UNIVERSITY COLLEGE LONDON

Refactoring Bad Boids

Module : MPHYG001

Maria Stasinou

Student Number: 16059576

Physics Department- CDT Quantum Technologies

Aim:

The aim of this assignment was to use the refactoring approach in order to construct a clean implementation of the Boids flocking example of this course.

Code smells:

Smell: Global variables and functions related to the boids.

Global variables are accessible by each part of the program and thus there is a probability of being altered by accident. There must also be coherency between the functions and variables in order to achieve efficiency and proper structure. Therefore, we created a Boids class with local variables and functions.

Smell: Repeated values in code.

If a repeated value is changed then all instances should be updated. This can lead to bugs. Thus, we replaced repeated values with more sensibly named ones, so as to improve readability.

Smell: Nested loops to iterate over arrays.

Looping over every element of an array can be inefficient and reduce comprehension. Therefore, we used Numpy arrays to represent the positions and velocities.

Smell: Lots of code into one function.

When there is a lot of code into one function it becomes increasingly complex and reduces readability. Thus, we broke up the code into small functions each of which is aiming at specific tasks.

UML DIAGRAM

Below we indicate the UML diagram for the Boids class.

Boids
Positions: np.array Velocities: np.array Move_to_middle_strength: float Alert_distance: float Formation_flying_distance: float Formation flying strength: float
simulate( ) animate(frame) update_boids (positions, velocities) new_flock (count, lower_limits, upper_limits) separations_square_distances (positions) fly_towards_middle (positions, velocities) fly_away_from_nearby_boids (positions, velocities, separations, square_distances) match_speed_with_nearby_boids (positions, velocities, square_distances)

Advantages of refactoring

Refactoring is the process of changing a computer's program source code without modifying its external functional behaviour in order to improve some of the non functional attributes of the software. We identify several advantages: Code readability, reduced complexity aiming at maintainability of the source code and a more expressive internal architecture to improve extensibility. By maintainability we mean that it is easier to fix bugs and the intent of the author' s code is concrete. By extensibility we mean that it is easier to extend the capabilities of our code and thus gain more flexibility, where none may have existed before.

Problems

One of the main problems encountered was the failure of the regression test which was related to the 'match_speed_with nearby_birds ( )' function. In the original code there was a 'for loop' in which the velocities were updated depending on the state of current velociti es. Thus, the individual updates were not independent. This is not the case however, for the code presented in the lecture which uses Numpy array. The regression test fails because the corresponding algorithm produces slightly different velocities. We therefore constructed the regression test so as to match the new set of velocities. Another issue was that although the positions and velocities were set as class variables they were passed as parameters to the 'update_boids( )'.