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](http://www.cs.uml.edu/~yrykalov/pmwiki/pmwiki.php/Comp2040/PS3b), 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 construct is known as an ["input redirect"](http://www.cs.uml.edu/~yrykalov/pmwiki/pmwiki.php/Comp2040/InputRedirect).

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:

|  |
| --- |
| http://www.cs.uml.edu/~yrykalov/pmwiki/uploads/Comp2040/ps3-init-state.png |

|  |
| --- |
| ...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 must implement a class representing the celestial bodies. We suggest naming the class Body. The class 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 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 this [ps3a-readme.txt](http://www.cs.uml.edu/~yrykalov/pmwiki/uploads/Comp2040/ps3a-readme.txt) file with your work.

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.
     + 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 Body 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.

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

Submit your work via Bottlenose: <https://grader.cs.uml.edu/assignments/703>

The executable file that your Makefile builds should be called NBody (the grading script checks that this executable builds successfully.)

Grading rubric

|  |  |  |
| --- | --- | --- |
| **Feature** | **Value** | **Comment** |
| core implementation | 70 | full & correct implementation |
|  |  | 10 pt celestial body object is Drawable and SFML while loop uses window.draw( obj ) |
|  |  | 10 pt implementation loads universe from stdin |
|  |  | 10 pt body class has >> overloaded to read in a row from universe file |
|  |  | 10 pt supports arbitrary number of body objects (per universe file) |
|  |  | 10 pt scaling works for arbitrary universe size and given SFML window size declared in main.cpp |
| Makefile | 15 | Makefile included |
|  |  | targets all and clean must exist |
|  |  | all should build NBody |
|  |  | must have dependencies correct |
| screenshot.png | 5 | included |
| ps3a-readme.txt | 10 | **must explain how each of the features noted above is implemented to receive credit for those features!** |
| **Total** | **100** |  |