Adding Baddie Score

I’m going to add tracking the number of baddies destroyed, and displaying the score to my game.

Tracking Game-wide Data

The SpaceshipData class tracks all sorts of game data. It updates that data, and displays the relevant information in the display. For example, the list of active bullets is managed and displayed by SpaceshipData. Also the collision between bullets and baddies is handled here.

Creating the **kills** Variable

We’ll add a variable to SpaceshipData that is used to count the number of destroyed baddies. At the beginning of the game, this will have the value 0, since no baddies have been killed yet.

In SpaceshipData.py in the **__init__** method, we will add this line to create the variable and set its value to 0.

```python
self.kills = 0
```

Updating the **kills** Variable

We want to add 1 to the kills variable every time a baddie is destroyed by a bullet. We do this in the `evolve` method of SpaceshipData. Midway through the method, you can find the line of code where a baddie is killed. Look for the line that says `baddie.setAlive(False)`. Immediately following this line, we will add this:

```python
self.kills = self.kills + 1
```

This will make sure we add to the number of kills, but only when a live baddie is destroyed by a bullet.

Displaying the Score

We want the user to see the number of kills as a score. We will draw text to the screen of the form “Score: 999”. We choose to put the score in the top left corner of the screen. Since our background is black, we’ll draw the score in white.

Configuring the Score Display

Back in the **__init__** method, we’ll configure the color and position of the score. We add these 3 lines.

```python
self.score_color = (255, 255, 255)
score.x = 10
score.y = 30
```

Displaying the Score Text

Now we are ready to display the score. We go to the **draw** method in SpaceshipData. We add two lines of code. One to create the text we want to display, and one to display the text.

```python
score_str = "Score: " + str(self.kills)
score.drawTextLeft(surface, score_str, self.score_color, self.x, self.y, self.font)
```

Notice the use of the configuration parameters we set in **__init__**. If we are unhappy with the display, we can change the parameters and look at it again.

Finer Points

Colors

Colors in pygame are specified by three channels. One for each of red (R), green (G) and blue (B). Each of the channels is represented with a number from 0 to 255, with 0 meaning none, and 255 meaning all.

This is a fairly standard color representation scheme, and you can use any of a number of online color selection
applications I use http://www.colorpicker.com/, and I’m careful to grab the correct R, G, B numbers from the website.

**Fonts**

The font we use to draw the text is configured in the __init__ method as well. You can see that it is 20 pixels tall. That’s why we bring the initial position of the text 30 pixels from the top of the screen, and 10 pixels from the left of the screen.

If we wanted to right-justify the text rather than left justify, we can use drawTextRight instead of drawTextLeft.

**Learning More About Fonts in Pygame**

- Font Documentation