```
CC = g++
CFLAGS = -Wall -Werror -pedantic --std=c++14
LIBS = -lboost_unit_test_framework
DEPS = CelestialBody.h Universe.h
SFMLFLAGS = -lsfml-graphics -lsfml-window -lsfml-system

%.o: %.cpp $(DEPS)
	$(CC) $(CFLAGS) -c $<

all: NBody

NBody: main.o CelestialBody.o Universe.o
	$(CC) $(CFLAGS) -o NBody $^  $(LIBS) $(SFMLFLAGS)

clean:
	rm *.o NBody
```

```
/**
* main.cpp - as a base to run the program
*
* Date 2/14/22 - 2/22/22
*
* Created by: Anson Cheang
*
*/

/*#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>*/
#include "CelestialBody.h"
#include "Universe.h"
#include <iostream>
#include <cstdlib>

using namespace std;

int main(int argc, char* argv[])
{
    double time = atoi(argv[1]);
    double seconds = 0;
        sf::RenderWindow window(sf::VideoMode(700, 700), "Input");

        window.setVerticalSyncEnabled(true);
        window.setFramerateLimit(15);

    sf::Image image;
    if(!image.loadFromFile("starfield.jpg"))
    {
        return -1;
    }
    sf::Texture texture;
    texture.loadFromImage(image);
    sf::Sprite sprite;
    sprite.setTexture(texture);
    sf::Vector2u size = image.getSize();
    sprite.setScale((1+700/size.x),(1+700/size.y));

    int amount;

    cin >> amount;

    Universe space(amount);

    while (window.isOpen())
        {
                sf::Event event;
                while (window.pollEvent(event))
                {
                        if (event.type == sf::Event::Closed)
            {
                                window.close();
            }
                }

                window.clear();
        window.draw(sprite);
        if(seconds <= time)
        {
            space.step(atoi(argv[2]));
            seconds += atoi(argv[2]);
        }
        window.draw(space);
```

```
                window.display();
        }
    cout << space;
    return 0;
}
```

```cpp
#ifndef Universe_H_
#define Universe_H_

#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>
#include <vector>
#include <iostream>
#include "CelestialBody.h"

using namespace std;

class Universe : public sf::Drawable
{
public:
    Universe(int size);
    void step(double seconds);
    friend ostream& operator<<(ostream& out, const Universe& Galaxy);
private:

    void draw(sf::RenderTarget& target, sf::RenderStates states) const;
    int galaxySize;
    double maxR;
    vector<unique_ptr<CelestialBody> > galaxy;
};

ostream& operator<<(ostream& out, const Universe& Galaxy);

#endif
```

```
/**
* Universe.cpp - as an implementation to store every CelestialBody object
* in order to have them created, and draw them out, essentially storage
* and also to do the physics to make each individual particle move in rotation
*
* Date 2/14/22 - 2/22/22
*
* Created by: Anson Cheang
*
*/

#include "Universe.h"
#include <cmath>

Universe::Universe(int size)
{
    //double posX, posY, Xvel, Yvel, Imass;
    double radius, scale;
    galaxySize = size;
    //string filename;
    cin >> radius;
    maxR = radius;
    scale = 350/radius;
    //double scaledXPos, scaledYPos;
    for(int i = 0; i < size; i++)
    {
        galaxy.push_back(make_unique<CelestialBody>(scale));
        cin >> *(galaxy[i]);
        galaxy[i]->createImage();
        //cin >> posX >> posY >> Xvel >> Yvel >> Imass >> filename;
        //scaledXPos = posX*scale + 350;
        //scaledYPos = posY*scale + 350;
        //galaxy.push_back(make_unique<CelestialBody>(scaledXPos, scaledYPos, Xvel, Yvel, Imass, filename));
    }
}

void Universe::draw(sf::RenderTarget& target, sf::RenderStates states) const
{
    for(int i = 0; i < galaxySize; i++)
    {
        target.draw(*(galaxy[i]), states);
    }
}

void Universe::step(double second)
{
    sf::Vector2f netForce;
    sf::Vector2f Accelleration;
    sf::Vector2f velocity;
    sf::Vector2f p;
    double CX, CY;
    double radius;
    double grav = 6.67e-11;
    double force;
    for(int i = 0; i < galaxySize; i++)
    {
        netForce.x = 0;
        netForce.y = 0;
        for(int j = 0; j < galaxySize; j++)
        {
            if(i != j)
            {
                CX = galaxy[i]->getXPos()-galaxy[j]->getXPos();
                CY = galaxy[i]->getYPos()-galaxy[j]->getYPos();
                radius = sqrt(pow(CX, 2) + pow(CY, 2));
                force = (grav * galaxy[i]->getMass() * galaxy[j]->getMass())/pow(radius, 2);
                netForce.x += force * (CX/radius);
                netForce.y += force * (CY/radius);
            }
        }
        Accelleration.x = netForce.x/galaxy[i]->getMass();
        Accelleration.y = netForce.y/galaxy[i]->getMass();

        velocity.x = galaxy[i]->getXVel() + second * Accelleration.x;
        velocity.y = galaxy[i]->getYVel() + second * Accelleration.y;
        galaxy[i]->setVel(velocity);

        p.x = galaxy[i]->getXPos() + -(second * velocity.x);
        p.y = galaxy[i]->getYPos() + -(second * velocity.y);
        galaxy[i]->setPos(p);
        galaxy[i]->setImagePos();
    }
}

ostream& operator<<(ostream& out, const Universe& space)
{
    out << space.galaxySize << endl;
    out << space.maxR << endl;
    for(int i = 0; i < space.galaxySize; i++)
    {
        out << *(space.galaxy[i]);
    }
    return out;
}
```

```
#ifndef CelestialBody_H_
#define CelestialBody_H_

#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>
#include <string>
#include <cstdlib>
#include <iostream>

using namespace std;

class CelestialBody : public sf::Drawable
{
public:

CelestialBody(double val);
void createImage();

CelestialBody(double posX, double posY, double Xvel, double Yvel, double Imass, string _filename);
friend istream& operator>>(istream& instream, CelestialBody& planet);
friend ostream& operator<<(ostream& out, CelestialBody planet);

void setPos(sf::Vector2f Pos);
void setVel(sf::Vector2f Vel);
void setImagePos();
double getXPos();
double getYPos();
double getMass();
double getXVel();
double getYVel();

private:

void draw(sf::RenderTarget& target, sf::RenderStates states) const;
double XPosition;
double YPosition;
double XVelocity;
double YVelocity;
double Mass;
double scale;
string filename;
sf::Image image;
sf::Texture texture;
sf::Sprite sprite;
};


istream& operator>>(istream& instream, CelestialBody& planet);
ostream& operator<<(ostream& out, CelestialBody planet);

#endif
```

```cpp
/**
* CelestialBody.cpp - an implementation to create each celestial body
* 1 at a time, and also place them into the correct location
* for drawing. plus draw each one individually, and overode >> operator
*
* Date 2/14/22 - 2/22/22
*
* Created by: Anson Cheang
*
*/

#include "CelestialBody.h"
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>
#include <string>
#include <cstdlib>
#include <iostream>

using namespace std;

CelestialBody::CelestialBody(double val)
{
    scale = val;
    XPosition = 0;
    YPosition = 0;
    XVelocity = 0;
    YVelocity = 0;
    Mass = 0;
    filename = "";
}


void CelestialBody::createImage()
{
    if(!image.loadFromFile(filename))
    {
        exit(-1);
    }

    texture.loadFromImage(image);

    sprite.setTexture(texture);
    sf::Vector2u size = image.getSize();
    sprite.setOrigin(static_cast<int>(size.x)/2, static_cast<int>(size.y)/2);
    sprite.setPosition(sf::Vector2f(XPosition*scale + 350, YPosition*scale + 350));
}

CelestialBody::CelestialBody(double posX, double posY, double Xvel, double Yvel, double Imass, string _filename)
{
    XPosition = posX;
    YPosition = posY;
    XVelocity = Xvel;
    YVelocity = Yvel;
    Mass = Imass;
    filename = _filename;

    if(!image.loadFromFile(filename))
    {
        exit(-1);
    }

    texture.loadFromImage(image);

    sprite.setTexture(texture);
    sf::Vector2u size = image.getSize();
    sprite.setOrigin(static_cast<int>(size.x)/2, static_cast<int>(size.y)/2);
    sprite.setPosition(sf::Vector2f(posX, posY));
}

void CelestialBody::draw(sf::RenderTarget& target, sf::RenderStates states) const
{
    target.draw(sprite, states);
}

istream& operator>>(istream& instream, CelestialBody& planet)
{
    instream >> planet.XPosition >> planet.YPosition >> planet.XVelocity >> planet.YVelocity >> planet.Mass >> planet.filename;
    return instream;
}

void CelestialBody::setPos(sf::Vector2f Pos)
{
    XPosition = Pos.x;
    YPosition = Pos.y;
}

void CelestialBody::setVel(sf::Vector2f Vel)
{
    XVelocity = Vel.x;
    YVelocity = Vel.y;
}

double CelestialBody::getXPos()
{
    return XPosition;
}

double CelestialBody::getYPos()
{
    return YPosition;
}

double CelestialBody::getMass()
{
    return Mass;
}

void CelestialBody::setImagePos()
{
    //double CX = (Pos.x - XPosition) * scale;
    //double CY = (Pos.y - YPosition) * scale;
    sprite.setPosition(sf::Vector2f(XPosition*scale + 350, YPosition*scale + 350));
    //cout << XPosition*scale + 350 << ", " << YPosition*scale + 350 << endl;
    //XPosition = Pos.x;
    //YPosition = Pos.y;
}

double CelestialBody::getXVel()
{
    return XVelocity;
}
```

```

double CelestialBody::getYVel()
{
    return YVelocity;
}


ostream& operator<<(ostream& out, CelestialBody planet)
{
    out << planet.XPosition << "  " << planet.YPosition << "  " << planet.XVelocity << "  "
        << planet.YVelocity << "  " << planet.Mass << "  " << planet.filename << endl;
    return out;
}
```