The N-body Problem – Part 2 (N Bodies with sfml visuals)

This assignment must be completed individually. Working in groups is not permitted.
Due Monday, April 27th at the start of class.

Solution to Two-Body Problem

1. Start by creating a "Body" object in a header file.
ANSWER: Before creating the `Body` class, you may want to have a `Vector2d` class defined by

```
#pragma once
class Vector2d {
public:
    double x, y;
    Vector2d(double xin = 0, double yin = 0) : x(xin), y(yin) {}
    void operator +=(Vector2d& v) {  //define of operators += and *=
        x += v.x; y += v.y;
    }
    Vector2d operator*(double scalar) {
        return Vector2d(scalar * x, scalar * y);
    }
};
```

This gives a simplified 2-dimensional vector of doubles with the += and *= operators defined as we expect them for vector objects. The `Body` class can make use of this:

```
class Body {
private:
    string name;
    Vector2d position;
    Vector2d velocity;
    double mass;
public:
    Body(string n, Vector2d p = { 0,0 }, Vector2d v = { 0,0 }, double m = 0)
        : name(n), position(p), velocity(v), mass(m) {}
    void setPosition(Vector2d& p) { position += p; }
    void setVelocity(Vector2d& v) { velocity += v; }
    Vector2d getPosition() { return position; }
    Vector2d getVelocity() { return velocity; }
    string getName() { return name; }
    double getMass() { return mass; }
};
ostream& operator<<(ostream& os, Body& b) {
return os << b.getName() << " at {" << b.getPosition().x
                            << ", " << b.getPosition().y << "}\n";
}
```

Note the standard helper function for outputting data to an output stream.

2. The starter code provided left plenty of room for improvement (as usual.) The first blunder was to place Earth at $(0, 0)$ with velocity $(0, 0)$. One should take a more system-oriented approach and place the origin at the center of mass, $\bar{x}$. That is the point about which where the total moment is zero:

$$\sum_{i=0}^{1} m_i(x_i - \bar{x}) = 0$$

$$\bar{x}\sum_{i=0}^{1} m_i = \sum_{i=0}^{1} m_i x_i$$

$$\bar{x} = \frac{\sum_{i=0}^{1} m_i x_i}{\sum_{i=0}^{1} m_i} = \frac{m_0 x_0 + m_1 x_1}{m_0 + m_1} = 0$$

We learn from various authorities that the Earth/Moon distance is $x_1 - x_0 \approx 3.84402 \times 10^8$ meters. Also, the mass of Earth is $m_0 \approx 5.972 \times 10^{24}$ kg and the mass our moon is $m_1 \approx 7.34767 \times 10^{22}$ kg. Substituting $x_1 = x_0 + 3.84402e8$, we have

$$\begin{aligned}\frac{m_0 x_0 + m_1 x_1}{m_0 + m_1} \approx \frac{5.9725e24 \cdot x_0 + 7.34767e22 \cdot (x_0 + 3.844e8)}{5.972e24 + 7.348e22} &\approx 0\\ 5.9725e24 \cdot x_0 + 7.34767e22 \cdot (x_0 + 3.844e8) &\approx 0\\ 6.046e24 \cdot x_0 &\approx -2.824e31\\ x_0 &\approx -4.672e6\end{aligned}$$

whence $x_1 \approx -4.672e6 + 3.84402e8 \approx 3.7973e8$, where, admittedly, I'm following a long COD tradition of handwaving over significant digit issues, and will assert that the initial placement along the x-axis of out moon and Earth can be $(-4.672e6, 0)$ and $93.7973e8, 0)$, respectively.

As to the initial velocities, the sidereal lunar month is 27 days 7 hrs 43 min 11.6 sec, which gives us

```
27 days * 24 hr/day * 3600 sec/hr = 2,332,800 sec
                  7 hr * 3600 sec/hr = 25,200 sec
                 43 min * 60 sec/min =  2,580 sec
                                +      11.6 sec
                              = 2,360,591.6 sec
```

as an approximation for the period of orbit. Since our moon will orbit around the center of mass, the radius of the orbital circle is $r \approx 3.7973e8$, giving the speed of the moon as
$2\pi \cdot 3.9793e8/2.3606e62.3859e9/2.3606e6 \approx 1.0107e3$ meters/sec.
Similarly, Earth's motion can be approximated as a circle of radius $4.672e6$ so that its speed is $\approx 2\pi \cdot 4.672e6/2.3606e6 \approx 12.435$ meters/sec. So we can assume the moon is headed upwards with velocity $(0, 1010.7)$ and Earth is headed downwards at velocity $(0, -12.435)$.

We're now ready to sketch out the broad outlines of the `main()` function:

```
int main() {
    //Some ofstreams for gathering data:
    ofstream moon("moon.dat"), earth("earth.dat");
    // name, position, velocity, mass
    Body Earth("Earth", Vector2d(-4.672e6,0), Vector2d(0,-12.435), 5.972e24);
    Body Moon("Moon", Vector2d(3.7973e8,0), Vector2d(0,1010.7), 7.34767e22);
    vector<Body> bodies{Earth, Moon};

    double dT = 1.0;  //DeltaT for functions is 1 second
    int dT2 = 1;      //for the loop that prints out the output

    while (dT2 <= 4721184) { // 2 * 2360592 seconds / orbit
        update(bodies, dT);
        dT2++; //loop counter
        if (dT2 % 86400 == 0) { //PRINTING OUT EVERY DAY
            cout << "Time Elapsed: " << dT2 / 86400 << " Days" << endl;
            for (int i = 0; i < int(bodies.size()); ++i) {
                cout << bodies[i];
            }
            //Write to data files
            moon << bodies[1];  earth << bodies[0];
        }
    }
}
```

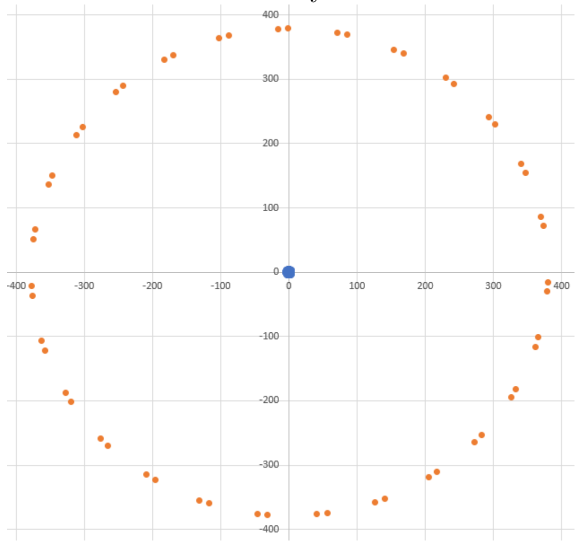
What remains is to define the workhorse, update():

```
void update(std::vector<Body>& bodies, double dT) {
    // vector that holds accelerations for each body
    vector<Vector2d> accelerations(bodies.size());
    // loop that computes total acceleration of each body.
    for (int i = 0; i < int(bodies.size()); i++) {
        for (int j = 0; j < int(bodies.size()); j++) {
            if (i != j) {
               double r = Distance(bodies[i], bodies[j]);
               accelerations[i] = Vector2d(
                   ((G * bodies[j].getMass())
                       * (bodies[j].getPosition().x -
                          bodies[i].getPosition().x
                       / (r * r * r),
                   ((G * bodies[j].getMass())
                     * (bodies[j].getPosition().y -
                        bodies[i].getPosition().y))
                       / (r * r * r));
            }
        }
    }
    // vectors that hold the velocities and positions of the bodies
    vector<Vector2d> changeInVelocities(bodies.size());
    vector<Vector2d> changeInPositions(bodies.size());
    // loop for updating the new velocities and positions
    for (int i = 0; i < int(bodies.size()); ++i) {
        changeInVelocities[i] = accelerations[i] * dT;
        bodies[i].setVelocity(changeInVelocities[i]);
        changeInPositions[i] = bodies[i].getVelocity() * dT;
        bodies[i].setPosition(changeInPositions[i]);
    }
}
```

Notice that this is a fairly robust foundation that will be easy to build on if we want more bodies.

To check that this is behaving reasonably well, I imported the data into MS Excel to produce the graph shown at right. The orange dots comprise two orbits of the moon and the blue dots show Earth, mostly clutered around the origin. The data have been scaled down so that each unit is a megameter (1000 kilometers).

This looks pretty stable!



Part II Problems

For part 2 of this project write code to accomplish all of the following:

1. Add SFML code so you can visualize the motion of the Earth/Moon system simulation.
2. Add our sun to the visualization of system simulation. You'll need to research Sun's mass and average distance to Earth and then re-calibrate the center of mass. You should make Sun yellow and a little larger than Earth, which should be larger than our moon, though not necessarily to scale.
3. Add planets Venus and Mars. What you're building is called an orrery.
4. Get the special three-body "leapfrog" simulation to work.