial #9: first improvements to the game

In the last pygame tutorial we wrote a functional, but not very interesting worm game. In this tutorial and the next I will be making various small improvements to the game. After that, I will finally leave the worm game and move on to other things. There will still be improvements to make, but those will be left as an exercise for you.

**Bigger food**

One problem with the game that we developed last time was that the game is insanely difficult unless your eyesight happens to be really good (which mine is not). Let’s start by sizing up the food a little. I will use a method called pygame.draw.rect which (surprise, surprise) draws a rectangle (and squares if width and height are equal). We modify the draw method of the Food class:

1 **def** draw(self):
2     pygame.draw.rect(self.surface, self.color, (self.x, self.y, 3, 3), 0)
3

This draws a square at the point (x, y). The width and the height are both 3 pixels. The final zero is the line width. A line width of zero draws a filled rectangle, which is what we want here.

To check if the worm has eaten the food we add a new method to Food, called check:

1 **def** check(self, x, y):
2     **if** x < self.x **or** x > self.x + 3:
3         **return** False
4     **elif** y < self.y **or** y > self.y + 3:
5         **return** False
6     **else**:
7         **return** True
8

This will check if the point (x, y) is a point within the food square. The method will return True if this is the case, or False if it isn’t.

Finally, in the game loop we need to change the test for if the worm has eaten the food. Instead of using the location() calls, we utilize the new check() method:

1 **elif** food.check(worm.x, worm.y):
2     score += 1
3     worm.eat()
4     **print** ("Score: %d" % score)
5     food = Food(screen)
6

**Improving the keyboard handler**

Another thing you will have noticed while playing is that it is far too easy to crash into yourself by “reversing” you direction. For example, if the worm is moving upwards, pressing down will cause the worm to crash onto itself. Let’s prevent that from happening:

 1 **def** event(self, event):
 2     """ Handle keyboard events. """
 3     **if** event.key == pygame.K\_UP:
 4         **if** self.vy == 1: **return**
 5         self.vx = 0
 6         self.vy = -1
 7     **elif** event.key == pygame.K\_DOWN:
 8         **if** self.vy == -1: **return**
 9         self.vx = 0
10         self.vy = 1
11     **elif** event.key == pygame.K\_LEFT:
12         **if** self.vx == 1: **return**
13         self.vx = -1
14         self.vy = 0
15     **elif** event.key == pygame.K\_RIGHT:
16         **if** self.vx == -1: **return**
17         self.vx = 1
18         self.vy = 0
19

For each key, there is one extra test here. This should not be difficult to understand.

**A little optimization**

Our little game could do with some optimization. The call to screen.fill() each time we run the loop slows the game down quite a bit. Also, since we are redrawing entire worm all the game should get just a little bit slower each time the worm’s length increases.

But why do we draw the entire worm? If we skip clearing the screen all the time, there are only to points we should draw: the first one and the last one. The last point of the worm should be plotted with the color of the background. We can change the draw() method in the worm class like this:

1 **def** draw(self):
2     #for x, y in self.body:
3     #    self.surface.set\_at((x, y), self.color)
4     x, y = self.body[0]
5     self.surface.set\_at((x, y), self.color)
6     x, y = self.body[-1]
7     self.surface.set\_at((x, y), (0, 0, 0))

This requires us to comment out the screen.fill() call in the game loop. But we also need to be able to erase the food once it has been eaten. We add an erase() method to Food. This looks similar to the draw() method, but uses a different color:

1 **def** erase(self):
2     pygame.draw.rect(self.surface, (0, 0, 0), (self.x, self.y, 3, 3), 0)
3

Of course, we have to remember to call the erase() method each time the worm eats the food.

**Adding sound**

Let’s make the worm make a chomping sound any time it eats the food. To do this we first have to initialize the mixer and load a sample. You can search on the ‘Net for a sound file, or if you have a microphone, record your own, using software like sox. I recorded myself saying “GULP!” and saved the file as *chomp.wav* in the same directory as the worm game. It’s a good idea to make sure that the sound is short.

 1 pygame.mixer.init()
 2 chomp = pygame.mixer.Sound("chomp.wav")

Once we have done this, the variable chomp will hold a Sound object that we can play when we want to. In our case that is any time the worm eats the food.

**The improved game**

Here is the full code for the game as it is now:

 1 # A simple worm game, 2nd attempt.
   2
   3 import pygame, random
   7
   8 **class** Worm:
   9     **def** \_\_init\_\_(self, surface):
 10         self.surface = surface
 11         self.x = surface.get\_width() / 2
 12         self.y = surface.get\_height() / 2
 13         self.length = 1
 14         self.grow\_to = 50
 15         self.vx = 0
 16         self.vy = -1
 17         self.body = []
 18         self.crashed = False
 19         self.color = 255, 255, 0
 20
 21     **def** eat(self):
 22         self.grow\_to += 25
 23
 24     **def** event(self, event):
 25         """ Handle keyboard events. """
 26         **if** event.key == pygame.K\_UP:
 27             **if** self.vy == 1: **return**
 28             self.vx = 0
 29             self.vy = -1
 30         **elif** event.key == pygame.K\_DOWN:
 31             **if** self.vy == -1: **return**
 32             self.vx = 0
 33             self.vy = 1
 34         **elif** event.key == pygame.K\_LEFT:
 35             **if** self.vx == 1: **return**
 36             self.vx = -1
 37             self.vy = 0
 38         **elif** event.key == pygame.K\_RIGHT:
 39             **if** self.vx == -1: **return**
 40             self.vx = 1
 41             self.vy = 0
 42
 43     **def** move(self):
 44         """ Move the worm. """
 45         self.x += self.vx
 46         self.y += self.vy
 47
 48         **if** (self.x, self.y) **in** self.body:
 49             self.crashed = True
 50
 51         self.body.insert(0, (self.x, self.y))
 52
 53         **if** (self.grow\_to > self.length):
 54             self.length += 1
 55
 56         **if** len(self.body) > self.length:
 57             self.body.pop()
 58
 59     **def** draw(self):
 60         #for x, y in self.body:
 61         #    self.surface.set\_at((x, y), self.color)
 62         x, y = self.body[0]
 63         self.surface.set\_at((int(x), int(y)), self.color)
 64         x, y = self.body[-1]
 65         self.surface.set\_at((int(x), int(y)), (0, 0, 0))
 66
 68 **class** Food:
 69     **def** \_\_init\_\_(self, surface):
 70         self.surface = surface
 71         self.x = random.randint(0, surface.get\_width())
 72         self.y = random.randint(0, surface.get\_height())
 73         self.color = 255, 255, 255
 74
 75     **def** draw(self):
 76         pygame.draw.rect(self.surface, self.color, (self.x, self.y, 3, 3), 0)
 77
 78     **def** erase(self):
 79         pygame.draw.rect(self.surface, (0, 0, 0), (self.x, self.y, 3, 3), 0)
 80
 81     **def** check(self, x, y):
 82         **if** x < self.x **or** x > self.x + 3:
 83             **return** False
 84         **elif** y < self.y **or** y > self.y + 3:
 85             **return** False
 86         **else**:
 87             **return** True
 88
 89 w = 500
 90 h = 500
 91
 92 screen = pygame.display.set\_mode((w, h))
 93 clock = pygame.time.Clock()
 94
 95 pygame.mixer.init()
 96 chomp = pygame.mixer.Sound("chomp.wav")
 97
 98 score = 0
 99 worm = Worm(screen)
100 food = Food(screen)
101 running = True
102
103 **while** running:
104     #screen.fill((0, 0, 0))
105     worm.move()
106     worm.draw()
107     food.draw()
108
109     **if** worm.crashed:
110         running = False
111     **elif** worm.x <= 0 **or** worm.x >= w - 1:
112         running = False
113     **elif** worm.y <= 0 **or** worm.y >= h - 1:
114         running = False
115     **elif** food.check(worm.x, worm.y):
116         score += 1
117         worm.eat()
118         chomp.play()
119         **print (** "Score: %d" % score)
120         food.erase()
121         food = Food(screen)
122
123     **for** event **in** pygame.event.get():
124         **if** event.type == pygame.QUIT:
125             running = False
126         **elif** event.type == pygame.KEYDOWN:
127             worm.event(event)
128
129     pygame.display.flip()
130     clock.tick(240)

**Exercises**

1. If you play the game for a while you will notice that from time to time the food gets placed in a location where you cannot reach it. This happens when the food is placed at the right side or the bottom of the game screen. Why is that? What modification do you have to make to the Food constructor to prevent this from happening?

2. Modify the keyboard handler to only use the left and right keys. Pressing right should change the mouse direction clock-wise. Left will change the direction counter-clockwise.

3. Give the game a different background than black.

4. Notice how Food.erase() can result in the worm being “cut”. Change the game so that any time the food is eaten, after the call to Food.erase() the entire worm gets redrawn. You may want to have add an optional flag argument to Worm.draw(). If the flag is true, the entire worm is redrawn. Otherwise only the first and last pixels are drawn.

5. Add a sound when the worm crashes.

6. We increased the size of the food by drawing rectangles instead of pixels. Could the same be applied to the worm? Experiment with this and see what the issues are, if any.