Refactoring a Poor Implementation of Boids

David Wise
University College London
Module: MPHYG001

February 26, 2016

1 Introduction

This report refers to work undertaken refactoring a poor implementation of a boids program and preparing the resulting code for release as part of a package. This was done as part of the research software engineering with python course at UCL. Included here is a description of the refactoring process, specifically with regard to the various 'smells' that were identified in the original code and the steps taken to rectify them.

2 Code Refactoring

Here I will detail the process of converting the code from a poor implementation to an improved one, identifying what was wrong with the original code and what steps were taken to fix it.

1.
   - Smell: The code is contained in one file containing many global variables referenced by different functions
   - Solution: Convert to a class based structure, placing global variables inside the class as properties in the class constructor and the functions as class methods.
   - Git commit: 9e10fb478448c743cf87e06d1c0dc37c8e0bb3c9
2.
   - Smell: Code contains repeated use of 'magic numbers'
   - Solution: Replace magic numbers with sensible named variables defined in class constructor
   - Git commit: da71e643a88238e16d952dc05e5a6a9fc15b1115
3.
   - Smell: Repeated 'boilerplate' code to create boid velocities and positions
   - Solution: Add a 'create-flock' function that is called instead

- Git commit: 6405bcd2f713d4eb5a37a4b27ca61d403adde951; 6f852e4beb72119f6e9c148287f979353d46de1d

4.
   - Smell: Repeated nested for loops inside update-boids function, inefficient and not readable due to poor variable names
   - Solution: Replace with numpy arrays using sensible variable names
   - Git commit: 6f5bc2d435b4c49f42dd7c69c97317c09f4aebc8

5.
   - Smell: update-boids contains a huge amount of functionality for a single function, making implementation of tests hard
   - Solution: Create a series of new functions which handle the velocity corrections required, these are then called by the update-boids function
   - Git commit: 71b4596d71fd28138d510539b4b6302c7980b29f

3 UML Diagram

UML diagrams provide a graphical representation of the structure of code, in this case showing the variables and methods in the Boids class. For more complicated cases it describes how entities interact with one another and is useful when large amounts of code needs to be visualised simply. A UML diagram of my Boids class can be seen in figure 1.

4 Benefits of Refactoring

Refactoring poorly written code acts as an alternative to simply rewriting it from scratch. The chief benefit of this is that it allows a slow progression and change of the code, allowing testing at each step and minimising the introduction of new bugs. Refactoring code aims to improve several aspects of code:

- Readability of code is an important feature, particularly for research programs that are likely used and edited by multiple people. Comments can be helpful but often serve as a crutch to make up for flaws that could be rectified within the code. Defining variables, rather than using magic numbers, and giving them sensible names is helpful, as is naming functions and methods well. Breaking down large functions into smaller ones can help also make code more readable and make structure more immediately apparent.

- Refactoring can improve the speed of code as well - replacing python lists and dictionaries with numpy arrays is one simple and effective means of doing this. Efficiency is another feature that can be vital to research programming, as programs will often run on computers to which one has limited access.

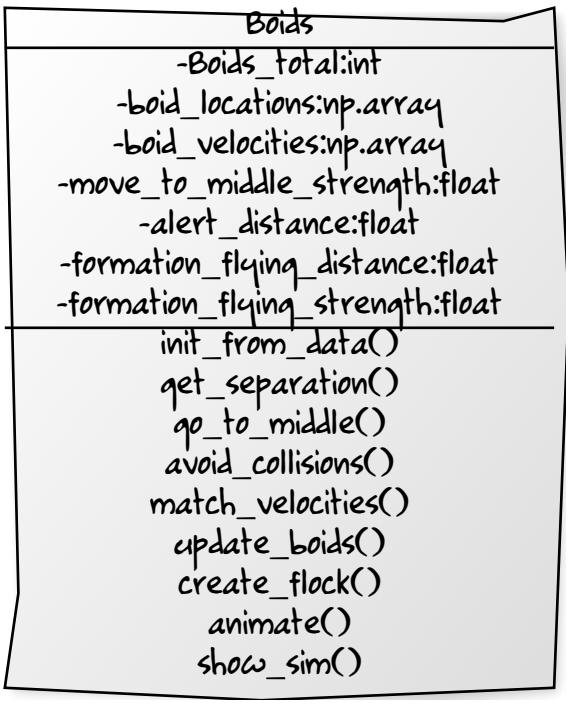

Figure 1: UML diagram of the boids class

- Testing can be made easier by appropriate refactoring, which allows easier maintanance of code and identification of bugs when they develop. Breaking code into classes and short methods allows easier and more efficient testing - the more exactly one can pinpoint a source of failure the quicker it is to fix it.

5 Difficulties Encountered

The chief mistake I made was failing to perform regular tests early on in my refactoring process, meaning that I allowed a couple of bugs to creep in whilst changing to a class based structure. I should probably have gone slower during this phase, making greater use of the regression test early on to ensure the code was still functional at all points.
Switching the update-boids function to be numpy based presented the problem of it outputing slightly different values from the initialising data in the regression test, which required editing of the yaml file to work with the updated functionality. This was subsequently helpful as it allowed easier editing of it later to produce new tests to work with the smaller created methods.
I encountered substantial difficulty creating functions within the class that an-

imate the boids - animate and show-sim. This seemed to me the most efficient way to create a function callable from the command line, but my initial attempts caused only the first frame to be animated, showing the initial boid locations. This was fixed by defining the matplotlib axes and figure within the functions themselves rather than as part of the class constructor.