ETGG1803	Lab3: **vector applications (physics + pygame)**	Points: **43**
Assigned: **1/25/2017**	Due[1]: **noon on (Friday) 2/3/2017**	

Tasks:

1. Get a working copy of the vector module (use your lab2 solution, or get mine)
2. A major goal of this lab is to *effectively* use the VectorN class functionality. These will lose you points:
    a. Manually doing vector operations (e.g. a[0] = b[0] + c[0]; a[1] = b[1] + c[1]; … rather than a = b + c)
    b. Using lists where you should be using vectors.
3. (**15 points**) Create a Planet class
    a. Attributes include: a name, radius, color (Vector3), position (Vector3), velocity (Vector3), and a mass
    b. Include an update(dt) and applyForce(forceVector, dt) method as discussed in lecture.
    c. Include a render method which takes a pygame surface and font object. The method should draw a circle (of the given radius and color) and centered on top of it, the name of the planet (drawn with color * 0.5). This method should also draw a line indicating the velocity of the planet as well as the acceleration being applied to it. Make the lengths of these lines proportional to the magnitude of the vector involved. When rendering, just ignore the z component.
4. (**8 points**) Make a function called **load_planets** that loads a text file (see my ETGG1802 page for examples) similar in format to that on the web site (universe01.txt). This function should return a list of planet objects. Each line is of the form:

```
name : mass : radius: red : green : blue : posX : posY : posZ : velX : velY : velZ
```

5. (**8 points**) Make a function called **apply_gravity** that takes a planet list (and dt) as arguments. This should calculate the gravitational force applied to pairs of planets and call applyForce on them. Make sure you only calculate each force once (e.g. if you've already calculated Fab, don't later on recalculate Fba [just apply -Fab to the second planet].
6. (**12 points**) Make a main program that calls all these methods and does the standard pygame startup / shutdown. Close the window when the user presses the close button or the escape key. In the main loop, I'd like you to sort the planets by z value (lower z values should be drawn first).
    a. Python lists can be sorted by…

    ```
    my_list.sort()
    ```

    b. …the catch is, python must know how to sort the objects in the list. For planets, I want you to sort them by z-value. Hint: in the sort method, python will try to do :

    ```
    if my_list[i] < my_list[j]
    ```

    c. So…we need to tell python how to do a less-than operation between planets. Hint: Use the `__lt__` hook.
7. Bonus Features:
    a. (**5 points per planet, up to 15**) Modify the text file so we have an additional planets in an interesting (i.e not flat) stable orbit (meaning that it's on-screen most of the time). It is acceptable if you modify the gravitational constant and any of the existing planets for this.
    b. (**15 points**) Add the ability to toggle between a "top-down" (looking along the z axis) view as described above to a "Cabinet" Projection (see https://en.wikipedia.org/wiki/Graphical_projection for a picture). My advice with this: treat the line going horizontally through the middle of the screen as the x-axis. Move the object diagonally to the north-east if it has a positive y-value (south-west if the y-value is negative). Move it vertically up if it has a positive z-value (vertically down if it has a negative z-value). Adding vertical lines and maybe a grid on the ground to indicate the z-value might help create the illusion as well.
8. Here's a video of my solution (using standard G and universe02.txt): https://youtu.be/AYsEhkvOFXg

[1] Note: We'll likely have a slight overlap with the next lab. So don't procrastinate!