```
 1: CC= g++
 2: CFLAGS= -Wall -Werror -ansi -pedantic --std=c++11
 3: SFMLFLAGS= -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
 4:
 5:
 6: NBody: NBody.o
 7:         $(CC) NBody.o -o NBody $(SFMLFLAGS)
 8: NBody.o: universe.cpp  universe.hpp
 9:         $(CC) -c universe.cpp -o NBody.o  $(CFLAGS)
10:
11: clean:
12:         rm *.o
13:         rm NBody
14:
```

```
    1: #include <iostream>
    2: #include <vector>
    3: #include <SFML/System.hpp>
    4: #include <SFML/Window.hpp>
    5: #include <SFML/Graphics.hpp>
    6: #include <SFML/Audio.hpp>
    7: #include "universe.hpp"
    8: #include <memory>
    9: #include <cmath>
   10: using namespace std;
   11: using namespace sf;
   12:
   13: const double G = 6.67e-11;
   14: double deltaT;
   15: double T;
   16: universe::universe() {
   17:         return;
   18: }
   19:
   20: void universe::addBody(unique_ptr<bodies> body){
   21:         solarSystem.push_back(move(body));
   22:         return;
   23: }
   24:
   25: void universe::draw_universe(RenderWindow &window){
   26:         for(std::size_t i=0; i<solarSystem.size(); ++i) {
   27:                 window.draw(*(solarSystem.at(i)));
   28:         }
   29: }
   30:
   31: bodies::bodies() {
   32:         return;
   33: }
   34:
   35: void bodies::createBody(double rad, int universe_size){
   36:         double nxpos = ( (xpos / rad) * (universe_size / 2) ) + (universe_si
ze / 2);//(pos / rads ) * (window size/2)
   37:         double nypos = ( (ypos / rad) * (universe_size / 2) ) + (universe_si
ze / 2);
   38:         //sprite.setPosition(sf::Vector2f(xpos, ypos));
   39:         texture.loadFromFile(filename);
   40:         sprite.setTexture(texture);
   41:         sprite.setPosition(sf::Vector2f(nxpos, nypos));
   42:         cout<<xpos<<"   "<< ypos<<endl;
   43:
   44:         return;
   45: }
   46:
   47:
   48: void bodies::draw(sf::RenderTarget& target, sf::RenderStates blend) const{
   49: //      cout<<"flag"<<endl;
   50:         target.draw(sprite);
   51:         return;
   52: }
   53:
   54:
   55: void universe::update(){
   56:         double totalForce;
   57:         double r;
   58:         double deltax;
   59:         double deltay;
```

```
   60:         double accelx;
   61:         double accely;
   62:         for(std::size_t i=0; i<solarSystem.size(); ++i) {
   63:                 for(std::size_t j=0; j<solarSystem.size(); ++j) {
   64:                         if(i!=j){
   65:                                 deltax = solarSystem.at(j)->getxpos() -  sol
arSystem.at(i)->getxpos();
   66:                                 deltay = solarSystem.at(j)->getypos() -  sol
arSystem.at(i)->getypos();
   67:                                 r = hypot(deltax, deltay);
   68:                                 totalForce = ((G * solarSystem.at(i)->getmas
s() *  solarSystem.at(j)->getmass())/(r*r));//F=Gm1m2/r^2
   69:                                 solarSystem.at(i)->setFx(solarSystem.at(i)->
getFx() + (totalForce * (deltax / r)));
   70:                                 solarSystem.at(i)->setFy(solarSystem.at(i)->
getFy() + (totalForce * (deltay / r)));
   71:                         }
   72:                 }
   73:                 accelx = solarSystem.at(i)->getFx() / solarSystem.at(i)->get
mass();
   74:                 accely = solarSystem.at(i)->getFy() / solarSystem.at(i)->get
mass();
   75:                 solarSystem.at(i)->setxvel(solarSystem.at(i)->getxvel() + (d
eltaT * accelx));
   76:                 solarSystem.at(i)->setyvel(solarSystem.at(i)->getyvel() + (d
eltaT * accely));
   77:
   78:                  totalForce = 0;
   79:                  r = 0;
   80:                  deltax = 0;
   81:                  deltay = 0;
   82:                  accelx = 0;
   83:                  accely = 0;
   84:         }
   85:
   86: }
   87:
   88: void universe::travel(double rad, int universe_size){
   89:         for(std::size_t i=0; i<solarSystem.size(); ++i) {
   90:
   91:                 solarSystem.at(i)->setxpos(solarSystem.at(i)->getxpos() + (d
eltaT * solarSystem.at(i)->getxvel()));
   92:                 solarSystem.at(i)->setypos(solarSystem.at(i)->getypos() + (d
eltaT * solarSystem.at(i)->getyvel()));
   93:
   94:                 double nxpos = (        (solarSystem.at(i)->getxpos() / rad)
 *      (universe_size/2) + (universe_size / 2) );
   95:                 double nypos = (        (solarSystem.at(i)->getypos() / rad)
 *      (universe_size/2) + (universe_size / 2) );
   96:
   97:                 solarSystem.at(i)->sprite.setPosition(sf::Vector2f(nxpos, ny
pos));
   98:
   99:                 cout<< rad << endl;
  100:                 cout<<solarSystem.at(i)->getxpos() <<"     "<<solarSystem.at
(i)->getypos()<< "    " <<solarSystem.at(i)->getxvel() <<"     "<<solarSystem.at(i)-
>getyvel()<< endl;
  101:                 solarSystem.at(i)->setFx(0);
  102:                 solarSystem.at(i)->setFy(0);
  103:         }
  104:         cout<<endl;
```

```
105:
106: }
107:
108:
109:
110: int main(int argc, char *argv[]){
111:         //gets the heatdeath timer
112:         T = atof(argv[1]);
113:         deltaT = atof(argv[2]);
114:         cout<< T << "   "<<deltaT<<endl;
115:         //get the planet count from the the top of the planet file
116:         int planet_count;
117:         cin >> planet_count;
118:         cout<< planet_count<<endl;
119:
120:         //get the radius from the planet file
121:         double radius;
122:         cin >> radius;
123:         cout<< radius<<endl;
124:
125:         //setting up the viewing window with image background
126:         int size = 500;
127:         sf::RenderWindow window(sf::VideoMode(size, size), "Universe");
128:         sf::Music music;
129:         music.openFromFile("2001.wav");
130:         music.play();
131:
132:
133:         sf::Texture universal_Texture;
134:         universal_Texture.loadFromFile("starfield.jpg");
135:         sf::Sprite background;
136:         background.setTexture(universal_Texture);
137:         window.draw(background);
138:         universe system;
139:         //time to play god
140:         window.setFramerateLimit(60);
141:         for(int i=0; i<planet_count; i++){
142:                 std::unique_ptr<bodies> body(new bodies);
143:                 cin >> *body;
144:                 body->createBody(radius, size);
145:                 system.addBody(move(body));
146:         }
147:         system.draw_universe(window);
148:
149:         //window.display();
150:
151:         for(int i = 0; i<T; i+=deltaT){//run the physics
152:                 window.clear();
153:                 window.draw(background);
154:                 system.update();
155:                 system.travel(radius, size);
156:                 system.draw_universe(window);
157:                 window.display();
158:         }
159:                 window.display();
160:
161:         //window loop
162:         while (window.isOpen()){
163:                 sf::Event event;
164:                 while(window.pollEvent(event)){
165:                         if (event.type == sf::Event::Closed){
```

```
166:                            window.close();
167:                            }
168:                    }
169:            }
170:
171:            return 0;
172:
173: }
```

```
    1: #include <iostream>
    2: #include <vector>
    3: #include <SFML/System.hpp>
    4: #include <SFML/Window.hpp>
    5: #include <SFML/Graphics.hpp>
    6: #include <SFML/Audio.hpp>
    7: #include <memory>
    8: using namespace std;
    9: using namespace sf;
   10:
   11:
   12: class bodies : public sf::Drawable{
   13: public:
   14:         bodies();
   15:         sf::Sprite sprite;
   16:         //bodies(double xpossition, double ypossition, double xvelocity, dou
ble yvelocity, double size, std::string imagename);//construcot
   17:         friend std::istream& operator>> (std::istream &in, bodies& body){//i
nsertion for taking the data from inputfiles
   18:         in >> body.xpos;
   19:         in >> body.ypos;
   20:         in >> body.xvel;
   21:         in >> body.yvel;
   22:         in >> body.mass;
   23:         in >> body.filename;
   24:         body.Fx = 0;
   25:         body.Fy = 0;
   26:         return in;
   27:         }
   28:         void virtual draw(sf::RenderTarget& target, sf::RenderStates blend)
const;
   29:         void createBody(double rad, int universe_size);
   30:         double getxpos(){
   31:                 return xpos;
   32:         }
   33:         double getypos(){
   34:                 return ypos;
   35:         }
   36:         double getxvel(){
   37:                 return xvel;
   38:         }
   39:         double getyvel(){
   40:                 return yvel;
   41:         }
   42:         double getmass(){
   43:                 return mass;
   44:         }
   45:
   46:         double getFx(){
   47:                 return Fx;
   48:         }
   49:         double getFy(){
   50:                 return Fy;
   51:         }
   52:
   53:         void setxpos(double x){
   54:                 xpos = x;
   55:                 return;
   56:         }
   57:         void setypos(double y){
   58:                 ypos = y;
```

```
    59:                 return;
    60:         }
    61:         void setxvel(double x){
    62:                 xvel = x;
    63:                 return;
    64:         }
    65:         void setyvel(double y){
    66:                 yvel = y;
    67:                 return;
    68:         }
    69:         void setspritePos(double x, double y){
    70:                 sprite.setPosition(sf::Vector2f(x, y));
    71:                 return;
    72:         }
    73:         void setFx(double x){
    74:                 Fx = x;
    75:                 return;
    76:         }
    77:         void setFy(double y){
    78:                 Fy = y;
    79:                 return;
    80:         }
    81: private:
    82:         double xpos;
    83:         double ypos;
    84:         double xvel;
    85:         double yvel;
    86:         double mass;
    87:         std::string filename;
    88:         Texture texture;
    89:         //sf::Sprite sprite;
    90:         double Fx;
    91:         double Fy;
    92: };
    93:
    94:
    95:
    96:
    97: class universe : public bodies{
    98: public:
    99:         universe();
   100:         void addBody(unique_ptr<bodies> body);
   101:         void draw_universe(RenderWindow &window);
   102:         void update();
   103:         void travel(double rad, int universe_size);
   104: private:
   105:         vector <unique_ptr<bodies>> solarSystem;
   106:
   107: };
   108:
```