MPHYG001 Assignment 2: Refactoring Bad-Boids

Paul Brookes

SN: 15078010

University College London

Smells

The majority of the code smells encountered during the refactoring process and the methods used to solve them are summarised in the table below with references to the Git commit log.

Smell	Reference	Solution
Raw numbers.	a160379	Replaced number of boids with boid_count variable.
	f8fe85c	Created variables to specify the window in which positions and velocities are generated
	9d552ac	Created variables to describe the behaviour of the boids ie attraction_strength, etc.
Fragments of repeated code.	8d6d5d6	Created new_flock function to replace repeated code used to generate initial positions and velocities.
	6d68be5	Vectorized new_flock to replace repeated calls of this function.
Calculations which loop over list.	2b76cae 9bf830f	Implemented movement step in update_boids with numpy. More readable, compact and faster.
	59ab6e4	Implemented numpy for attraction to other boids in update_boids.
	0f54795	Implemented numpy for boid repulsion.
	45d0c48	Implemented numpy for boid speed matching.
Code that can be replaced by a library.	See above.	See above.
Confusing variable names.	fa7b90c	Combined x and y co-ordinates of boids into positions variable and x and y velocities into velocities variable to be more readable and easier to handle.
Need to change source code to explore different parameters.	9b2f753	Specified initial limits on boid positions and velocities using a fixtures file.
	7b084c0	Added movement parameters to this fixtures file.
	081dc42	Allowed the movement parameters to be specified as command line arguments.
A large function doesn't fit onto one page.	7473843e	Broke up the Flock.update_boids method into small functions so the code became clearer.

Advantages of the refactoring process.

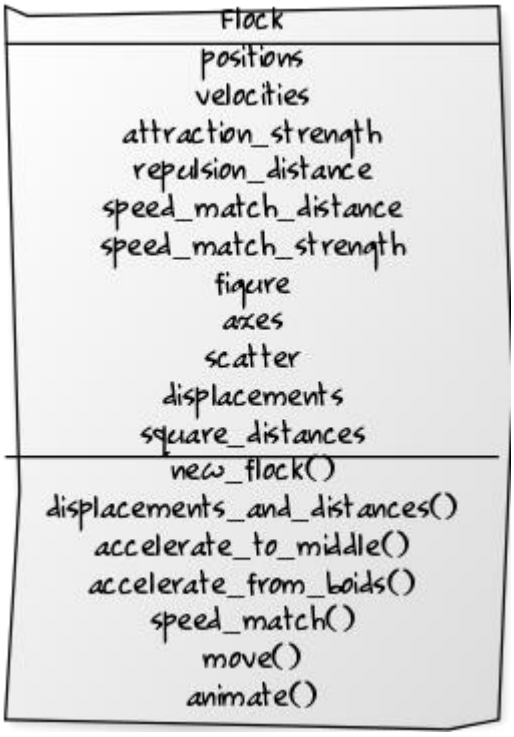
Refactoring is a robust way of improving code. By producing comprehensive regression tests at the start of the refactoring process we can change the way code works while making sure that its functionality isn't altered. This makes refactoring a relatively safe way of developing code. If we are maintaining code which is constant use since the code remains fully functional for users while being continuously improved. Since refactoring consists of many small changes, which are all simple to carry out on their own, the entire process of improving code is simply easier to approach.

Refactoring is a great way of improving the readability of code. Some of the key reasons for code being difficult to read are poor structure, confusing variables names, hand writing code which has already been implemented in a library, etc. All of these problems can be improved by refactoring since they do not change code functionality.

Class diagram

The Flock class consisted of the following variables and methods:

Variables	Descriptions
positions velocities	Data of the positions and locations of all boids in the flock.
displacements square_distances	Data of the separation of all pairs of boids. Needed for controlling when they repel or match their speeds.
attraction_strength repulsion_distsance speed_match_distance speed_match_strength	Model parameters which control the behaviour of the boids.
figure scatter	Plotting objects used for the animation of the flock.
Methods	**Descriptions**
new_flock()	Generates a given number of vectors randomly distributed over a given window.
displacements_and_distances()	Calculated the displacement vectors and square distances between all pairs of boids.
accelerate_to_middle()	Adjust velocities according to attraction between boids.
accelerate_from_boids()	Adjust velocities according to repulsion between boids which are too close.
speed_match()	Adjust velocities to match speeds between nearby boids.
move()	Move boids according to their velocities
animate()	Animate flock.



Difficulties encountered.

The majority of refactorings carried out during the project did not change the output produced by the simulation. They only changed the way calculations were carried out. Therefore after each refactoring the code still passed the regression test. However when converting the flocking code to use numpy the new code broke failed the test.

In the old version of the code speed matching was carried out by iterating over all pairs of boids, comparing their velocities and adjusting the velocity of one accordingly. Since velocity adjustments were made continually throughout the iteration adjustments to the velocities of boids would depend on adjustments which had been made previously during the same round of iteration. The numpy implementation adjusted the velocity of each boid independently of what adjustments would be made to other boids.

In short this meant that the model had been changed and the regression test failed. However the new implementation was a more faithful representation of the model than the old version. This meant it shouldn’t be abandoned and a new fixtures file had to be recorded with the new implementation in order to stop the regression test from failing. The Git log reference for this is 3731a05.