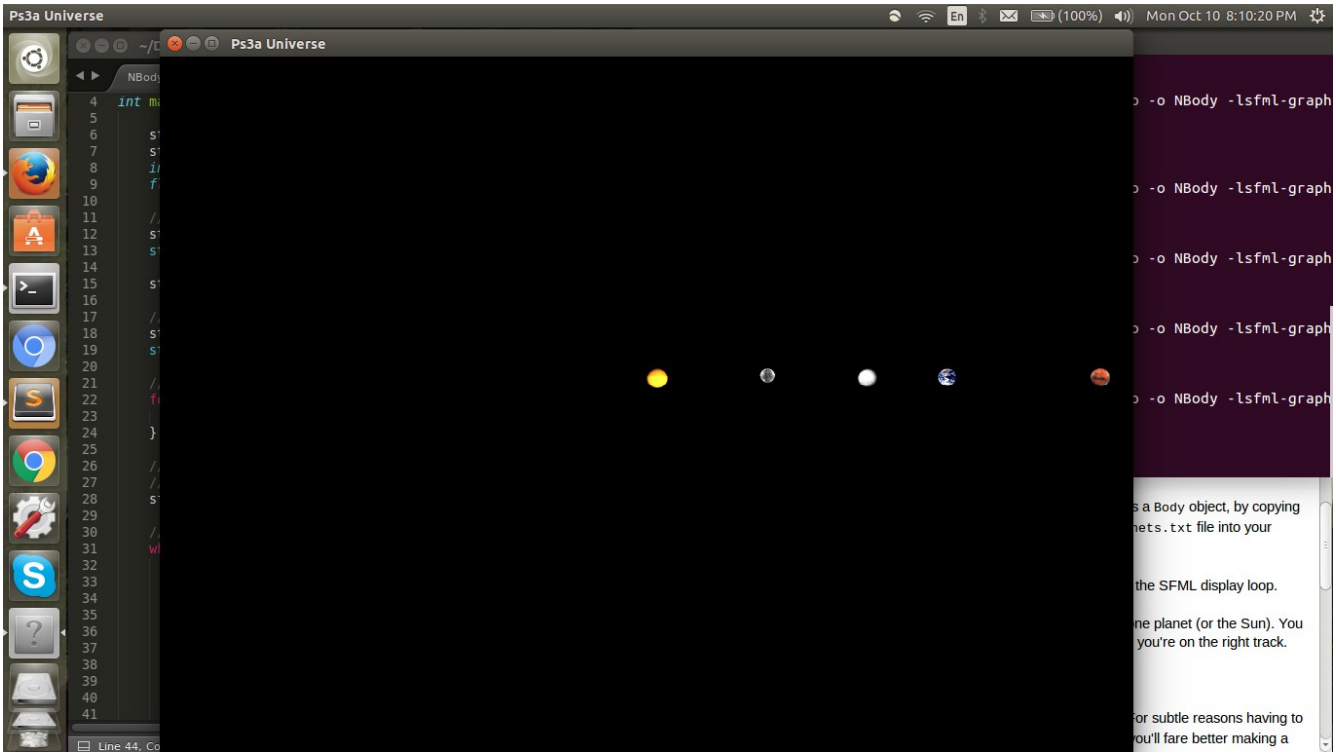
PS3: *N-Body Simulation*

Ps3a:

In the Ps3a, the assignment was to be able to make a static solar system using the coordinates from a text file. To accomplish this, we need to read each line and save each string into the associated variable. While doing that we would use the operator to be able to do that.

I used type vectors for the position and velocity because both had a x and y coordinate. I created a global variable that was the scale. We also needed to set each position by taking the screen size and setting each planet to the right position.

I learned that even with though the planets position numbers were huge, it could easily be placed by multiplying the position by scale and by the size of the window. I also learned that cin can actually read in a string and could read the next string just by using it again. This is done by giving the program a text file to read in.



```
    1: cc = g++
    2:
    3:
    4: all : NBody_main
    5:
    6: NBody_main : NBody_main.o NBody.o
    7:         $(cc) -Wall -Werror -ansi -pedantic NBody_main.cpp NBody.cpp -o NBod
y -lsfml-graphics -lsfml-window -lsfml-system
    8:
    9: NBody_main.o : NBody.hpp
   10:         $(cc) -c NBody_main.cpp -o NBody_main.o
   11:
   12: NBody : NBody.cpp NBody.hpp
   13:         $(cc) -c NBody.cpp -o NBody.o
   14:
   15: clean:
   16:         rm *.o NBody
```

```
    1: #include "NBody.hpp"
    2:
    3:
    4: int main(int argc, char* argv[]){
    5:
    6:     std::string store;//stores the input from the file
    7:     std::string name;//name of planet
    8:     int numberOfPlanets;
    9:     float radius;//radius of window
   10:
   11:     //stores the number of planets
   12:     std::cin >> store;
   13:     std::stringstream(store) >> numberOfPlanets;
   14:
   15:     std::vector<Body> objects(numberOfPlanets);//vector of objects to store
all objects
   16:
   17:     //stores the radius of the window
   18:     std::cin >> store;
   19:     std::stringstream(store) >> radius;
   20:
   21:     //loop that stores all relevant data from the file
   22:     for (int x = 0; x < numberOfPlanets; x++){
   23:         std::cin >> objects[x];
   24:     }
   25:
   26:     //take the data inside the
   27:     //vector of bodies and print it on the screen using SFML
   28:     sf::RenderWindow window(sf::VideoMode(1000, 800), "Ps3a Universe");
   29:
   30:     //display window
   31:     while(window.isOpen()){
   32:         sf::Event event;
   33:         while(window.pollEvent(event)){
   34:             if(event.type == sf::Event::Closed)
   35:                 window.close();
   36:         }
   37:         window.clear(sf::Color::Black);
   38:         //display all the planets
   39:         for(int i = 0; i < numberOfPlanets; i++){
   40:             window.draw(objects[i]);
   41:         }
   42:         window.display();
   43:     }
   44:
   45:     return 0;
   46: }
```

```
    1: #ifndef NBODY_H
    2: #define NBODY_H
    3:
    4:
    5: #include <SFML/Graphics.hpp>
    6: #include <SFML/Window.hpp>
    7: #include <iostream>
    8: #include <string>
    9: #include <vector>
   10: #include <sstream>
   11:
   12:
   13: class Body : public sf::Drawable
   14: {
   15:  private:
   16:   double _time;
   17:   sf::Vector2f _position;
   18:   sf::Vector2f _velocity;
   19:   float _mass;
   20:   std::string _filename;
   21:
   22:  public:
   23:   Body(float xCoord, float yCoord, float xVelocity, float yVelocity, float m
ass, std::string fileName);
   24:
   25:   Body();
   26:   //friend istream &operator>>( istream &input, const Body &B);
   27:   virtual void draw(sf::RenderTarget &target, sf::RenderStates states) const
;
   28:
   29:   //input stream overloader
   30:   friend std::istream& operator>>(std::istream& in, Body& Body);
   31:
   32:   void setTime(double sTime);
   33:   double getTime();
   34:
   35:   void setPosition(sf::Vector2f sPosition);
   36:   sf::Vector2f getPosition();
   37:
   38:   void setVel(sf::Vector2f sVel);
   39:   sf::Vector2f getVel();
   40:
   41:   void setMass(float sMass);
   42:   float getMass();
   43:
   44:   void setFilename(std::string sFile);
   45:   std::string getFilename();
   46:
   47:   ~Body();
   48: };
   49:
   50: #endif
```

```
    1: #include "NBody.hpp"
    2: //.000000002
    3: const float SCALE = (2.50e+11);//This number negates the e+10 in the x posit
ion.
    4:
    5: void Body::draw(sf::RenderTarget &target, sf::RenderStates states) const{
    6:   sf::Image image;
    7:   sf::Texture texture;
    8:   sf::Sprite sprite;
    9:
   10:   //std::cout << _position.x << _position.y << _filename << std::endl;
   11:
   12:   if(!image.loadFromFile(_filename)){
   13:       std::cout << "ERROR: could not load image from file" << std::endl;
   14:       return;
   15:     }
   16:
   17:     texture.loadFromFile(_filename);
   18:     sprite.setTexture(texture);
   19:
   20:     //need to multiply the x position by SCALE so the planets are not off th
e screen
   21:     sprite.setPosition((_position.x/SCALE) * 500 + target.getSize().x/2,(_po
sition.y/SCALE) * 400 + target.getSize().y/2);
   22: //x position / universe size * window size
   23:     target.draw(sprite);
   24:
   25: }
   26:
   27: std::istream& operator>>(std::istream& in, Body& body){
   28:
   29:   in >> body._position.x  >> body._position.y >> body._velocity.x >> body._v
elocity.y >> body._mass >> body._filename;
   30:
   31:   return in;
   32:
   33: }
   34:
   35: Body::Body(float xCoord, float yCoord, float xVelocity, float yVelocity, flo
at mass, std::string fileName){
   36:   //setting vars to specifications
   37:   _position.x = xCoord;
   38:   _position.y = yCoord;
   39:   _velocity.x = xVelocity;
   40:   _velocity.y = yVelocity;
   41:   _mass = mass;
   42:   _filename = fileName;
   43:
   44: }
   45: Body::Body(){
   46:
   47: }
   48:
   49:  sf::Vector2f Body::getPosition(){
   50:   return _position;
   51:  }
   52:
   53:  sf::Vector2f Body::getVel(){
   54:   return _velocity;
   55:  }
   56:
```

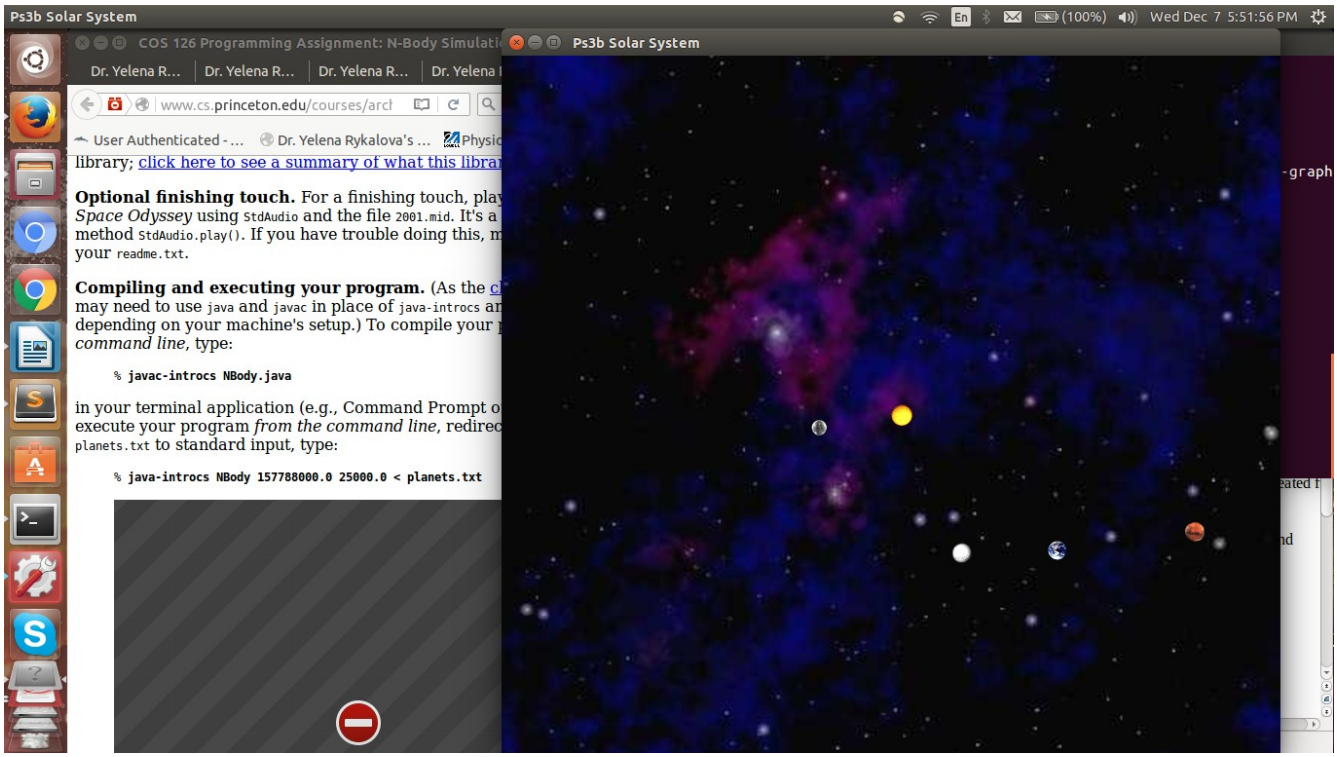
```
   57:  float Body::getMass(){
   58:   return _mass;
   59:  }
   60:
   61:  std::string Body::getFilename(){
   62:   return _filename;
   63:  }
   64:
   65:  Body::~Body(){
   66:
   67:  }
   68:
   69: //in main, before we make a new Body, we read in the file name that has the
proper characteristics, save them into multiple vars or strings, then feed them int
o the Body constructor
```

Ps3b:

In the Ps3b, this assignment was to able move the planets by giving the planets a acceleration and velocity. To accomplish this, we needed to use the Pairwise force, net force, and acceleration equations.

I used the equations to figure out what the acceleration was from each planet from the sun. Giving each planets its assigned velocity, position, mass, and the name of the file. After storing each planets assigned values, I stored each planet in a vector of the number of planets. I created functions for the force, acceleration, and radius.

I never took physics ever in my high school. So learning about the right equations to figure out the net force and the acceleration. Learning to play sound once the window was open was useful and I will use in my later projects. Also, I learned that a vector can take a any type. For example, its possible to allocate a vector of objects.



```
    1: cc = g++
    2:
    3:
    4: all : NBody_main
    5:
    6: NBody_main : NBody_main.o NBody.o
    7:         $(cc) -Wall -Werror -ansi -pedantic NBody_main.cpp NBody.cpp -o NBod
y -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
    8:
    9: NBody_main.o : NBody.hpp
   10:         $(cc) -c NBody_main.cpp -o NBody_main.o
   11:
   12: NBody : NBody.cpp NBody.hpp
   13:         $(cc) -c NBody.cpp -o NBody.o
   14:
   15: clean:
   16:         rm *.o NBody
```

```
    1: #include "NBody.hpp"
    2:
    3: double radius(sf::Vector2f pos1, sf::Vector2f pos2);
    4: sf::Vector2f force(double mass1, double mass2, double radius, sf::Vector2f d
elta_p);
    5: sf::Vector2f accel(double mass, sf::Vector2f p_force);
    6: sf::Vector2f changeInPosition(sf::Vector2f pos1, sf::Vector2f pos2);
    7:
    8: const double G = (6.67e-11);
    9:
   10: int main(int argc, char* argv[]){
   11:
   12:
   13:     if (argc < 3){
   14:         std::cout << "max_time , time increment" << std::endl;
   15:         return -1;
   16:     }
   17:
   18:     double max_time = atof(argv[1]);
   19:     double step_time = atof(argv[2]);
   20:     double start_time = 0;
   21:
   22:     std::string store;//stores the input from the file
   23:     std::string name;//name of planet
   24:     int numberOfPlanets;
   25:     double radius_of_window;//radius of window
   26:     sf::Vector2f tempForce;
   27:     double c_radius;
   28:     sf::Vector2f c_force;
   29:     sf::Vector2f c_accel;
   30:     sf::Vector2f delta_p;
   31:
   32:     //stores the number of planets
   33:     std::cin >> store;
   34:     std::stringstream(store) >> numberOfPlanets;
   35:
   36:     std::vector<Body> objects(numberOfPlanets);//vector of objects to store
all objects
   37:
   38:     //stores the radius of the window
   39:     std::cin >> store;
   40:     std::stringstream(store) >> radius_of_window;
   41:
   42:     //loop that stores all relevant data from the file
   43:     for (int x = 0; x < numberOfPlanets; x++){
   44:         std::cin >> objects[x];
   45:     }
   46:
   47:     //take the data inside the
   48:     //vector of bodies and print it on the screen using SFML
   49:     sf::RenderWindow window(sf::VideoMode(800, 800), "Ps3b Solar System");
   50:
   51:     sf::Image background;
   52:     if(!background.loadFromFile("starfield.jpg"))
   53:         return -1;
   54:     sf::Texture backtex;
   55:     backtex.loadFromImage(background);
   56:
   57:     sf::Sprite backsprite;
   58:     backsprite.setTexture(backtex);
   59:
```

```
   60:     //plays audio
   61:     sf::SoundBuffer buff;
   62:     buff.loadFromFile("st.wav");
   63:     sf::Sound sound;
   64:     sound.setBuffer(buff);
   65:     sound.play();
   66:
   67:     //backsprite.setScale(500,500);
   68:
   69:     //display window
   70:     while(window.isOpen()){
   71:         sf::Event event;
   72:         while(window.pollEvent(event)){
   73:             if(event.type == sf::Event::Closed)
   74:                 window.close();
   75:         }
   76:         window.clear();
   77:         window.draw(backsprite);
   78:         //as long as we don't go past the max time
   79:         if(start_time < max_time){
   80:             //display all the planets
   81:             for(int i = 0; i < numberOfPlanets; i++){
   82:                 window.draw(objects[i]);
   83:             }
   84:             //for every planet
   85:             for(int i = 0; i < numberOfPlanets; i++){
   86:                 //don't do it for the current planet
   87:                 if(i != 3){
   88:                     delta_p = changeInPosition(objects[i].getPosition(),obje
cts[3].getPosition());
   89:                     c_radius = radius(objects[i].getPosition(),objects[3].ge
tPosition());
   90:                     c_force = force(objects[i].getMass(), objects[3].getMass
(),c_radius, delta_p);
   91:                     c_accel = accel(objects[i].getMass(), c_force);
   92:                     objects[i].setAccel(c_accel);
   93:                     objects[i].step(step_time);
   94:                 }
   95:             }
   96:             //run step() to calculate the new positions, update start_time
   97:             start_time += step_time;
   98:
   99:         }
  100:         window.display();
  101:     }
  102:
  103:     return 0;
  104: }
  105:
  106: sf::Vector2f changeInPosition(sf::Vector2f pos1, sf::Vector2f pos2){
  107:     sf::Vector2f change_p;
  108:
  109:     change_p.x = pos2.x - pos1.x;
  110:     change_p.y = pos2.y - pos1.y;
  111:
  112:     return change_p;
  113: }
  114:
  115: double radius(sf::Vector2f pos1, sf::Vector2f pos2){
  116:   return std::sqrt(std::pow(pos1.x - pos2.x,2) + std::pow(pos1.y - pos2.y,2)
);
```

```
  117: }
  118:
  119: sf::Vector2f force(double mass1, double mass2, double radius,sf::Vector2f de
lta_p){
  120:     double F = (G * (mass1 * mass2)/(std::pow(radius,2)));
  121:
  122:     sf::Vector2f f_temp;
  123:
  124:     f_temp.x = F * (delta_p.x/radius);
  125:     f_temp.y = F * (delta_p.y/radius);
  126:
  127:     return f_temp;
  128:
  129:
  130: }
  131: sf::Vector2f accel(double mass, sf::Vector2f p_force){
  132:     sf::Vector2f cook;
  133:
  134:     cook.x = p_force.x/mass;
  135:     cook.y = p_force.y/mass;
  136:
  137:     return cook;
  138: }
```

```
    1: #ifndef NBODY_H
    2: #define NBODY_H
    3:
    4:
    5: #include <SFML/Graphics.hpp>
    6: #include <SFML/Window.hpp>
    7: #include <SFML/Audio.hpp>
    8: #include <iostream>
    9: #include <string>
   10: #include <vector>
   11: #include <sstream>
   12: #include <cmath>
   13:
   14: class Body : public sf::Drawable
   15: {
   16:  private:
   17:   double _time;
   18:   sf::Vector2f _position;
   19:   sf::Vector2f _velocity;
   20:   sf::Vector2f _acceleration;
   21:   double _mass;
   22:   std::string _filename;
   23:   sf::Vector2f _netforce;
   24:   int _numberOfPlanets;
   25:   sf::Vector2f _updatedAcceleration;
   26:   sf::Vector2f _updatedPosition;
   27:   sf::Vector2f _updatedVelocity;
   28:
   29:  public:
   30:   Body(double xCoord, double yCoord, double xVelocity, double yVelocity, dou
ble mass, std::string fileName);
   31:
   32:   Body();
   33:   //friend istream &operator>>( istream &input, const Body &B);
   34:   virtual void draw(sf::RenderTarget &target, sf::RenderStates states) const
;
   35:
   36:   //input stream overloader
   37:   friend std::istream& operator>>(std::istream& in, Body& Body);
   38:
   39:   void step(double t_time);
   40:
   41:   void setNetforce(sf::Vector2f sNetforce);
   42:   sf::Vector2f getNetforce();
   43:
   44:   void setNumPlanets(int sNoplanets);
   45:   int getNumPlanets();
   46:
   47:   void setupdatePos(sf::Vector2f sUpdatepos);
   48:   sf::Vector2f getupdatePos();
   49:
   50:   /*void Body::setupdateVel(sf::Vector2f sUpdateVel);
   51:   sf::Vector2f getupdateVel();*/
   52:
   53:   /*void setupdatedAccel(sf::Vector2f sUpdatedAccel);
   54:   sf::Vector2f getupdatedAccel();*/
   55:
   56:   void setTime(double sTime);
   57:   double getTime();
   58:
   59:   void setPosition(sf::Vector2f sPosition);
```

```
   60:   sf::Vector2f getPosition();
   61:
   62:   void setVel(sf::Vector2f sVel);
   63:   sf::Vector2f getVel();
   64:
   65:   void setAccel(sf::Vector2f sAccel);
   66:   sf::Vector2f getAccel();
   67:
   68:   void setMass(double sMass);
   69:   double getMass();
   70:
   71:   void setFilename(std::string sFile);
   72:   std::string getFilename();
   73:
   74:   ~Body();
   75: };
   76:
   77: #endif
```

```
    1: #include "NBody.hpp"
    2: //.000000002
    3: const double SCALE = (3.5e+11);//This number negates the e+10 in the x posit
ion.
    4: const double G = (6.67e-11);
    5: void Body::draw(sf::RenderTarget &target, sf::RenderStates states) const{
    6:   sf::Image image;
    7:   sf::Texture texture;
    8:   sf::Sprite sprite;
    9:
   10:   //std::cout << _position.x << _position.y << _filename << std::endl;
   11:
   12:   if(!image.loadFromFile(_filename)){
   13:       std::cout << "ERROR: could not load image from file" << std::endl;
   14:       return;
   15:     }
   16:
   17:     texture.loadFromFile(_filename);
   18:     sprite.setTexture(texture);
   19:
   20:     //need to multiply the x position by SCALE so the planets are not off th
e screen
   21:     sprite.setPosition((_position.x/SCALE) * 500 + target.getSize().x/2,(_po
sition.y/SCALE) * 500 + target.getSize().y/2);
   22: //x position / universe size * window size
   23:     target.draw(sprite);
   24:
   25: }
   26:
   27: std::istream& operator>>(std::istream& in, Body& body){
   28:
   29:   in >> body._position.x  >> body._position.y >> body._velocity.x >> body._v
elocity.y >> body._mass >> body._filename;
   30:
   31:   return in;
   32:
   33: }
   34:
   35: Body::Body(double xCoord, double yCoord, double xVelocity, double yVelocity,
 double mass, std::string fileName){
   36:   //setting vars to specifications
   37:   _position.x = xCoord;
   38:   _position.y = yCoord;
   39:   _velocity.x = xVelocity;
   40:   _velocity.y = yVelocity;
   41:   _mass = mass;
   42:   _filename = fileName;
   43:
   44: }
   45: Body::Body(){
   46:
   47: }
   48:
   49: void Body::step(double t_time){
   50:
   51:
   52:   sf::Vector2f distance;
   53:   sf::Vector2f o_accel = getAccel();
   54:   sf::Vector2f oldVel = getVel();
   55:   sf::Vector2f newVel;
   56:   sf::Vector2f endVel;
```

```
   sf::Vector2f oldPosition = getPosition();
   sf::Vector2f newPosition;

   distance.x = t_time * oldVel.x;
   distance.y = t_time * oldVel.y;

   newPosition.x = distance.x + oldPosition.x;
   newPosition.y = distance.y + oldPosition.y;

   newVel.x = o_accel.x * t_time;
   newVel.y = o_accel.y * t_time;

   endVel.x = newVel.x + oldVel.x;
   endVel.y = newVel.y + oldVel.y;


   setPosition(newPosition);

   setVel(endVel);


}


   void Body::setNetforce(sf::Vector2f sNetforce){
     _netforce = sNetforce;
   }
   sf::Vector2f Body::getNetforce(){
     return _netforce;
   }

   void Body::setNumPlanets(int sNoplanets){
     _numberOfPlanets = sNoplanets;
   }
   int Body::getNumPlanets(){
     return _numberOfPlanets;
   }

   void Body::setupdatePos(sf::Vector2f sUpdatepos){
     _updatedPosition = sUpdatepos;
   }
   sf::Vector2f Body::getupdatePos(){
     return _updatedPosition;
   }



   void Body::setPosition(sf::Vector2f sPosition){
    _position = sPosition;
   }

  sf::Vector2f Body::getPosition(){
   return _position;
  }

  void Body::setVel(sf::Vector2f sVelocity){
   _velocity = sVelocity;
  }

  sf::Vector2f Body::getVel(){
   return _velocity;
```

```
  118:  }
  119:
  120:   void Body::setAccel(sf::Vector2f sAccel){
  121:    _acceleration = sAccel;
  122:   }
  123:
  124: sf::Vector2f Body::getAccel(){
  125:   return _acceleration;
  126: }
  127:
  128: void Body::setMass(double sMass){
  129:   _mass = sMass;
  130: }
  131:
  132: double Body::getMass(){
  133:   return _mass;
  134:  }
  135:
  136: void Body::setFilename(std::string sFile){
  137:     _filename = sFile;
  138:   }
  139:
  140:  std::string Body::getFilename(){
  141:   return _filename;
  142:  }
  143:
  144:  Body::~Body(){
  145:
  146:  }
  147:
  148: //in main, before we make a new Body, we read in the file name that has the
proper characteristics, save them into multiple vars or strings, then feed them int
o the Body constructor
```