University College London

EPSRC Centre for Doctoral Training

Delivering Quantum Technologies

MPHYG001 Second Continuous Assessment: Refactoring the Bad Boids

Author:
Gareth Siôn Jones

Student Number:
16070299

February 24, 2017

1 Introduction

The program '*good_boids*' generates a graph showing the flight patterns of flocking birds, and in particular how they avoid colliding with one another during transit. The assigned task was to refactor a deliberately poorly written implementation of this code, and package it ready for release, meeting minimum software engineering requirements. The requirements of this work are as follows:

- Code to be refactored, after identifying several 'smells'
- Final State of code to be well broken down in functions, classes, methods, and modules.
- Employ git version control.
- Add a command line entry point.
- Package ready for PIP installation.
- Create automated tests for each method and class.

This report details the process undertaken to achieve these goals.

1.1 Git Version Control

The project was uploaded to a personal git hub account, and updated and committed regularly. A folder containing the git submission is included with this submission.

2 Command Line Entry Point

One of the objectives of this project was to allow a user to run the program from the command line, using the following example command:

good_boids –file "filename.yml" –size –dist –fly_strength –avoid_dist –strength

Terms preceded with '–' are optional arguments. If the user does not provide these arguments, the execution will default to arguments found in the config.yml file.

In order to achieve this, *run_boids.py* was created, which uses Python's argparse library to parse user input command line arguments:

```
#!/usr/bin/env python
from boids import Flock, Boid
from matplotlib import pyplot as plt
from matplotlib import animation
from argparse import ArgumentParser
import yaml
import os

def parse_args():
    parser = ArgumentParser(description = "Runs the program.")
    parser.add_argument('--file', type=str,
        help='YAML file to load data from')
    parser.add_argument('--size', default=50, type=int,
        help='Number of boids in flock.')
    parser.add_argument('--dist', default=100, type=float,
```

```
            help='Distance over which boids try to match speed.')
        parser.add_argument('--strength', default=0.125, type=float,
            help='How strongly boids try and match speed.')
        parser.add_argument('--avoid_dist', default=10, type=float,
            help='Distance in which boids avoid each other.')
        parser.add_argument('--mid_strength', default=0.01, type=float,
            help='How strongly boids try and flock together.')
        arguments=parser.parse_args()

        initial_position = [[-450,50],[300,600]]
        initial_velocity = [[0,10],[-20,20]]
        axis_limits = [[-500,1500],[-500,1500]]

        cfgdata={"input_file":arguments.file,
            "number_of_boids":arguments.size,
            "group_flying_dist":arguments.dist,
            "group_flying_strength":arguments.strength,
            "alert_distance":arguments.avoid_dist,
            "mid_strength":arguments.mid_strength,
            "initial_position":initial_position,
            "initial_velocity":initial_velocity,
            "axis_limits":plot_axis_limits}

        return cfgdata

def load_config(config_filename):
    cfgfile = open(config_filename,'r')
    cfgdata = yaml.load(cfgfile)
    cfgfile.close()

    cfgdata['input_file'] = config_filename

    return cfgdata

def run(cfgdata):
    number_of_boids = cfgdata["number_of_boids"]
    formation_flying_distance = cfgdata["group_flying_dist"]
    formation_flying_strength = cfgdata["group_flying_strength"]
    alert_distance = cfgdata["alert_distance"]
    attraction_strength = cfgdata["mid_strength"]
    initial_position_range = cfgdata["initial_position"]
    initial_velocity_range = cfgdata["initial_velocity"]
    plot_axis_limits = cfgdata["axis_limits"]

    flock = Flock(flock_size,
              formation_flying_distance,
              formation_flying_strength,
              alert_distance,
              attraction_strength,
              initial_position_range,
              initial_velocity_range)

    figure=plt.figure()
    axes=plt.axes(xlim=(plot_axis_limits[0][0],
                    plot_axis_limits[0][1]),
```

```
                ylim=(plot_axis_limits[1][0],
                      plot_axis_limits[1][1]))

        x_vals = [boid.position[0] for boid in flock.boids]
        y_vals = [boid.position[1] for boid in flock.boids]
        boid_scatter = [boid.position for boid in flock.boids]

        scatter=axes.scatter(x_vals,y_vals)

    def animate(frame):
        flock.new_boids()
        scatter.set_offsets(boid_scatter)

    anim = animation.FuncAnimation(figure, animate,
                                   frames=50, interval=50)

    plt.title('Boids')
    plt.show()
```

This function passes the command line arguments as arguments to methods in the Boid and Flock classes, and plots the resulting flight graph.

The entry point to the program was implemented using the setuptools library in the *setup.py* file:

```
#!/usr/bin/env python

from setuptools import setup, find_packages

setup(
    name = "good_boids",
    version = "1.0.0",
    author = 'Gareth Jones',
    author_email = 'gareth.jones.16@ucl.ac.uk',
    license = 'The MIT License (MIT)',
    packages = find_packages(exclude=['*test']),
    scripts = ['scripts/good_boids'],
    install_requires = ['argparse', 'numpy', 'PyYaml', 'matplotlib']
)
```

The setup file searches for the setup script *good_boids*, which ultimately allows the program to run from the command line:

```
#!/usr/bin/env python
from boids.run_boids import parse_args, load_config, run_boids

cfgdata = parse_args()
if cfgdata["from_file"]:
    cfgdata = load_config(cfgdata["from_file"])
run_boids(cfgdata)
```

Once a user runs the setup file, he can run good_boids from the command line.

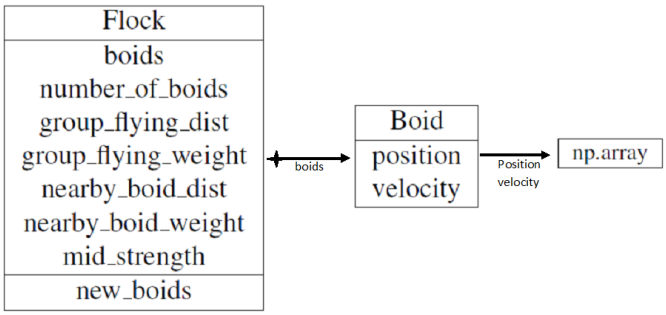
3 Refactoring 'Smells' Encountered

A number of refactoring 'smells' were encountered, and subsequently dealt with in the course of this work. These are listed belows, with the respective commits:

- commit 88b5bba900a8ddddd1208e30ffb830432a2cd9cc - Many raw numbers appeared in the code. These were removed and replaced with variables that could be interacted with.
- commit 36dae594f5397a89ccb96aebf611afad3efaea62 - Some of the functions had poor indentation. This was rectified.
- commit 7214fdaf93a16ff5363888c91614e40eefe3e57a - some of the array functionality was replaced with numpy arrays
- commit 7e602ddfa50896f6d43d1801ac862dce0194a127 commit 613ed899ad36a836d6e6758991ea1cb61ba6c4ec commit 875d2dd54471ee1286bd2f1f36a960444b9d53bd - large parts of the code was unintelligible. This was rewritten so as to be readable by a regular user.
- commit 719a6e11221363e9980f19eacc95469481843ee8 - the function update_boids contained unnecessary for loops. These were removed.
- commit d13576ddba3f06839dec094d628920a9a63e3782 - The code was implemented in an object oriented manner, rather than the given procedural method.

4 Class Structure and UML

The class structure of my code consists of a Flock class, which through boids owns several Boids. Each of these Boids has the property position and velocity, each of which are of the class np.array.



5 Discussion of Refactoring

Refactoring is the process of making gradual changes to existing code without altering its behaviour. the aim is to generate clean, readable, efficient code from otherwise poorly written code. By refactoring in small steps at a time, and checking at each stage the code still functions, it is possible to keep on top of all the changes made without having to spend a significant time debugging in the event that a change alters the operation of the code. Regular regression tests immediately bring attention to bugs and mistakes that would otherwise require a large amount of debugging. The result is code that the code can be significantly improved and ore maintainable.

6 Problems Encountered

The primary difficulty encountered was in attempted to keep the code congruent through each refactor. The code would occasionally stop working and required debugging. However, this itself highlight the benefit of the fractional approach to refactoring, i.e. doing a small amount at a time, and then uploading to git hub. It is then possible to identify exactly where the error may be.

A further problem faced was compatibility between computers. The code seemed to work fine on my home PC, but when transferred to another PC fell down. This is most likely due to the way that the code parameters were being stored in memory on my PC, and was later rectified.