# PS3A

# N-BODY SIMULATION – LOADING UNIVERSE FILES; BODY CLASS; GRAPHICS.

We'll be working through the Princeton assignment at http://www.cs.princeton.edu/courses/archive/fall13/cos126/assignments/nbody.html .

More details will be provided as their material assumes Java and we're using C++.

**For this part of the assignment, Part A, we will create a program that loads and displays a static universe.** In Part B, we will add the physics simulation, and animate the display!

Here are the particular assignment requirements for us:

- Make sure to download the universe specification files and image files from Princeton: ftp://ftp.cs.princeton.edu/pub/cs126/nbody.zip
- You should build a command-line app which reads the universe file (e.g., `planets.txt`) from `stdin`. Name your executable `NBody`, so you would run it with e.g:

```
./NBody < planets.txt
```

The `planets.txt` universe file contains the Sun and the first four planets, with the Sun at the center of universe (x=0, y=0) and the four planets in order toward the right

```
   ...xpos...  ...ypos...  ...xvel...  ...yvel...  ...mass...
filename
   1.4960e+11  0.0000e+00  0.0000e+00  2.9800e+04  5.9740e+24
earth.gif
   2.2790e+11  0.0000e+00  0.0000e+00  2.4100e+04  6.4190e+23
mars.gif
   5.7900e+10  0.0000e+00  0.0000e+00  4.7900e+04  3.3020e+23
mercury.gif
   0.0000e+00  0.0000e+00  0.0000e+00  0.0000e+00  1.9890e+30
sun.gif
   1.0820e+11  0.0000e+00  0.0000e+00  3.5000e+04  4.8690e+24
```

object needed to hold the sprite's image).



- For full credit, you should override the input stream operator >>, and use it to load parameter data into an object.

- Please see the grading rubric for all the details and pieces of the project.
- Please submit all files needed to build your project: `.cpp`, any header files, and a `Makefile`.
- Please submit the `planet.txt` file, and the specific GIF images associated with it.
- Please submit a screenshot of your running code, named `screenshot.png`.
- Fill out and include a Readme (`ps3a-readme.txt`) file with your work, as you did with ps1 and ps2.

# DEVELOPMENT PROCESS

There are a lot of parts to this assignments. We'd suggest the following incremental development process:

1. Create a bare-bones implementation of your `Body` class that has a constructor where you specify all the initial parameters (x,y position and velocity; mass; image filename).
   - Have the constructor load the image into a new Texture object; create a new Sprite with that Texture.
   - Given the initial x,y position in the universe, figure out the corresponding pixel-position for display in an SFML window.
     - Hint 1: your class will need to know and store the universe radius, and display window dimensions.

At this point, the way forward is to:

1. Implement a vector of `Body` objects. For subtle reasons having to do with the default copy constructor, you'll fare better making a vector of pointers to `Body` objects, and instantiating them with `new`.
2. When you have the vector working, you can write the code to overload the stream input operator `>>`, and read in the universe file to set up your bodies. (See http://www.tutorialspoint.com/cplusplus/input_output_operators_overloading.htm for example code.)

# SUBMITTING

The executable file that your `Makefile` builds should be called `NBody`.

Submit using the submit utility as follows:

**submit schakrab ps3a ps3a**

# GRADING RUBRIC

**Core implementation: 8**

**(**full & correct implementation 3; celestial body object is `Drawable` and SFML while loop uses `window.draw(obj)` 1; implementation loads universe from `stdin` 1; body class has `>>` overloaded to read in a row from universe file 1; supports arbitrary number of body objects (per universe file) 1; scaling works for arbitrary universe size and given SFML window size declared in `main.cpp` 1)

for those features)



**Total: 15**

**extra credit: +1 Use background image**