# PS2: N-Body Simulation

## Part A: loading universe files; body class; graphics

In 1687 Sir Isaac Newton formulated the principles governing the motion of two particles under the influence of their mutual gravitational attraction in his famous *Principia*. However, Newton was unable to solve the problem for three particles. Indeed, in general, solutions to systems of three or more particles must be approximated via numerical simulations. Your challenge is to write a program to simulate the motion of *N* particles in the plane, mutually affected by gravitational forces, and animate the results. Such methods are widely used in cosmology, semiconductors, and fluid dynamics to study complex physical systems. Scientists also apply the same techniques to other pairwise interactions including Coulombic, Biot-Savart, and van der Waals.

Average time to complete assignment ~5 hours.

### Program specification

**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 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
```

#### *Reading in the universe*

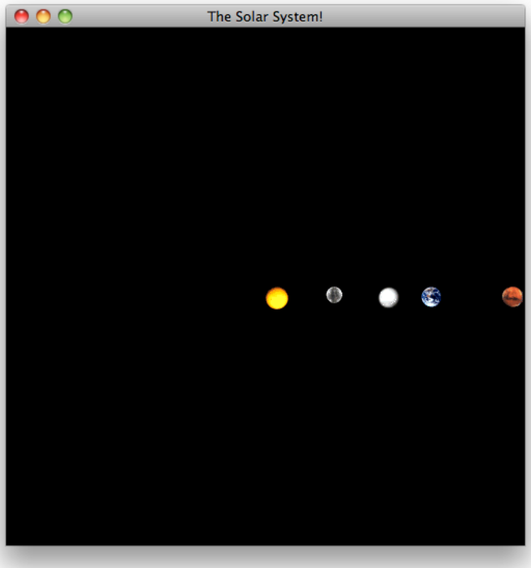
The input format is a text file that contains the information for a particular universe (in SI units). The first value is an integer *N* which represents the number of particles. The second value is a real number *R* which represents the radius of the universe, used to determine the scaling of the drawing window. Finally, there are *N* rows, and each row contains 6 values. The first two values are the *x*- and *y*-coordinates of the initial position; the next pair of values are the x- and y-components of the initial velocity; the fifth value is the mass; the last value is a String that is the name of an image file used to display the particle. As an example, planets.txt contains data for our own solar system (up to Mars):

```
...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   venus.gif
```

You should read in exactly as many rows of body information as are indicated by *N*, the first value in the file.

The `< planets.txt` construct is known as an "*input redirect*".

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 (per below). When this is working, you should be rewarded with:




- For this project you'll implement a `Universe` class which will contain all celestial bodies, and a `CelestialBody` class representing the celestial bodies.
- The class `Universe` should create celestial bodies.
- The class `CelestialBody` should have the following features:

  - It must be `sf::Drawable` with a private virtual void method named `draw`.
  - Each instance of the class should contain all properties needed for the simulation; e.g.: x and y position, x and y velocity, mass, and image data.
  - It probably should contain an `sf::Sprite` object (as well as the `sf::Texture` object needed to hold the sprite's image).

- **For full credit**:

  - you must 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 this `ps2a-readme.txt` file with your work.

## *Development process*

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

1. Create a bare-bones implementation of your `CelestialBody` 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.
     - *Hint* 2: the universe's center is (0,0), and SFML's (0,0) point is in the upper-left.
2. Implement the `draw` method in your `CelestialBody` class.
3. Write a main file that manually creates a `Body` object, by copying initialization parameters from the `planets.txt` file into your source code.
4. Have the main file draw that object in the SFML display loop.

At this point, you should be able to display one planet (or the Sun). You can add one or two more manually, to know you're on the right track.

5. Create class Universe that contains a vector of pointers to CelestialBody objects and instantiate them with new.
6. 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.)
7. The implementation must be contained in files named CelestialBody.cpp and CelestialBody.h, and Universe.h,
8.

### *How to turn it in*

Make sure all your files are in a directory named ps2a,gzipped tar archive should be named PS2a_YourName.tar.gz

Submit your work on Blackboard.

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

## Grading rubric

| **Feature** | **Value** | **Comment** |
|---|---|---|
| core implementation | 10 | full & correct implementation |
| | | 2 pts submission contains all required files with correct names |
| | | 2 pts you have all classes required implemented |
| | | 1 pt celestial body object is Drawable and SFML while loop uses window.draw(obj) |
| | | 1 pt implementation loads universe from stdin |
| | | 1 pt body class has >> overloaded to read in a row from universe file |
| | | 1 pt supports arbitrary number of body objects (per universe file) |
| | | 2 pt scaling works for arbitrary universe size and given SFML window size declared in main.cpp |
| Makefile | 2 | Makefile included |
| | | targets all and clean must exist |
| | | all should build NBody |
| | | must have dependencies correct |
| screenshot.png | 1 | included |
| ps3a-readme.txt | 2 | **must explain how each of the features noted above is implemented to receive credit for those features!** |
| **Total** | **15** | |
| extra credit | +1 | Use background image |