Jamie Potter

Student number 16081492

MPHYG001 – coursework 2 – Refactoring the Bad Boids

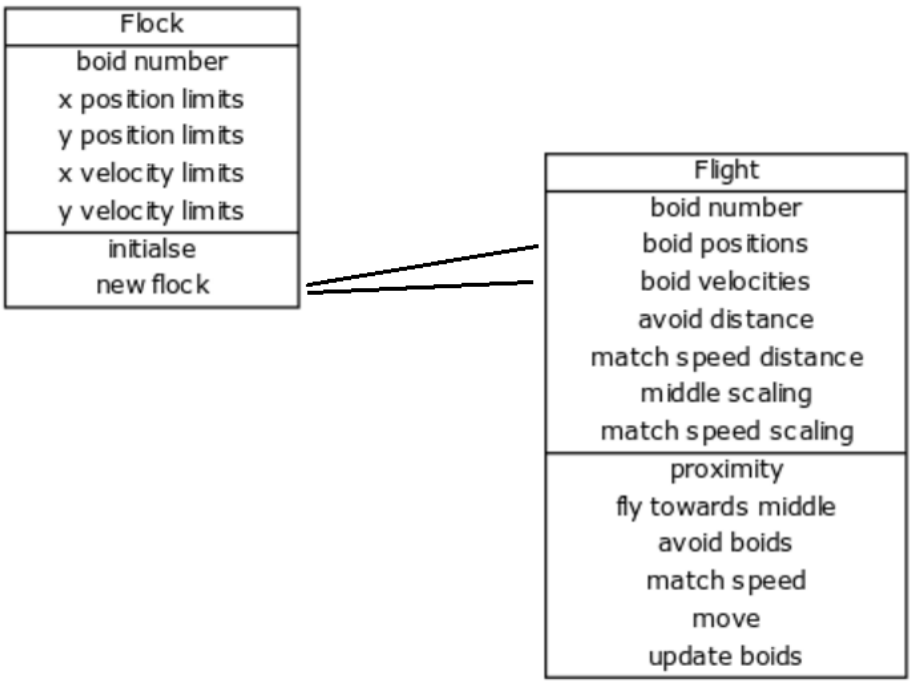
Code smells and refactorings

The following list of code smells and refactorings is not necessarily in chronological order, each smell was not addressed entirely before moving on to the next. The order is roughly representative but the true process can be seen in the Git log.

A lot of raw numbers appeared in the code, for example 50, so these were replaced with constant, for example boid_number. Lots of fragments of code were repeated, so these were replaced with functions, which were repeatedly called. Examples of these functions are initialise, which contains the random number generation, or proximity, which tests how close two given boids are. Lots of variable names were not clear, and data was passed around a lot between different variables, for example xs, vs, xvs, yvs = boids and boids = xs, vs, xvs, yvs made the code confusing to read. The positions and velocities of the boids were renamed boid_positions and boid_velocities, and the data stayed within these variables throughout. Other variable names were also changed to be made clearer. Some loops were repeated unnecessarily, so neighbouring loops were merged where possible. Another smell was that the code needed to be changed to explore other research scenarios, so the constants in the code were replaced by a configuration file. Finally the code was broken down into classes and modules.

The state of the code before being broken down into separate files can be seen in the file bad_boids. Some code smells that weren't addressed were replacing loops with iterators, which seemed to complicate the code rather than simplify it for nested loops and indexing multi-dimensional lists. The hand written code wasn't replaced with library code because to rewrite the code using Numpy felt like more of a fundamental change of the entire program than a refactoring.

UML diagram



Advantages of refactoring

The initial code performed its intended function perfectly well, but was clumsy and difficult to read. We therefore required a method to change this into clear and coherent code, without actually changing the function of the code. Refactoring fits this remit, as gradual changes are made, constantly checking that the program acts exactly as it did before. Over time this approach completely changed the appearance of the code, while fundamentally leaving it exactly the same.

Problems encountered

One problem encountered was that when the code was packaged and this package was pip installed from GitHub, the configuration file was not downloaded with the rest of the package, meaning the code would not work. The solution to this was to include a manifest file, which pointed to the configuration file, and also add another line into the setup file pointing to the manifest file.

For this coursework the command_line file was more complicated than for coursework 1, and this caused problems. In the end the animate function was included within the process function. This is probable not the most elegant solution, but problems were encountered when trying to write a separate file for the animate function, and then call the function from that file. This problem was probably because of my limited understanding.