Lab 3 — The *N*-Body Simulation

Data Structures and Algorithms – Spring 2024

In this lab you will write a Java program to simulate the movement of multiple celestial bodies, using Newton's laws of motion. This lab is based on a programming assignment developed at Princeton by Robert Sedgewick and Kevin Wayne. (More information can be found at http://introcs.cs.princeton.edu/java/assignments/nbody.html)

Imagine *N* celestial bodies (planets, moons, stars, black holes, etc.) all moving and interacting under the influence of gravity in a flat 2-dimensional "universe". Each body keeps track of its current position (x, y), its current velocity (v_x, v_y), and its mass. On a single time step of the simulation, each body computes the sum of all of the gravitational forces acting on it from the other bodies. Then, after all of the forces have been computed, each body updates its current position and velocity based on the forces acting on it. The cycle then repeats continuously.

1. Go to the class web page under *Labs* and download the file **lab03_files.zip**. Unzipping this file will create a folder called lab03_files containing all of the files you will need for this lab. We will use a simple graphics class developed for this assignment called StdDraw. We will only need to use a few of its methods; the full documentation is available at http://introcs.cs.princeton.edu/java/stdlib/javadoc/StdDraw.html. The file **TestStdDraw.java** contains a simple test program that demonstrates most of the methods we will need. Compile and run this program to see what it does. Try changing the last line of the program to pause 0 milliseconds (or some other value) instead of 10. What happens if you omit this line altogether? Try it!

2. We will represent a celestial body as a Body object that keeps track of the following instance variables:
   - `x` and `y`: the body's current spatial position
   - `vx` and `vy`: the body's current velocity in the *x* and *y* directions
   - `mass`: the body's mass
   - `imagefile`: the name of the body's GIF image file (e.g. "earth.gif")
   - `netXforce` and `netYforce`: the total amount of gravitational force currently acting on the body in the *x* and *y* directions

   The file **Body.java** contains skeleton code for the Body class definition. The file **NBodySimulation.java** will be the main program. Open both of these files in DrJava. We will develop the code incrementally, in stages. At each step of the process, you should thoroughly test and debug your code before going on to the next step! You should get into this habit for all of your programming this semester.

3. First off, make sure that your skeleton code compiles and runs, even though it doesn't actually do anything yet. If you encounter a compilation problem, put in a temporary fix so that the code will at least compile.

4. The initial state of the "universe" is specified by a text file containing information about each body in the universe. For example, to develop our program, we will use the file **planets.txt**, which looks like this:

```
5
2.50e+11
 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
```

   The first line specifies the total number of bodies in the universe. The second line is a number *R* (in scientific notation) that specifies the size of the universe: both the *x*-axis and *y*-axis coordinates should be set to the range $[-R, +R]$. Each line after that specifies the initial position (x, y), initial velocity (v_x, v_y), mass, and GIF image filename of a body. The above universe, for example, ranges from -2.5×10^{11} to $+2.5 \times 10^{11}$ meters on each axis. The Earth's initial position is at $(1.496 \times 10^{11}, 0)$ and its initial velocity is 0 meters/sec in the *x* direction and $+2.98 \times 10^{4}$ meters/sec in the *y* direction. Its mass is 5.974×10^{24} kg. Add some code to **NBodySimulation.java** that prompts the user for the name of a universe file, opens the file, and then immediately closes it, without doing anything else. Test your code, and make sure that it works correctly before going on to the next step.

5. Modify **NBodySimulation.java** so that it reads in the *first two numbers* from the file before closing, and then prints them out in the format shown above. Hint: the string formatting code `%.2e` can be used to format a number in scientific notation with 2 decimal places of precision. Test your code to make sure it works correctly.

6. Now add some more code: after reading in the first two numbers, your program should read in the six values from the next line (for Earth) *individually*, and store them in separate variables called `x`, `y`, `vx`, `vy`, `mass`, and `imagefile`, all of which should be floating-point numbers except for `imagefile`, which should be a string. To do this, you'll need to use the Scanner's `next()` method instead of `nextLine`. The `next()` method ignores all whitespace (including newlines) and returns the next complete "token" from the input as a string, where a *token* means any group of contiguous non-whitespace characters. You will also need to use the `Double.parseDouble` method to convert the tokens for position, velocity, and mass from strings to numbers. Make sure your code compiles and runs correctly.

7. The next step is to read in all of the bodies from the file, creating a new Body object for each one and storing all of the objects in an array of type Body[]. Make sure your code compiles and runs correctly.

8. Next, complete the `toString()` method of the Body class so that it returns a string representation of a Body in the same format as the input data. For example, if `b` is a Body object representing the Earth, `b.toString()` should return the string:

   ```
   " 1.4960e+11  0.0000e+00  0.0000e+00  2.9800e+04  5.9740e+24 earth.gif"
   ```

   Hint: using a formatting code of `%11.4e` for each number will provide an extra space to accommodate a negative sign, if needed, and will make the output look more uniform. Add some code to your main program to print out all of the Body objects after reading in the data from the file.

9. Now add some code to draw the initial state of the universe in a graphics window. You can use the `draw()` method of a Body object to draw it in the graphics window at its current spatial position. In your main program, create the graphics window and then call `draw()` for each body in the universe, using a loop. Finally, call `StdDraw.show(2000)` to update the window and pause for 2 seconds. Test your code. You should see the Sun, Mercury, Venus, Earth, and Mars all lined up horizontally in the right half of the window.

10. Now we're ready for some animation! Each time step *T* of our simulation will correspond to 25,000 seconds of real time (about 7 hours). On each time step, each body first needs to calculate the *total net gravitational force* acting on it, in both the *x* and *y* directions, from all other bodies. Then, once all bodies know the net forces acting on them, they can update their velocity and position for this time step. Finally, the graphics window is updated with the new positions of each body, and the cycle is repeated—indefinitely. See the next page for a more detailed description of the algorithm.

    You should define the following five helper methods in your Body class:

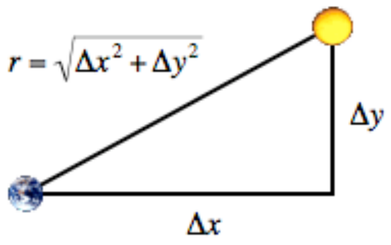
`double getX()`	returns the body's current *x* position
`double getY()`	returns the body's current *y* position
`double getMass()`	returns the body's mass
`void clearForces()`	resets the net forces acting on the body to zero
`void addMoreForce(double Fx, double Fy)`	adds F_x and F_y to the net forces acting on the body

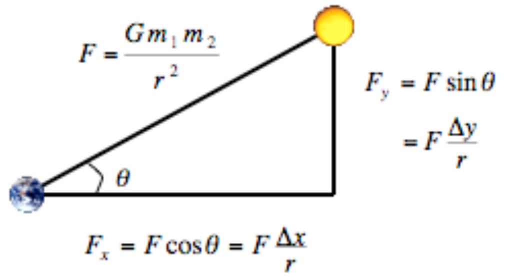
    Feel free to define any other helper methods you like.

11. Once your program works for **planets.txt**, try running it on **planets-elliptical.txt**, which creates elliptical orbits instead of circular ones. The *UniverseFiles* folder contains a large number of other initialization files created by students at Princeton, many of which are truly amazing. Can you create your own variations?

12. For a bigger challenge, here's another idea: try making the simulation three-dimensional by doing calculations for *x*-, *y*-, and *z*-coordinates, then using the *z*-coordinate to vary the sizes of the planets. The method `StdDraw.picture` accepts two extra arguments *w* and *h* that rescale the GIF image to width *w* and height *h*.

Here is the algorithm spelled out in a little more detail:




$F = \frac{G m_1 m_2}{r^2}$
$F_y = F \sin\theta = F \frac{\Delta y}{r}$
θ
$F_x = F \cos\theta = F \frac{\Delta x}{r}$

1. Reset the net forces acting on each body *i* to zero. Use a separate for-loop for this step.

2. For each body *i*, add up the forces F_x and F_y acting on it from all other bodies. This step requires a nested for-loop. That is, for each body *i*, loop through all the other bodies *j* and do the following (except skip the case when *i* and *j* are equal, because a body doesn't act on itself):

   (a) Calculate the *x* and *y* distances from body *j* to body *i*: $\Delta x = x_j - x_i$ and $\Delta y = y_j - y_i$

   (b) Calculate the straight-line distance between body *j* and body *i*: $r = \sqrt{\Delta x^2 + \Delta y^2}$

   (c) Calculate the gravitational force exerted by body *j* on body *i*: $F = (\mathrm{G}\, m_i\, m_j) / r^2$
   (where $\mathrm{G} = 6.67 \times 10^{-11}$ is Newton's gravitational constant)

   (d) Calculate the forces F_x and F_y exerted by body *j* on body *i*: $F_x = F\,\Delta x / r$ and $F_y = F\,\Delta y / r$

   (e) Add F_x and F_y to the net forces acting on body *i*

3. At this point, each body knows the total net forces F_x and F_y acting on it. The next step is to update each body's velocity and position, using another for-loop. That is, for each body *i*:

   (a) Calculate the acceleration acting on the body in the *x* and *y* directions: $a_x = F_x / m$ and $a_y = F_y / m$
   (where F_x and F_y are the total net forces acting on the body).

   (b) Calculate the body's new velocity: $v_x = v_x + T\,a_x$ and $v_y = v_y + T\,a_y$
   (where *T* is 25000, representing one time step of the simulation, in seconds).

   (c) Calculate the body's new position: $x = x + T\,v_x$ and $y = y + T\,v_y$
   (where *T* is 25000, representing one time step of the simulation, in seconds).

4. Update the position of each body in the graphics window.

5. Go to step 1 and repeat.