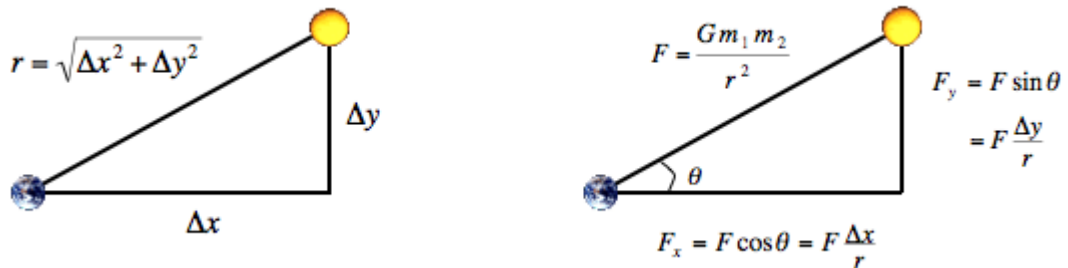


N-Body Simulation

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, systems of three or more particles can only be solved numerically. Write a program to simulate the motion of N particles, 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.

The physics. We review the equations governing the motion of the particles according to Newton's laws of motion and gravitation. Don't worry if your physics is a bit rusty; all of the necessary formulas are included below. We'll assume for now that the position (r_x, r_y) and velocity (v_x, v_y) of each particle is known. In order to model the dynamics of the system, we must know the net force exerted on each particle.

- **Pairwise force.** *Newton's law of universal gravitation* asserts that the strength of the gravitational force between two particles is given by the product of their masses divided by the square of the distance between them, scaled by the gravitational constant G ($6.67 \times 10^{-11} \text{ N m}^2 / \text{kg}^2$). The pull of one particle towards another acts on the line between them. Since we are using Cartesian coordinates to represent the position of a particle, it is convenient to break up the force into its x and y components (F_x, F_y) as illustrated below.



- **Net force.** The *principle of superposition* says that the net force acting on a particle in the x or y direction is the sum of the pairwise forces acting on the particle in that direction.
- **Acceleration.** *Newton's second law of motion* postulates that the accelerations in the x and y directions are given by: $a_x = F_x / m$, $a_y = F_y / m$.

The numerics. We use the *leapfrog finite difference approximation scheme* to numerically integrate the above equations: this is the basis for most astrophysical simulations of gravitational systems. In the leapfrog scheme, we discretize time, and update the time variable t in increments of the *time quantum* Δt . We maintain the position (r_x, r_y) and

velocity (v_x, v_y) of each particle at each time step. The steps below illustrate how to evolve the positions and velocities of the particles.

1. For each particle: Calculate the net force (F_x, F_y) at the current time t acting on that particle using Newton's law of gravitation and the principle of superposition.
2. For each particle:
 - a. Calculate its acceleration (a_x, a_y) at time t using the net force computed in Step 1 and Newton's second law of motion: $a_x = F_x / m$, $a_y = F_y / m$.
 - b. Calculate its new velocity (v_x, v_y) at the next time step by using the acceleration computed in Step 2a and the velocity from the old time step: Assuming the acceleration remains constant in this interval, the new velocity is $(v_x + \Delta t a_x, v_y + \Delta t a_y)$.
 - c. Calculate its new position (r_x, r_y) at time $t + \Delta t$ by using the velocity computed in Step 2b and its old position at time t : Assuming the velocity remains constant in this interval, the new position is $(r_x + \Delta t v_x, r_y + \Delta t v_y)$.
3. For each particle: Draw it using the position computed in Step 2.

The simulation is more accurate when Δt is very small, but this comes at the price of more computation. Use $\Delta t = 25,000$ in this assignment for a reasonable tradeoff.

Creating an animation. Draw each particle at its current position using standard draw, and repeat this process at each time step. By displaying this sequence of snapshots (or frames) in rapid succession, you will create the illusion of movement. After each time step (i) draw the background image `nightsky.jpg`, (ii) redraw all the bodies in their new positions, and (iii) control the animation speed using `StdDraw.show()`.

Input format. The input file is a text file that contains the information for a particular universe. 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: assume all particles will have x - and y -coordinates that remain between $-R$ and R . 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 second two values are the x - and y -coordinates of the initial velocity; the third 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, the input file [planets.txt](#) contains data for our solar system (in SI units).

```
5
2.50e11
1.496e11 0.000e00 0.000e00 2.980e04 5.974e24 earth.gif
2.279e11 0.000e00 0.000e00 2.410e04 6.419e23 mars.gif
5.790e10 0.000e00 0.000e00 4.790e04 3.302e23 mercury.gif
0.000e00 0.000e00 0.000e00 0.000e00 1.989e30 sun.gif
1.082e11 0.000e00 0.000e00 3.500e04 4.869e24 venus.gif
```

The subdirectory `nbody` from the [COS 126 ftp site](#) contains the `planets.txt` file, images of the planets, and many other sample universes.

Your program. Write a program `NBody.java` that reads in the universe from standard input using `StdIn`, simulates its dynamics using the leapfrog scheme described above, and animates it using `StdDraw`. Maintain several arrays to store the data. To make the computer simulation, write an infinite loop that repeatedly updates the position and velocity of the particles. When plotting, use `StdDraw.setXscale(-R, +R)` and `StdDraw.setYscale(-R, +R)` to scale the physics coordinates to the screen coordinates.

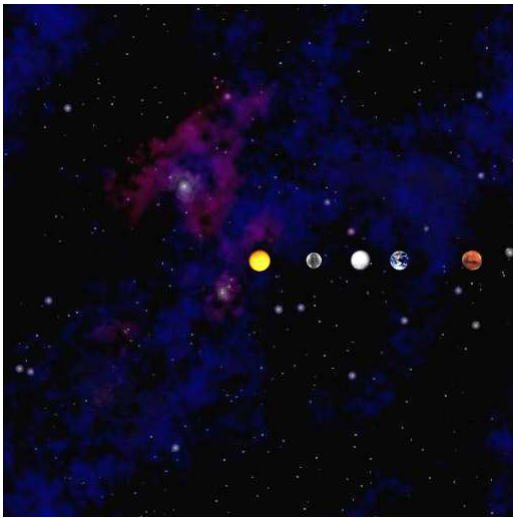
Finishing touch. For a finishing touch, play the theme to *2001: A Space Odyssey* using `StdAudio` and the file `2001.mid`. It's a one-liner using the method `StdAudio.play()`.

Compiling and executing your program. Follow these instructions [[Windows](#) · [Mac](#) · [Linux](#)] to configure the command-line on your system. To compile your program *from the command-line*, type:

```
% javac NBody.java
```

in your Terminal application. To run your program *from the command-line* on the input file `planets.txt`, redirect the file to standard input:

```
% java NBody < planets.txt
```



It produces [this animation](#).

Extra credit. Submit a universe in our input format along with the necessary image files. If its behavior is sufficiently interesting, we'll award extra credit.

Challenge for the bored. There are limitless opportunities for additional excitement and discovery here. Try adding other features, such as supporting elastic or inelastic collisions. Or, make 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. Add a rocket ship that launches from one planet and has to land on another. Allow the rocket ship to exert force with consumption of fuel.