MPHYG001 Second Continuous Assessment: Refactoring the Bad Boids

Summary

In an appendix, taken from https://github.com/jamespjh/bad-boids, is a very poor implementation of the Boids flocking example, described at http://development.rc.ucl.ac.uk/training/engineering/ch01data/084Boids.html

Your task is to take this code, and build from it a clean implementation of the flocking code, finishing with an appropriate object oriented design, using the **refactoring** approach to gradually and safely build your better solution from the supplied poor solution. Simply submitting a good, clean solution will not complete this assignment; you are being assessed on the step-by-step refactoring process, on the basis of individual git commits. You should develop unit tests for your code as units (functions, classes, methods and modules) emerge, and wrap the code in an appropriate command line entry point providing an appropriate configuration system for simulation setup.

Assignment Presentation

For this coursework assignment, you are expected to submit a short report and your code. The purpose of the report is to answer the non-coding questions below, to present your results and provide a brief description of your design choices and implementation. The report need not be very long or overly detailed, but should provide a succinct record of your coursework. The report must have a cover sheet stating your name, your student number, and the code of the module (MPHYG001).

Submission

You should submit your report and all of your source code so that an independent person can run the code. The code and report must be submitted as a single zip or tgz archive of a folder which contains **git** version control information for your project. Your report should be included as a PDF file, report.pdf, in the root folder of your archive.

All coursework should be submitted electronically through the Moodle for the course. There is no need to include your source code in your report, but you can refer to it and if necessary reproduce lines if it helps to explain your solution. The deadline for submission is Thursday 14th January, 2016. Marks will be available by 11th February.

Marks Scheme

- Final state of code is well broken down into functions, classes, methods and modules [3 marks]
- Final state of code is readable with good variable, function, class and method names and necessary comments [2 marks]
- Git version control used, with a series of sensible commit messages [2 marks]
- Command line entry point and configuration file, using appropriate libraries [3 marks]
- Packaging for `pip` installation, with `setup.py` file with appropriate content [2 marks]
- Automated tests for each method and class. [5 marks total]
- Supplementary files to define license, usage, and citation. [3 marks]
- A text report which:
- Lists by name the code smells identified refactorings used, making reference to the git commit log [2 marks]
- Includes a UML diagram of the final class structure [1 mark]
- Discusses in your own words the advantages of a refactoring approach to improving code [1 marks]
- Discusses problems encountered during the project [1 mark]

[25 marks total]

Appendix

```
from matplotlib import pyplot as plt
from matplotlib import animation
import random

# Deliberately terrible code for teaching purposes

boids_x=[random.uniform(-450,50.0) for x in range(50)]
boids_y=[random.uniform(300.0,600.0) for x in range(50)]
boid_x_velocities=[random.uniform(0,10.0) for x in range(50)]
boid_y_velocities=[random.uniform(-20.0,20.0) for x in range(50)]
boids=(boids_x,boids_y,boid_x_velocities,boid_y_velocities)

def update_boids(boids):
    xs,ys,xvs,yvs=boids
    # Fly towards the middle
    for i in range(50):
        for j in range(50):
            xvs[i]=xvs[i]+(xs[j]-xs[i])*0.01/len(xs)
    for i in range(50):
```

```
        for j in range(50):
            yvs[i]=yvs[i]+(ys[j]-ys[i])*0.01/len(xs)
    # Fly away from nearby boids
    for i in range(50):
        for j in range(50):
            if (xs[j]-xs[i])**2 + (ys[j]-ys[i])**2 < 100:
                xvs[i]=xvs[i]+(xs[i]-xs[j])
                yvs[i]=yvs[i]+(ys[i]-ys[j])
    # Try to match speed with nearby boids
    for i in range(50):
        for j in range(50):
            if (xs[j]-xs[i])**2 + (ys[j]-ys[i])**2 < 10000:
                xvs[i]=xvs[i]+(xvs[j]-xvs[i])*0.125/len(xs)
                yvs[i]=yvs[i]+(yvs[j]-yvs[i])*0.125/len(xs)
    # Move according to velocities
    for i in range(50):
        xs[i]=xs[i]+xvs[i]
        ys[i]=ys[i]+yvs[i]


figure=plt.figure()
axes=plt.axes(xlim=(-500,1500), ylim=(-500,1500))
scatter=axes.scatter(boids[0],boids[1])

def animate(frame):
   update_boids(boids)
   scatter.set_offsets(zip(boids[0],boids[1]))


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

if __name__ == "__main__":
    plt.show()
```