```
    1: /**
    2: * Universe.cpp - as an implementation to store every CelestialBody object
    3: * in order to have them created, and draw them out, essentially storage
    4: * and also to do the physics to make each individual particle move in rot
ation
    5: *
    6: * Date 2/14/22 - 2/22/22
    7: *
    8: * Created by: Anson Cheang
    9: *
   10: */
   11:
   12: #include "Universe.h"
   13: #include <cmath>
   14:
   15: Universe::Universe(int size)
   16: {
   17:     //double posX, posY, Xvel, Yvel, Imass;
   18:     double radius, scale;
   19:     galaxySize = size;
   20:     //string filename;
   21:     cin >> radius;
   22:     maxR = radius;
   23:     scale = 350/radius;
   24:     //double scaledXPos, scaledYPos;
   25:     for(int i = 0; i < size; i++)
   26:     {
   27:         galaxy.push_back(make_unique<CelestialBody>(scale));
   28:         cin >> *(galaxy[i]);
   29:         galaxy[i]->createImage();
   30:         //cin >> posX >> posY >> Xvel >> Yvel >> Imass >> filename;
   31:         //scaledXPos = posX*scale + 350;
   32:         //scaledYPos = posY*scale + 350;
   33:         //galaxy.push_back(make_unique<CelestialBody>(scaledXPos, scaledY
Pos, Xvel, Yvel, Imass, filename));
   34:     }
   35: }
   36:
   37: void Universe::draw(sf::RenderTarget& target, sf::RenderStates states) co
nst
   38: {
   39:     for(int i = 0; i < galaxySize; i++)
   40:     {
   41:         target.draw(*(galaxy[i]), states);
   42:     }
   43: }
   44:
   45: void Universe::step(double second)
   46: {
   47:     sf::Vector2f netForce;
   48:     sf::Vector2f Accelleration;
   49:     sf::Vector2f velocity;
   50:     sf::Vector2f p;
   51:     double CX, CY;
   52:     double radius;
   53:     double grav = 6.67e-11;
   54:     double force;
   55:     for(int i = 0; i < galaxySize; i++)
   56:     {
   57:         netForce.x = 0;
   58:         netForce.y = 0;
   59:         for(int j = 0; j < galaxySize; j++)
   60:         {
   61:             if(i != j)
   62:             {
```

```
   63:                 CX = galaxy[i]->getXPos()-galaxy[j]->getXPos();
   64:                 CY = galaxy[i]->getYPos()-galaxy[j]->getYPos();
   65:                 radius = sqrt(pow(CX, 2) + pow(CY, 2));
   66:                 force = (grav * galaxy[i]->getMass() * galaxy[j]->getMass
())/pow(radius, 2);
   67:                 netForce.x += force * (CX/radius);
   68:                 netForce.y += force * (CY/radius);
   69:             }
   70:         }
   71:         Accelleration.x = netForce.x/galaxy[i]->getMass();
   72:         Accelleration.y = netForce.y/galaxy[i]->getMass();
   73:
   74:         velocity.x = galaxy[i]->getXVel() + second * Accelleration.x;
   75:         velocity.y = galaxy[i]->getYVel() + second * Accelleration.y;
   76:         galaxy[i]->setVel(velocity);
   77:
   78:         p.x = galaxy[i]->getXPos() + -(second * velocity.x);
   79:         p.y = galaxy[i]->getYPos() + -(second * velocity.y);
   80:         galaxy[i]->setPos(p);
   81:         galaxy[i]->setImagePos();
   82:     }
   83: }
   84:
   85: ostream& operator<<(ostream& out, const Universe& space)
   86: {
   87:     out << space.galaxySize << endl;
   88:     out << space.maxR << endl;
   89:     for(int i = 0; i < space.galaxySize; i++)
   90:     {
   91:         out << *(space.galaxy[i]);
   92:     }
   93:     return out;
   94: }
```