**N Body Simulation**

In 1687, 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 (bodies) 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.

**You will write a program that will do the following:**

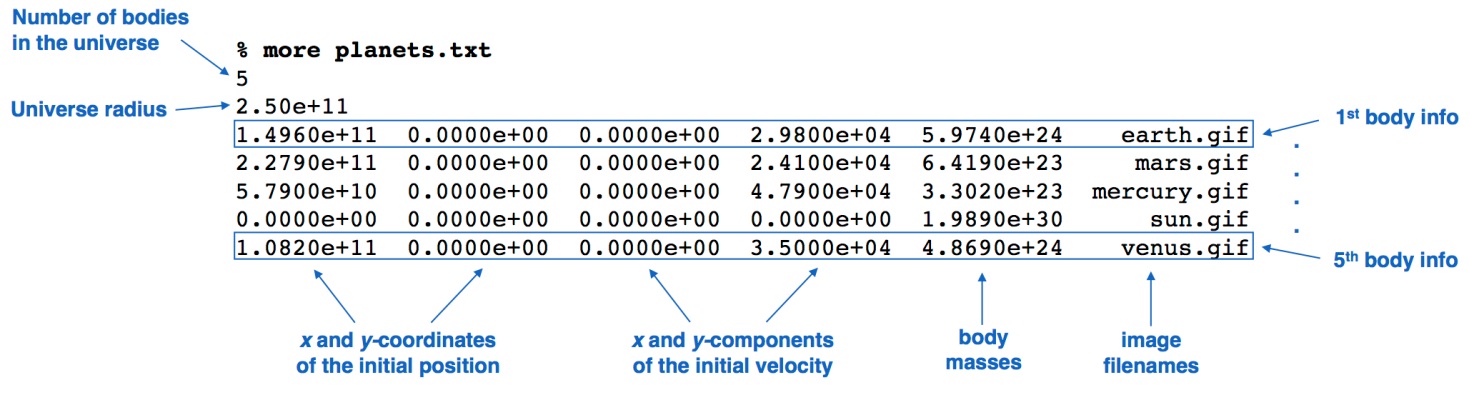
* Reads in the "details of the universe" to be simulated from a text file.
* Simulates the universe, starting at time t = 0.0, and continuing in Δt (the delta symbol (Δ) represents the change in something, in this case the change in time or "time step") increments while t<Τ (where T is the total simulation run time), using the leapfrog scheme described below.
* Animates the results using the drawing methods of the StdDraw class (included).

The input 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 (bodies). The second value is a real number (a double) radius which represents the radius of the universe (which is used to determine the scaling of the drawing window and display particles with X- and Y-coordinates between −radius and +radius).

Next, there are n lines (one for each particle), with each line containing 6 values, as follows:

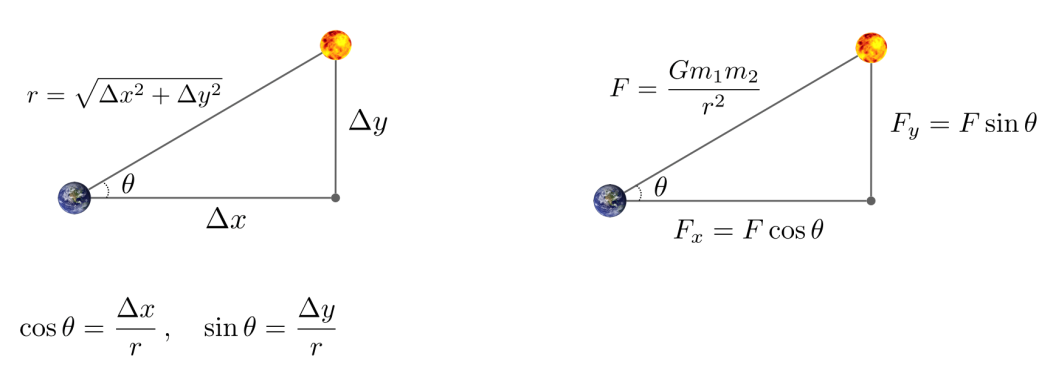
* The first two values are the X- and Y-coordinates, i.e. the body's initial position
* The next pair of values are the X- and Y-components of the initial velocity
* The fifth value is the mass
* The last value is a String that is the name of an image file used to display the particle.

The remainder of the file (optionally) contains a description of the universe, which your program should ignore. The file **"planets.txt"** contains real data from part of our Solar System.



Don't worry if your physics is a bit rusty; all the necessary formulas are provided. We'll assume for now that the starting position (px, py) and velocity (vx, vy) of each particle is known (you'll get this information from the text file). To model the dynamics of the system, we must know the ***net (total) force exerted on each particle***. The calculations you'll need to do are as follows:

**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 × 10−11 N·m2·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 (Fx, Fy) 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 all 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 the equations ax= Fx/ m and ay= Fy/ m.

We will use the leapfrog finite difference approximation scheme to numerically integrate the above equation; this is the basis for many 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 (measured in seconds). We maintain the position (px*,*py) and velocity (vx*,*vy) of each particle at each time step. The steps below illustrate how to evolve the positions and velocities of the particles.

Some starter code has been provided for you. Below are the descriptions of the classes you'll be completing, read on for further instructions!

The **Body** class is a bundle of related variables and methods for keeping track of single particle's (body's) position, velocity, forces acting upon it, etc. A full simulation will be comprised of n bodies. Body is comprised of the following variables:

* private double x, y - body's current X- and Y-coordinate positions at a particular time
* private double xVelocity, yVelocity - body's current X- and Y-components of velocity at a particular time
* private double mass - the mass of this body
* private double xNetForce, yNetForce - the X- and Y-components of the net forces acting on this body at a particular time
* private double xAccel, yAccel - the X- and Y-components of this body's current acceleration at a particular time
* private String image - the file name of the image that will be used when drawing this body

The **NBodySimulation** class represents a total "n-body" simulation, and is comprised of n bodies and the information on the "universe" being simulated (read from a text file). All particles will exert mutual gravitational forces on each other, and the result will be animated. This class has the following:

* private Body[] bodies - stores all the bodies in the simulation
* private int numBodies - the number of bodies in this simulation
* private double uRadius - radius of the universe
* private String fileName - name of the file providing the input

To begin, import the necessary files:

* **BlueJ:** Download the starter code folder and extract (unzip) it. Copy the entire folder to your H: drive and rename it with the lab number and project name.
* **Eclipse:** Create a new project folder (in Eclipse) with the lab number and project name. Download the starter code folder and extract (unzip) it. Drag/drop the .java files into the "src" folder in Eclipse. Drag the images and text file into the main project folder (NOT the "src" folder). Move the "2001.wav" audio file into the "bin" folder (the folder that stores .class files) in Windows Explorer (on disk, not in Eclipse - this folder won't show up by default if using the package explorer view).

**Check the FAQ if you get stuck.**

**For the mathematically disinclined, there is a link to some helpful hints at the bottom.**

Complete the **Body** class as follows:

1. Write Body's constructor. This constructor should initialize all the instance variables to the value of the parameters, except for xNetForce / yNetForce and xAccel / yAccel, which are initialized to 0 (these values are calculated in methods and are not read from the text file).
2. Write a *private* helper method double getDistance(Body other) that returns the distance between this Body (the Body calling the method) and other. Refer to the formulas shown previously.
3. Write a private helper method double getPairwiseForce(Body other) that returns the "pairwise" force between this Body and other, according to Newton's law.
   1. Do not use Math.abs() to "fix" sign issues with the calculations! Force is a vector; in other words, it has a sign. In particular, Δx and Δy (the change in X and Y) are signed.
      1. In the earth / sun diagram shown previously, for the force the sun exerts on the earth, the sun is pulling the earth up (Δy positive) and to the right (Δx positive).
4. Write two private helper methods double getPairwiseForceX(Body other) and getPairwiseForceY(Body other) that return the X- and Y-components of the forces exerted on this Body by other. Make sure to use methods that have already been written!
5. Complete the void setNetForce() method that, using the array of Body objects supplied, calculates and saves (into the instance vars.) the net force exerted on this Body by all the input bodies.
   1. Make sure the force doesn't include the forces exerted in past time steps (set net forces to 0)!
   2. Remember that bodies don't exert gravitational forces on themselves! This would be problematic as it would probably cause the collapse of the universe.
      1. If this Body (the Body calling the setForce() method) is included in the input array, simply continue. To do this, add the following lines:

if (this == bodies[i]) //check if *this* object and object at *i* are same

continue; //more on *this* in the inheritance labs!

1. Write a void update(double dt) method that updates this Body's accelerations, velocities, and positions (coordinates), given the time step supplied. It is necessary to update the body's current X- and Y-position, such that the GUI can re-draw the body in its current location. Some hints:
   1. Acceleration (ax, ay) at time t can be found using the net forces computed previously and Newton's second law of motion: F = m \* a.
   2. Velocity (vx, vy) at the next time step can be found by using the acceleration computed previously and the velocity from the old time step: Assuming the acceleration remains constant in this (small) interval, the new velocity is v + a \* Δt.
   3. To calculate this particle's new position at time t + Δt, use the velocity computed previously and its old position at time t. Assuming the velocity remains constant in this interval, the new position is r + v \* Δt.
2. The draw() method has been completed for you. Nice! This method draws this Body at its current X- and Y-coordinates, given the image file of this Body, using the StdDraw class' picture() method.

Complete the **NBodySimulation** class as follows:

1. Complete NBodySimulation's constructor.
   1. Initialize fileName (given the value of the parameter).
   2. Make a new Scanner object that will get input from the file specified by fileName.
   3. Initialize numBodies and uRadius, given the values in the file.
   4. Initialize the bodies array, given the number of bodies in this simulation.
   5. Iterate through the bodies array, creating a new Body object and saving it into the array, given the information in the file. The order of the information in the file was given previously.
2. (Riddle) 90 D in a R A
3. Complete the run() method, that should do the following:
   1. Iterate from time = 0 to T in increments of dt.
      1. Iterate through all the bodies in the simulation.
         1. Set the net force for the current body.
         2. Call the update() method on the current body, supplying the time step.
      2. Add the following line that will repaint the background:

StdDraw.picture(0, 0, "starfield.jpg");

* + 1. Draw all the bodies in the simulation (use a loop) with the draw() method.
    2. Add the following line that will buffer the graphics and pause for 10ms:

StdDraw.show(10); //don't worry about the compiler warning

A completed runner class with a main() method has been provided. Done properly, your program should show the first four planets in our solar system orbiting the sun, all affected by mutual gravitational forces. You can also uncomment the line in the runner to have your program play music during the simulation.

**(Optional) Other N-body simulations / universes**

The universe specified by the **"planets.txt"** file is just one of many. You can run your program with other simulations using the text files / images found in the ***nbody\_other\_files*** folder.

In addition, you can create an alternate universe of your own (using the same input format and adding images as necessary).

**(Advanced) Extension ideas**

There are *tons* of opportunities for exploration with this project. Try the following:

* Add support for elastic or inelastic collisions between bodies.
* 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.
  + Check the StdDraw class' documentation for methods to scale an image (rather than doing it manually)
* Add a rocket ship that launches from one planet and must land on another. Allow the rocket ship to exert force with the consumption of fuel, and then prepare for job offers from NASA to start pouring in.
  + Seriously, this is *really* cool
  + Also, you could add keyboard controls that would allow you to control the spaceship (which is subject to the gravitational pulls of other planets)

Based on the **N Body Simulation**, version 1.0 project

*http://datastructur.es/sp16/materials/proj/proj0/proj0.html*