MPHYG001

The Boids

Simon Stiebellehner (ucabsti)

February 2017

1 Identified code smells

The following listing describes the identified code smells, what I did for improvement and in which git commit these changes are included.

```
commit: e6ac4bdce18b3765eabde1a866af5c3ba6a08d1d
smell: everything in one folder
improvement: separate folders/created appropriate folder structure
```

```
commit: e6ac4bdce18b3765eabde1a866af5c3ba6a08d1d
smell: confusing variable names
improvement: replaced variable names with self-explanatory names
```

```
commit: 15176a456c65eb91cee9d33ac2bbb9deca657309
smell: magic numbers
improvement: replaced magic numbers with variables
```

```
commit: 8241fef71e70eddcc99eb9adcba379316682ac73
smell: function too large; neighboring for-loops
improvement: break large function into smaller units; merged
    neighboring for-loops
```

```
commit: 638e351c37b990aaafa0a4e9c17f59db74cbc0fd
smell: no classes used; all code in one file; global variables;
    integers as iterators
improvement: separated code into classes/files; converted global
    variables to function arguments; iterating over iterators (
    objects)
```

```
commit: 638e351c37b990aaafa0a4e9c17f59db74cbc0fd
smell: set of arrays
improvement: array of structures (Boid objects)
```

```
commit: 7d4a75d0f369789b64bb3bbf5c69d8b9bb92b57f
smell: no augmented assignment statements used
improvement: implemented augmented assignment statements
```

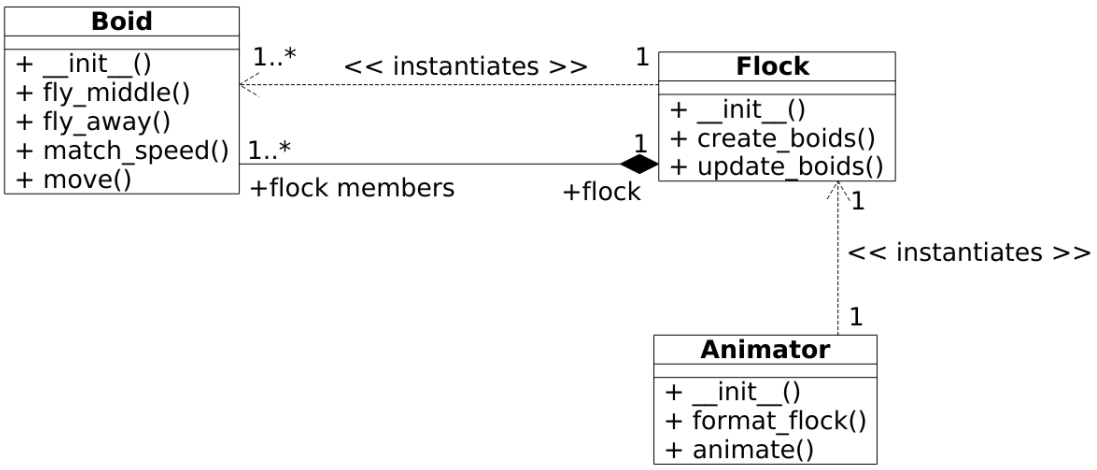
```
commit: 8da8ddefe45b5ef0f95106ae07564db063b20713
smell: overly complicated calculations
```

```
improvement: introduced support variables to ease understanding of
    calculations


commit: 30f735935e4741c58cdfc67258982623cc2e1ed6 and
        910e3b0ece5e0e67132725175a7d1231f94eee14
smell: no configuration file
improvement: added configuration file
```

2 UML diagram of final class structure

The following UML diagram illustrates the final class structure including variables on class level (none defined) and methods. For reason of simplicity and keeping the report tidy I did not include information on the input parameters and return values of the methods (some methods have more than 5 input parameters).



The architecture was motivated by the best-practise of separating displaying instance (class Animator), controlling instance (class Flock) and functionality implementing instance (class Boid). The user only interacts with the Animator class, which then creates a Flock object based on the input parameters specified by the user through the command line interface. The Flock object creates Boid objects and, for each frame of the animation, initiates change of position (movement) of each Boid object. Each Boid object then computes its new position (movement) based on its current position and the position of the other Boid objects, before passing its new position back to the Flock object. The Flock object then returns the updated positions of the Boid objects in an aggregated form to the Animator object, which then formats the position values appropriately and displays them in the plot, resulting in an animation.

3 Discussing advantages of refactoring to improving code

The refactoring approach aims to restructure existing program code while keeping the original functionality and behaviour of the program. Due to its step-by-step character refactoring facilitates small and safe changes of the code. However, at the same time this gradual character may also make it slow and time-consuming. Ideally, the result of refactoring is a cleaner, better readable, better understandable code. The improved structure increases modularity and extensibility of the code. This leads to better testability of the code. Moreover, although usually improving performance is not a principle goal of refactoring, it may be a positive byproduct. Overall, refactoring increases long-term efficiency and usability of the code.
Despite the various advantages of refactoring, naturally, there are also some disadvantages or risks associated with it. The step-by-step character does not only make it a slow and time-consuming process, but it also does not necessarily shield from unwanted alterations of the functionality/behaviour of the code. Furthermore, refactoring often requires re-writing of unit tests, which consumes even more time. Also, refactoring does not turn bad code into good code because it is not intended to change the core of the code itself. It rather converts bad code into "better readable, more modular and extensible suboptimal" code.
Overall, refactoring is a convenient and low-risk approach for improving some important properties of software code. However, its power is limited. Refactoring does not replace writing good code from the beginning.

4 Problems encountered during the project

The following subsections describe problems I encountered during the assignment.

4.1 Animation not working

From the very beginning of the assignment the animation was not working. Boids were shown in their initial position in the matplotlib graph, however, they did not move. Research on this problem suggested a plethora of possible reasons and solutions. After trying out a considerable number of these without success, I decided to reinstall Python and Anaconda, which eventually solved the problem.

4.2 Boids increasing speed / not keeping original flying behaviour

After merging some for-loops, I encountered the problem that each boid, i.e. the flock as a whole, was increasing velocity with the duration of the animation. I found out that the reason for this was that I changed the order of computation of the metrics that specify the boids' behaviour. In order to restore the original movement behaviour I had to re-implement the nested for-loops in a very similar way they were coded in the original code.
In general, ensuring that the movement pattern of the boids was not changed in the process of refactoring the code was difficult. Due to high accuracy of floating point variables, values for these variables change slightly when replacing calculations for equivalent ones. Also, since the movement of one boid depends on the position of the other boids, the order of calculating each boid's properties is crucial for the correct simulation of the flying behaviour.

I think that changing the for-loops to matrix operations to increase performance would have altered the flying behaviour considerably, as the sequentiality of computation of new positions would not have been in place anymore.

4.3 Testing the Animator class

Due to the fact that the Animator class, in essence, creates an animation writing tests for it was especially difficult. Since the animation itself cannot be assessed for its correctness, I decided to test everything that contributes to the creation of it, such as ensuring that the instantiation of the Animator class itself is performed as expected.

4.4 Including config file in package installation

Setuptools does not automatically include files, which are not of format .py or include certain keywords in the installation. Therefore, the configuration file config.yml was not included in installations of the package. However, due to the functionality of the package allowing users to use a provided config file by setting parameter '-c' to '1' in the command line interface, config.yml had to be included in the installation. It took me a significant amount of time until I eventually managed to have config.yml considered in the installation procedure by specifying the file in the MANIFESTO.in, and in setup.py under data_packages and package_data (for pip installation).