Programming Assignment Checklist: N-Body Simulation

Goals

- Learn about a scientific computing application.
- Learn about reading input using the `StdIn` library.
- Learn about plotting graphics using the `StdDraw` library.
- Learn about using the command-line to redirect standard input to read from a file.
- Use arrays.

Frequently Asked Questions

What's the easiest way to copy the `nbody` directory from the COS 126 ftp site to my computer?

- *Internet Explorer*. Click the ftp link above, then right click the `nbody` folder, select *Copy To Folder*, and choose the desired location.
- *Safari*. Click the ftp link above to mount the ftp site, then drag-and-drop the `nbody` folder from the ftp mount to the desired location.
- *Firefox*. Install the DownThemAll! plugin to support downloading all of the links contained in a webpage and use it.
- *Alternative*. Click the ftp link above, download the file `nbody.zip`, and unzip the contents.

DrJava doesn't let me redirect standard input. What should I do instead? Use the command-line. Here are instructions for configuring your system to use the command-line: [Windows · Mac · Linux]

What does the following error mean?

```
class file has wrong version 49.0, should be 48.0
```

You are running Java 1.4, not 1.5. You need 1.5 to use `StdIn` and `StdDraw`. Please follow the command-line instructions.

How do I plot a picture using `StdDraw`? The command `StdDraw.picture(x, y, filename)` plots the image in the given filename (either JPEG, GIF, or PNG format) on the canvas, centered on (x, y).

My computer doesn't display the animation smoothly. Is there anything I can do? Use `StdDraw.show(30)` to turn on the animation mode of standard draw. Call it once at the end of each time step, not after each each drawing command.

Can I combine Steps 1, 2, and 3 into one massive loop? No! This will simultaneously screw up the physics and make your code harder to understand and debug.

I draw the planets, but nothing appears on the screen. Why? Use `StdDraw.setXscale()` and `StdDraw.setYscale()` to change the coordinate system to use the physics coordinates instead of the unit box.

My planets don't move. Make sure that you are using a large enough value of Δt (we specify 25000, but you might want a smaller one when debugging).

My planets repel each other. Why don't they attract each other? Make sure that you get the sign right when you apply Newton's law of gravitation: `(x[j] - x[i])` vs. `(x[i] - x[j])`. Note that Δx and Δy can be positive or negative so do not use `Math.abs()`. Do not consider changing the universal gravitational constant G to patch your code!

How should I compute x^2? The simplest way is `x*x`. In Java, the `^` operator means XOR (not exponentiation).

What is Δx? It's the change in *x*-coordinate between two bodies (not between a body at time *t* and at time *t* + Δ*t*).

When I compile NBody.java, it says "cannot resolve symbol StdDraw." Any thoughts? Be sure you have either `StdDraw.java` or `StdDraw.class` in the current directory.

How do I submit the extra credit? Submit the files as usual using Moodle. Document what you did in your readme.txt file.

How do I change my directory to the H: drive from the Windows Command Prompt? Type `H:` at the command prompt. Then `cd` to the appropriate directory.

Submission

readme.txt. Use the following readme file template and answer any questions.

Submission. Submit `NBody.java`. Don't forget to hit the "Run Script" button on the submission system to test that it compiles cleanly.

Possible Progress Steps

These are purely suggestions for how you might make progress. You do not have to follow these steps. Warning: this program is more involved than the previous ones, and you should budget more time accordingly. The best advice we can give you is to carefully test, debug, and re-test your code *as you write it*. Do *not* attempt to write the whole program at once - if you do, then you will have no idea where to find the error if the program doesn't work. We promise that proactive testing will save you enormous amounts of time in the long run. Trust us! Also, if you get stumped or frustrated on some portion of the assignment, you should not hesitate to consult a preceptor.

- Copy the `nbody` directory from the COS 126 ftp site to your computer in a new directory `nbody`. In particular, be sure you have `StdDraw.class`, `StdIn.class`, `StdAudio.class`, the image files, and the sample input files in your working directory.

- Read Section 1.5. Review DeluxeBouncingBall.java for help with standard draw. There are many methods in StdDraw, but for this assignment you won't need more than the ones that are used in DeluxeBouncingBall.java. Review Students.java for help with standard input. Compile and execute both of these programs to make sure your system is configured properly. For DeluxeBouncingBall.java you will also need the sound files laser.wav and pop.wav. For Students.java you can use the file students.txt as input.

- Read in the data file `planets.txt` using `StdIn`. To test that you read it in correctly, print the information back out using `System.out.println()`. Do *not* even think of continuing until you have checked that you read in the data correctly.

- Now, read in the data file and store the information in several arrays. Let `rx[i]`, `ry[i]`, `vx[i]`, `vy[i]`, and `mass[i]` be real numbers which store the current position (`x` and `y` coordinates), velocity (`x` and `y` components), and mass of planet `i`. Let `image[i]` be a string which represents the filename of the GIF image used to display planet `i`. Print the information back out.

- **Step 3.** Plot the background `nightsky.jpg` image. Note that `StdDraw.picture(x, y, file)` centers the picture on (`x`, `y`). Test that it works. Now, write a loop to plot the `N` bodies. If all goes correctly, you should see the four stationary planets and the sun. Now, go and test it on another data file.

- **Step 2.** Write a loop to calculate the new velocity and position for each body. (This code goes before the plotting code you wrote in Step 3.) Since we haven't yet incorporated gravity, assume the acceleration acting on each planet is zero. Add an outer loop to repeat Steps 2 and 3 forever. Test it. You should now see the four planets moving off the screen in a straight line, with constant velocity. *Test it* on another data file.

- **Step 1.** Now, calculate the net force acting on each body. (This code goes before the stuff you wrote in Step 2.) You will need two additional arrays `fx[i]` and `fy[i]` to store the net force acting planet `i`. First, write a loop to initialize all the net forces to 0.0. Then write two nested `for` loops to calculate the net force exerted by planet `j` on planet `i`. Add these values to `fx[i]` and `fy[i]`, but skip the case when `i` equals `j`. Once you have these values computed, go back to Step 2 and use them to compute the acceleration (instead of assuming it is zero). Test your program on *several* data files.

- Finally, add the music. At the beginning of the simulation (outside of any loop), use `StdAudio.play("2001.mid")` to play the 2001 theme. Test it.

Enrichment

- Here are some interesting two-body systems, perhaps relevant to the extra credit. Here is beautiful 21-body system in a figure-8 - reproducing this one will definitely earn you extra credit.

- Here is a website that generates music using an N-body simulator!

- Here's a wealth of information on N-body simulation [pdf].

- **I'm a physicist. Why should I use the leapfrog method instead of the formula I derived in high school? Why does the position update formula use the velocity at the *updated* time step rather than the previous one? Why not use the 1/2 a t^2 formula?** The leapfrog method is more stable for integrating Hamiltonian systems than conventional numerical methods like Euler's method or Runge-Kutta. The leapfrog method is *symplectic*, which means it preserves properties specific to Hamiltonian systems (conservation of linear and angular momentum, time-reversibility, and conservation of energy of the discrete Hamiltonian). In contrast, ordinary numerical methods become dissipative and exhibit qualitatively different long-term behavior. For example, the earth would slowly spiral into (or away from) the sun. For these reasons, symplectic methods are extremely popular for N-body calculations in practice. You asked!

  Here's an explanation.

  The classic *Euler method* updates the position uses the velocity at time t instead of using the updated velocity at time $t + \Delta t$. A better idea is to use the velocity at the midpoint $t + \Delta t / 2$. The leapfrog method does this in a clever way. It maintains the position and velocity one-half time step out of phase: At the beginning of an iteration, (r_x, r_y) represents the position at time t and (v_x, v_y) represents the velocity at time $t - \Delta t / 2$. Interpreting the position and velocity in this way, the updated position $(r_x + \Delta t\, v_x, r_y + \Delta t\, v_y)$. uses the velocity at time $t + \Delta t / 2$. Almost magically, the only special care needed to deal with the half time-steps is to initialize the system's the velocity at time $t = -\Delta t / 2$ (instead of $t = 0$). Note also that the acceleration is computed at time t so that when we update the velocity, we are using the acceleration at the midpoint of the interval under consideration.

  The leapfrog method has several important advantages (over class numerical integration methods such as Euler or Runge-Kutta) for integrating Hamiltonian systems. (See Feynman Lectures on Physics, Vol. I, Sec. 9.6.) The leapfrog method is *symplectic*, which means it preserves properties specific to Hamiltonian systems (conservation of linear and angular momentum, time-reversibility, and conservation of energy of the discrete Hamiltonian). In contrast, ordinary numerical methods become dissipative and exhibit qualitatively different long-term behavior. For example, the earth would slowly spiral into (or away from)

the sun. For these reasons, symplectic methods are extremely popular for N-body calculations in practice. You asked!

- Here's some of the N-body physics.
- N-body simulations play a crucial role in our understanding of the universe. Astrophysicists use it to study stellar dynamics at the galactic center, stellar dynamics in a globular cluster, colliding galaxies, and the formation of the structure of the Universe. The strongest evidence we have for the belief that there is a black hole in the center of the Milky Way comes from very accurate N-body simulations. Many of the problems that astrophysicists want to solve have millions or billions of particles. More sophisticated computational techniques are needed.
- The same methods are also widely used in molecular dynamics, except that the heavenly bodies are replaced by atoms, gravity is replaced by some other force, and the leapfrog method is called *Verlet's method*. With van der Waals forces, the interaction energy may decay as 1/R^6 instead of an inverse square law. Occasionally, 3-way interactions must be taken into account, e.g., to account for the covalent bonds in diamond crystals.

COS 126 Assignments
Kevin Wayne