```
    1: #include "CelestialBody.hpp"
    2:
    3: #include <iostream>
    4: #include <cmath>
    5:
    6:
    7: using namespace std;
    8:
    9:
   10: Universe::Universe(){
   11:
   12:   r = 0;
   13:   winsize = 0;
   14: }
   15:
   16: Universe::Universe(double radius, int window, int num_of_planets, istream
 &in){
   17:
   18:   int i;
   19:
   20:   r = radius;
   21:   winsize = window;
   22:   numplanets = num_of_planets;
   23:
   24:   for (i = 0; i < num_of_planets; i++){
   25:
   26:     unique_ptr <CelestialBody> ptr(new CelestialBody());
   27:
   28:     CelestialBody();
   29:     planets.push_back(move(ptr));
   30:     planets[i]->set_radius(r);
   31:     planets[i]->set_window(window);
   32:     in >> *planets[i];
   33:   }
   34:
   35: }
   36:
   37: void Universe::draw(sf::RenderTarget &target, sf::RenderStates states) co
nst {
   38:
   39:   int i;
   40:
   41:   for (i = 0; i < numplanets; i++){
   42:     target.draw(*planets.at(i), states);
   43:   }
   44: }
   45:
   46:
   47: void Universe::step(double seconds){
   48:
   49:   int i, k;
   50:
   51:   double ax;
   52:   double ay;
   53:
   54:   double dx;
   55:   double dy;
   56:
   57:   double force;
   58:   double forcex;
   59:   double forcey;
   60:
   61:   double fx;
   62:   double fy;
   63:
```

```
   64:   double G;
   65:
   66:   double velx;
   67:   double vely;
   68:
   69:   double x2;
   70:   double y2;
   71:
   72:   double r;
   73:
   74:   for (i = 0; i < numplanets; i++){
   75:     fx = 0;
   76:     fy = 0;
   77:
   78:     for (k = 0; k < numplanets; k++){
   79:
   80:       if (k != i){
   81:
   82:         G = 6.67e-11; // gravitational constant
   83:
   84:         dx = planets[k]->get_posx() - planets[i]->get_posx();
   85:         dy = planets[k]->get_posy() - planets[i]->get_posy();
   86:
   87:         r = sqrt(pow(dx, 2) + pow(dy, 2));
   88:
   89:         force = (G * planets[k]->get_mass() * planets[i]->get_mass())/ po
w(r, 2);
   90:         forcex = force * (dx / r);
   91:         forcey = force * (dy / r);
   92:
   93:         fy += forcey;
   94:         fx += forcex;
   95:
   96:       }
   97:     }
   98:
   99:     ax = fx / planets[i]->get_mass();
  100:     ay = fy / planets[i]->get_mass();
  101:
  102:     velx = planets[i]->get_velx() + seconds * ax;
  103:     vely = planets[i]->get_vely() + seconds * ay;
  104:
  105:     planets[i]->set_velx(velx);
  106:     planets[i]->set_vely(vely);
  107:
  108:     x2 = (planets[i]->get_posx()) + velx * seconds;
  109:     y2 = (planets[i]->get_posy()) + vely * seconds;
  110:
  111:     planets[i]->set_x_y_pos(x2, y2);
  112:   }
  113: }
  114:
  115: void Universe::printInfo(){
  116:
  117:   int i; // for loop
  118:
  119:   cout << numplanets << endl;
  120:   cout << r << endl;
  121:   for (i = 0; i < numplanets; i++){
  122:     cout << planets[i]->get_posx() << " " << planets[i]->get_posy() << "
"
  123:       << planets[i]->get_velx() << " " << planets[i]->get_vely() << " "
  124:       << planets[i]->get_mass() << " " << planets[i]->get_filename() << e
ndl;
  125:   } // print out x pos, y pos, velx, vely, mass, and name
```

```
  126: }
  127:
  128: double Universe::get_r(){ return r; }
  129:
  130: int Universe::get_numPlanets(){ return numplanets; }
  131:
  132: CelestialBody::CelestialBody(){
  133:
  134:   winsize = 0;
  135:   xpos = 0;
  136:   ypos = 0;
  137:   xvel = 0;
  138:   yvel = 0;
  139:   mass = 0;
  140:   radius = 0;
  141:   filename = "";
  142: }
  143:
  144: CelestialBody::CelestialBody(double x_pos, double y_pos, double x_vel,
  145:                              double y_vel, double m, string name,
  146:                              double rad, double window_size){
  147:
  148:   double radx;
  149:   double rady;
  150:
  151:   xpos = x_pos; // updated values of xpos
  152:   ypos = y_pos; // updated values of ypos
  153:   xvel = x_vel; // updated values of xvel
  154:   yvel = y_vel; // updated values of yvel
  155:
  156:   mass = m; // update mass
  157:   radius = rad; // update radius
  158:   winsize = window_size; // update window size
  159:   filename = name; // update filename
  160:
  161:   radx = (winsize / 2) * (xpos / radius) + (winsize / 2);
  162:   rady = (winsize / 2) * (ypos / radius) + (winsize / 2);
  163:
  164:
  165:   texture.loadFromFile(filename);
  166:
  167:
  168:   sprite.setTexture(texture);
  169:   sprite.setPosition(radx, rady);
  170: }
  171:
  172: istream &operator >>(istream &in, CelestialBody &ci){
  173:
  174:   double radx;
  175:   double rady;
  176:
  177:   in >> ci.xpos >> ci.ypos >> ci.xvel >> ci.yvel >> ci.mass >> ci.filenam
e;
  178:
  179:   radx = (ci.winsize / 2) * (ci.xpos / ci.radius) + (ci.winsize / 2);
  180:   rady = (ci.winsize / 2) * (ci.ypos / ci.radius) + (ci.winsize / 2);
  181:
  182:   ci.texture.loadFromFile(ci.filename);
  183:
  184:   ci.sprite.setTexture(ci.texture);
  185:   ci.sprite.setPosition(radx, rady);
  186:
  187:
  188:   return in; // return input
  189: }
```

```
  190:
  191: void CelestialBody::draw(sf::RenderTarget &target, sf::RenderStates state
s) const { target.draw(sprite, states); }
  192:
  193: CelestialBody::~CelestialBody(){}
  194:
  195: double CelestialBody::get_posx(){ return xpos; } // return the xpos
  196: double CelestialBody::get_posy(){ return ypos; } // return the ypos
  197: double CelestialBody::get_velx(){ return xvel; } // return the xvel
  198: double CelestialBody::get_vely(){ return yvel; } // return the yvel
  199: double CelestialBody::get_mass(){ return mass; } // return the mass
  200:
  201: string CelestialBody::get_filename(){ return filename; }
  202:
  203: void CelestialBody::set_radius(double r){ radius = r; }
  204: void CelestialBody::set_window(double size){ winsize = size; }
  205: void CelestialBody::set_velx(double vx){ xvel = vx; }
  206: void CelestialBody::set_vely(double vy){ yvel = vy; }
  207: void CelestialBody::set_x_y_pos(double x_input, double y_input){
  208:
  209:   double radx;
  210:   double rady;
  211:
  212:   xpos = x_input;
  213:   ypos = y_input;
  214:
  215:   radx = (winsize / 2) * (xpos / radius) + (winsize / 2);
  216:   rady = (winsize / 2) * (-ypos / radius) + (winsize / 2);
  217:
  218:   sprite.setPosition(sf::Vector2f(radx, rady));
  219: }
```