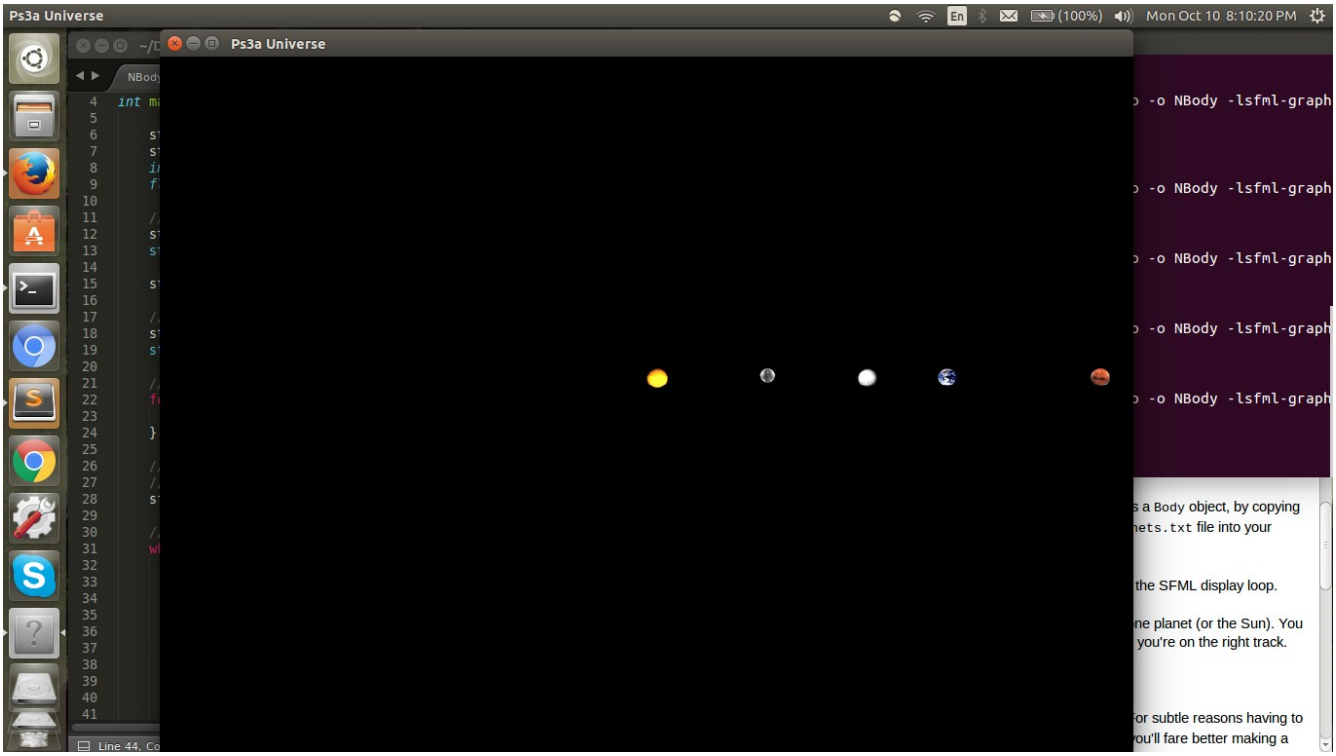
PS3: *N-Body Simulation*

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
```