MPHYG001 Bad-Boids Assignment

Simon Schaal, SN: 15072430

24.01.2016

1 Project layout

```
bad-boids/
|-- boids
|   |-- boids.py
|   |-- command.py
|   |-- __init__.py
|   `-- test
|       |-- fixtures
|       |   |-- avoidCollisions.yml
|       |   |-- fixture_np.yml
|       |   |-- fixture.yml
|       |   |-- flyToMiddle.yml
|       |   `-- matchSpeed.yml
|       |-- __init__.py
|       |-- record_fixture.py
|       `-- test_boids.py
|-- CITATION.md
|-- config.cfg
|-- LICENSE.md
|-- README.md
|-- scripts
|   `-- boids
`-- setup.py
```

2 Command line entry point

```
usage: boids [-h] [--config CONFIG] [--count COUNT]
             [--fly_to_middle_strength FLY_TO_MIDDLE_STRENGTH]
             [--alert_distance ALERT_DISTANCE]
             [--formation_flying_distance FORMATION_FLYING_DISTANCE]
             [--formation_flying_strength FORMATION_FLYING_STRENGTH]

Boid simulator: Simulates birds flying in flocks using the boids model.

optional arguments:
  -h, --help            show this help message and exit
  --config CONFIG, -c CONFIG
                        Boid simulator YAML configuration file. Defaults are
                        shown and used if not specified.
  --count COUNT         Overwrite bird count given in cfg file.
  --fly_to_middle_strength FLY_TO_MIDDLE_STRENGTH
                        Overwrite settings given in cfg file.
  --alert_distance ALERT_DISTANCE
                        Overwrite settings given in cfg file.
  --formation_flying_distance FORMATION_FLYING_DISTANCE
                        Overwrite settings given in cfg file.
  --formation_flying_strength FORMATION_FLYING_STRENGTH
                        Overwrite settings given in cfg file.
```

3 List of identified code smells

1. Create proper folder structure (c48826d485fe0571aef8fe0ed9546d1f16283cc8)
   - *Smell:* All files in a single folder.
   - *Solution*: Create proper folder structure for future boids module including a separate test folder. Additionally add license, citation, readme and setup.py file for future pip installation.
2. Introduce variables (1bf86bec697222644acb20930ac3a68a1e8d8a24)
   - *Smell*: Replace repeated magic numbers with variables.
   - *Solution*: Introduce a variable for boid count and position and velocity limits.
3. Introduce function to generate random positions / velocities (06ee6d7ee0eb5e304f64dc3c5492a3554abb8070)
   - *Smell*: Repeated code in position/velocity initialisation.
   - *Solution*: Introduce function gen_random(. . .) to initialise positions and velocities.
4. Add test for random position function (fe253768faeef2f3cbb241ae2c576b39c8b847ce)
   - *Smell*: So far there is only a single regression test for the complete boids model.
   - *Solution*: Provide a unit test for the new gen_random(. . .) function.
5. Replace python arrays with numpy arrays (83279f8d8ddb9d7e1487cd9ebbf1f1855d37fc4c)
   - *Smell*: Use more efficient library.
   - *Solution*: Gradually start switching to numpy arrays which are much faster and efficient in case of numeric matrix operations. Start by replacing the python arrays with numpy arrays in the initialisation.
6. Replace for-loops with numpy optimized calculations (8b2098b35e06c943ed3511865e4e18115039dac9)
   - *Smell*: Nested loops iterating over arrays.
   - *Solution*: Optimize code for numpy arrays where numerical operations are applied element-wise. These vectorized operations are very fast and don't require any for loops.
7. Rearrange code into Boids Class (8418b186733bde352003146af96021025be1b93c)
   - *Smell*: Replace ad-hoc structure with a class.
   - *Solution*: Rearrange code into Boids class to provide the boids model as a module.
8. Break up boid model in update_boids() into testable units (3e3d5948ddc9b7031af2924bece046f8c5a017e4)
   - *Smell*: A single function implements the whole model.
   - *Solution*: Break large function into smaller testable units. This strongly increases the readability and reduces the complexity.
9. Add first simple unit test (625520368c84ad312e9e6f95b6131123500c1a95)
   - *Smell*: New testable units don't have unit tests yet.
   - *Solution*: Implement unit tests for the new units. This will make it much easier to debug the code.
10. Add regression test for each unit (bdf3033549072e0dc3fbb711c5cebd423c32bc67)
    - *Solution*: Add regression test for each unit in addition to the complete regression test.
11. Add configuration file (efebc63891570004f0063a04fbc95a76ae5ea917)
    - *Smell*: Many constants are defined which need to be changed individually to explore different scenarios.
    - *Solution*: Replace constants with a configuration file by combining them into a dictionary which can be loaded from a YAML file. Add configuration file handling in the command line entry point.

4 UML diagram

UML diagrams are used to decribe class structures in a diagrammatic notation. In the following the class and aggregation diagram of the Boids model is given.

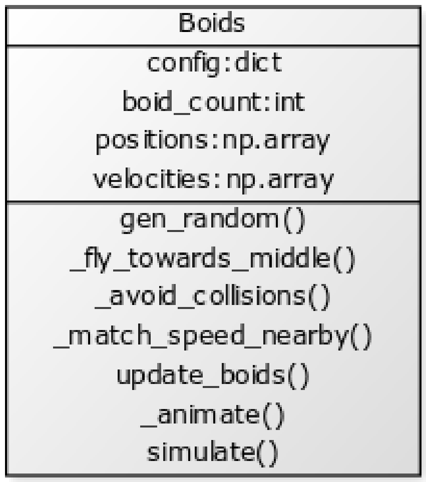

Figure 1: Picture 1

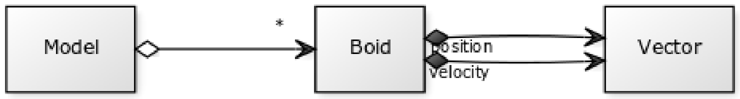

Figure 2: Picture 2

5 Advantages of the refactoring approach

In the refacoting approach software design is changed in order to improve the structure and readability while keeping the functionality untouched. This is usually done stepwise by identifying several smells as illustrated in the list of smells above.

Often these small changes won't have a large impact, however the **cumulative effect will be significant** which is the major advantage of the refactoring approach. By changing the code in very small steps it is much easier to **maintain the program's functionality**. Performing a regression test after each small change helps to check if the code still behaves as expected. If the regression test fails it should be **relatively easy to recover the intended functionality** because only a small part of the code has been changed. Additional version control with individual commits for each refactoring step allows to easily revert the latest changes in case the functionality can not be recovered that easily.

Improving the **readability** and structure of the code is the major goal of the refactoring approach. Often this will also help to improve the **maintainability** by structuring the code into smaller units which are testable. Very often the **code efficiency** will be improved as well. Structuring of the code will also improve the **reusability** of the code. Moreover having a very readable code will strongly increase the chance that it will be reused by others, which is one of the major goals of research software development.

All of these aspects highlight the advantages of the refactoring approach.

6 Problems encountered during the project

I encountered a problem when replacing the nested for-loops in the update_boids() with a numpy element-wise operation (see 6. above). This made the overall regression test fail due to the parallel character of the element-wise calculation which leads to a slightly different result for the positions and velocities. However, the difference was insignificant and I made the regression test pass by increasing the threshold in the assert statement to 0.02. Moreover, later on individual unit tests have been introduced (see 8.-10. above).

Switching to a configuration file also involved more work than I initially thought as I wanted to maintain further felxibility and check for false input. However, having a configuration file makes it very easy to change the simulation parameters.