MPHYG001: Research Software Engineering with Python

Assessment 2: Refactoring the Bad Boids

Jan Zmazek, 16105492

February 23, 2017

1 Code smells

When refactoring the bad-boids code, I used the following refactoring techniques for the "code smells":

1. Replacing magic numbers with constants (`commit 41b477b`)

   Constants such as `x_coordinate_interval`, `x_velocity_interval` and `cohesion_limit` are added.

2. Replacing constants with configuration file (`commit 4f3bc0d`)

   Constants are added to a `config.yaml` file.

3. Using a library code instead of hand-written code (`commit 4f3bc0d`)

   Libraries such as `yaml` are used.

4. Changing bad variable names to sensible variable names (`commit fd673a4`)

   Variable names such as `xs, xvs` are replaced with `x_coordinate`, `x_velocity`

5. Replacing set of arrays with array of structures – dictionary (`commit fd673a4`)

   Boids object's "properties" property is replaced with a dictionary.

6. Merging neighbouring loops (`commit fd673a4`)

   Two neighbouring "for" loops are replaced with only one.

7. Replacing loops with iterators (`commit fd673a4`)

   `i in range(number_of_birds)` loops are replaced with `bird_i in birds`.

8. Editing code according to PEP8 standards (`commit 77407ba`)

   Appropriate spacing, variable names conventions, modules/classes/methods/functions descriptions.

9. Structuring code to modules, classes, methods and function, breaking a large functions into smaller units, replacing repeated code with a function (`commits bdaf796, e70ad0b, fd673a4, 2ce7635`)

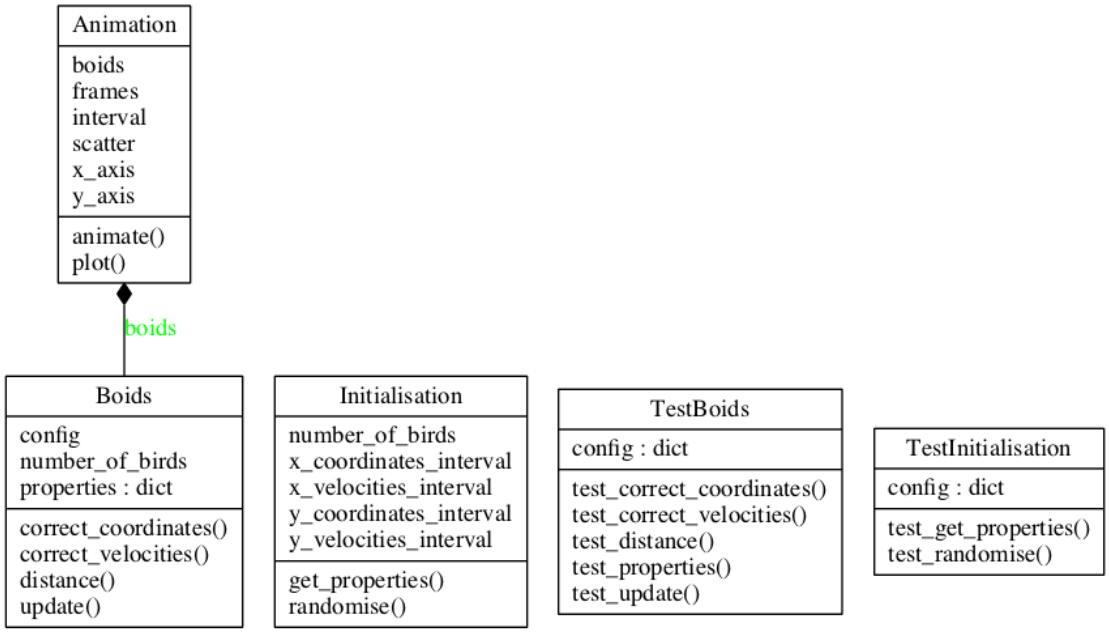
   Structuring code into classes `Initialisation`, `Boids`, `Animation` with methods and functions.

10. Testing the code (`commit 594cc50`)

    Writing unit tests for each method of the classes `Initialisation` and `Boids`.

2 UML diagram of the final class structure

The class UML diagram, generated with the automatic tool `pyreverse`, is shown on the image below:



3 Advantages of a refactoring approach to improving code

There are many advantages to a refactoring approach to improving the code. First of all, the code is after refactoring more readable, easier to understand and more customizable. Properly refactoring the code saves time as it is gradually and safely built, meaning that at each step the code must work. After refactoring, the code is often faster, as for example, one of the refactoring steps is using libraries, which are often heavily optimised.

I think refactoring is very time-consuming work that can be avoided by planning the structure of the code in advance and spending more time and resources on quality programming. However, in some cases, it is a logical and therefore useful step.

4 Problems encountered during the project

I spent quite a lot of time planning on which refactoring steps should do first and which last. Also, there are many more-or-less correct ways to breaking a code into smaller units, so that was another thing that I had trouble deciding on. During the refactoring, it was not easy for me to only do certain refactoring step in one commit. Other than that, I have not encountered any bigger problems.