```
#include "NBody.hpp"

double radius(sf::Vector2f pos1, sf::Vector2f pos2);
sf::Vector2f force(double mass1, double mass2, double radius, sf::Vector2f delta_p);
sf::Vector2f accel(double mass, sf::Vector2f p_force);
sf::Vector2f changeInPosition(sf::Vector2f pos1, sf::Vector2f pos2);

const double G = (6.67e-11);

int main(int argc, char* argv[]){


    if (argc < 3){
        std::cout << "max_time , time increment" << std::endl;
        return -1;
    }

    double max_time = atof(argv[1]);
    double step_time = atof(argv[2]);
    double start_time = 0;

    std::string store;//stores the input from the file
    std::string name;//name of planet
    int numberOfPlanets;
    double radius_of_window;//radius of window
    sf::Vector2f tempForce;
    double c_radius;
    sf::Vector2f c_force;
    sf::Vector2f c_accel;
    sf::Vector2f delta_p;

    //stores the number of planets
    std::cin >> store;
    std::stringstream(store) >> numberOfPlanets;

    std::vector<Body> objects(numberOfPlanets);//vector of objects to store all objects

    //stores the radius of the window
    std::cin >> store;
    std::stringstream(store) >> radius_of_window;

    //loop that stores all relevant data from the file
    for (int x = 0; x < numberOfPlanets; x++){
        std::cin >> objects[x];
    }

    //take the data inside the
    //vector of bodies and print it on the screen using SFML
    sf::RenderWindow window(sf::VideoMode(800, 800), "Ps3b Solar System");

    sf::Image background;
    if(!background.loadFromFile("starfield.jpg"))
        return -1;
    sf::Texture backtex;
    backtex.loadFromImage(background);

    sf::Sprite backsprite;
    backsprite.setTexture(backtex);

    //plays audio
    sf::SoundBuffer buff;
    buff.loadFromFile("st.wav");
    sf::Sound sound;
    sound.setBuffer(buff);
    sound.play();

    //backsprite.setScale(500,500);

    //display window
    while(window.isOpen()){
        sf::Event event;
        while(window.pollEvent(event)){
            if(event.type == sf::Event::Closed)
                window.close();
        }
        window.clear();
        window.draw(backsprite);
        //as long as we don’t go past the max time
        if(start_time < max_time){
            //display all the planets
            for(int i = 0; i < numberOfPlanets; i++){
                window.draw(objects[i]);
            }
            //for every planet
            for(int i = 0; i < numberOfPlanets; i++){
                //don’t do it for the current planet
                if(i != 3){
                    delta_p = changeInPosition(objects[i].getPosition(),objects[3].getPosition());
                    c_radius = radius(objects[i].getPosition(),objects[3].getPosition());
                    c_force = force(objects[i].getMass(), objects[3].getMass(),c_radius, delta_p);
                    c_accel = accel(objects[i].getMass(), c_force);
                    objects[i].setAccel(c_accel);
                    objects[i].step(step_time);
                }
            }
            //run step() to calculate the new positions, update start_time
            start_time += step_time;

        }
        window.display();
    }

    return 0;
}

sf::Vector2f changeInPosition(sf::Vector2f pos1, sf::Vector2f pos2){
    sf::Vector2f change_p;

    change_p.x = pos2.x - pos1.x;
    change_p.y = pos2.y - pos1.y;

    return change_p;
}

double radius(sf::Vector2f pos1, sf::Vector2f pos2){
  return std::sqrt(std::pow(pos1.x - pos2.x,2) + std::pow(pos1.y - pos2.y,2));
```

```
}

sf::Vector2f force(double mass1, double mass2, double radius,sf::Vector2f delta_p){
    double F = (G * (mass1 * mass2)/(std::pow(radius,2)));

    sf::Vector2f f_temp;

    f_temp.x = F * (delta_p.x/radius);
    f_temp.y = F * (delta_p.y/radius);

    return f_temp;


}
sf::Vector2f accel(double mass, sf::Vector2f p_force){
    sf::Vector2f cook;

    cook.x = p_force.x/mass;
    cook.y = p_force.y/mass;

    return cook;
}
```