

```
1: CC = g++
2: CFLAGS = -Wall -Werror -pedantic --std=c++14
3: LIBS = -lboost_unit_test_framework
4: DEPS = CelestialBody.h Universe.h
5: SFMLFLAGS = -lsfml-graphics -lsfml-window -lsfml-system
6:
7: %.o: %.cpp $(DEPS)
8:     $(CC) $(CFLAGS) -c $<
9:
10: all: NBody
11:
12: NBody: main.o CelestialBody.o Universe.o
13:     $(CC) $(CFLAGS) -o NBody $^ $(LIBS) $(SFMLFLAGS)
14:
15: clean:
16:     rm *.o NBody
```

```
1: /**
2:  * main.cpp - as a base to run the program
3:  *
4:  * Date 2/14/22 - 2/22/22
5:  *
6:  * Created by: Anson Cheang
7:  *
8:  */
9:
10: /*#include <SFML/System.hpp>
11: #include <SFML/Window.hpp>
12: #include <SFML/Graphics.hpp>*/
13: #include "CelestialBody.h"
14: #include "Universe.h"
15: #include <iostream>
16: #include <cstdlib>
17:
18: using namespace std;
19:
20: int main(int argc, char* argv[])
21: {
22:     double time = atoi(argv[1]);
23:     double seconds = 0;
24:     sf::RenderWindow window(sf::VideoMode(700, 700), "Input");
25:
26:     window.setVerticalSyncEnabled(true);
27:     window.setFramerateLimit(15);
28:
29:     sf::Image image;
30:     if(!image.loadFromFile("starfield.jpg"))
31:     {
32:         return -1;
33:     }
34:     sf::Texture texture;
35:     texture.loadFromImage(image);
36:     sf::Sprite sprite;
37:     sprite.setTexture(texture);
38:     sf::Vector2u size = image.getSize();
39:     sprite.setScale((1+700/size.x), (1+700/size.y));
40:
41:     int amount;
42:
43:     cin >> amount;
44:
45:     Universe space(amount);
46:
47:     while (window.isOpen())
48:     {
49:         sf::Event event;
50:         while (window.pollEvent(event))
51:         {
52:             if (event.type == sf::Event::Closed)
53:             {
54:                 window.close();
55:             }
56:         }
57:
58:         window.clear();
59:         window.draw(sprite);
60:         if(seconds <= time)
61:         {
62:             space.step(atoi(argv[2]));
63:             seconds += atoi(argv[2]);
64:         }
65:         window.draw(space);
```

```
66:             window.display();
67:         }
68:     cout << space;
69:     return 0;
70: }
```

```
1: #ifndef Universe_H_
2: #define Universe_H_
3:
4: #include <SFML/System.hpp>
5: #include <SFML/Window.hpp>
6: #include <SFML/Graphics.hpp>
7: #include <vector>
8: #include <iostream>
9: #include "CelestialBody.h"
10:
11: using namespace std;
12:
13: class Universe : public sf::Drawable
14: {
15: public:
16:     Universe(int size);
17:     void step(double seconds);
18:     friend ostream& operator<<(ostream& out, const Universe& Galaxy);
19: private:
20:
21:     void draw(sf::RenderTarget& target, sf::RenderStates states) const;
22:     int galaxySize;
23:     double maxR;
24:     vector<unique_ptr<CelestialBody> > galaxy;
25: };
26:
27: ostream& operator<<(ostream& out, const Universe& Galaxy);
28:
29: #endif
```

```
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: }
```

```
1: #ifndef CelestialBody_H_
2: #define CelestialBody_H_
3:
4: #include <SFML/System.hpp>
5: #include <SFML/Window.hpp>
6: #include <SFML/Graphics.hpp>
7: #include <string>
8: #include <cstdlib>
9: #include <iostream>
10:
11: using namespace std;
12:
13: class CelestialBody : public sf::Drawable
14: {
15: public:
16:
17:     CelestialBody(double val);
18:     void createImage();
19:
20:     CelestialBody(double posX, double posY, double Xvel, double Yvel, double
Imass, string _filename);
21:     friend istream& operator>>(istream& instream, CelestialBody& planet);
22:     friend ostream& operator<<(ostream& out, CelestialBody planet);
23:
24:     void setPos(sf::Vector2f Pos);
25:     void setVel(sf::Vector2f Vel);
26:     void setImagePos();
27:     double getXPos();
28:     double getYPos();
29:     double getMass();
30:     double getXVel();
31:     double getYVel();
32:
33: private:
34:
35:     void draw(sf::RenderTarget& target, sf::RenderStates states) const;
36:     double XPosition;
37:     double YPosition;
38:     double XVelocity;
39:     double YVelocity;
40:     double Mass;
41:     double scale;
42:     string filename;
43:     sf::Image image;
44:     sf::Texture texture;
45:     sf::Sprite sprite;
46: };
47:
48:
49: istream& operator>>(istream& instream, CelestialBody& planet);
50: ostream& operator<<(ostream& out, CelestialBody planet);
51:
52: #endif
```

```
1: /**
2:  * CelestialBody.cpp - an implementation to create each celestial body
3:  * 1 at a time, and also place them into the correct location
4:  * for drawing. plus draw each one individually, and overode >> operator
5:  *
6:  * Date 2/14/22 - 2/22/22
7:  *
8:  * Created by: Anson Cheang
9:  *
10: */
11:
12: #include "CelestialBody.h"
13: #include <SFML/System.hpp>
14: #include <SFML/Window.hpp>
15: #include <SFML/Graphics.hpp>
16: #include <string>
17: #include <cstdlib>
18: #include <iostream>
19:
20: using namespace std;
21:
22: CelestialBody::CelestialBody(double val)
23: {
24:     scale = val;
25:     XPosition = 0;
26:     YPosition = 0;
27:     XVelocity = 0;
28:     YVelocity = 0;
29:     Mass = 0;
30:     filename = "";
31: }
32:
33:
34: void CelestialBody::createImage()
35: {
36:     if(!image.loadFromFile(filename))
37:     {
38:         exit(-1);
39:     }
40:
41:     texture.loadFromImage(image);
42:
43:     sprite.setTexture(texture);
44:     sf::Vector2u size = image.getSize();
45:     sprite.setOrigin(static_cast<int>(size.x)/2, static_cast<int>(size.y)
/2);
46:     sprite.setPosition(sf::Vector2f(XPosition*scale + 350, YPosition*scale
e + 350));
47: }
48:
49: CelestialBody::CelestialBody(double posX, double posY, double Xvel, doubl
e Yvel, double Imass, string _filename)
50: {
51:     XPosition = posX;
52:     YPosition = posY;
53:     XVelocity = Xvel;
54:     YVelocity = Yvel;
55:     Mass = Imass;
56:     filename = _filename;
57:
58:     if(!image.loadFromFile(filename))
59:     {
60:         exit(-1);
61:     }
62:
```



```
63:     texture.loadFromImage(image);
64:
65:     sprite.setTexture(texture);
66:     sf::Vector2u size = image.getSize();
67:     sprite.setOrigin(static_cast<int>(size.x)/2, static_cast<int>(size.y)
/2);
68:     sprite.setPosition(sf::Vector2f(posX, posY));
69: }
70:
71: void CelestialBody::draw(sf::RenderTarget& target, sf::RenderStates state
s) const
72: {
73:     target.draw(sprite, states);
74: }
75:
76: istream& operator>>(istream& instream, CelestialBody& planet)
77: {
78:     instream >> planet.XPosition >> planet.YPosition >> planet.XVelocity
>> planet.YVelocity >> planet.Mass >> planet.filename;
79:     return instream;
80: }
81:
82: void CelestialBody::setPos(sf::Vector2f Pos)
83: {
84:     XPosition = Pos.x;
85:     YPosition = Pos.y;
86: }
87:
88: void CelestialBody::setVel(sf::Vector2f Vel)
89: {
90:     XVelocity = Vel.x;
91:     YVelocity = Vel.y;
92: }
93:
94: double CelestialBody::getXPos()
95: {
96:     return XPosition;
97: }
98:
99: double CelestialBody::getYPos()
100: {
101:     return YPosition;
102: }
103:
104: double CelestialBody::getMass()
105: {
106:     return Mass;
107: }
108:
109: void CelestialBody::setImagePos()
110: {
111:     //double CX = (Pos.x - XPosition) * scale;
112:     //double CY = (Pos.y - YPosition) * scale;
113:     sprite.setPosition(sf::Vector2f(XPosition*scale + 350, YPosition*scale
e + 350));
114:     //cout << XPosition*scale + 350 << ", " << YPosition*scale + 350 << e
ndl;
115:     //XPosition = Pos.x;
116:     //YPosition = Pos.y;
117: }
118:
119: double CelestialBody::getXVel()
120: {
121:     return XVelocity;
122: }
```

```
123:
124: double CelestialBody::getYVel()
125: {
126:     return YVelocity;
127: }
128:
129:
130: ostream& operator<<(ostream& out, CelestialBody planet)
131: {
132:     out << planet.XPosition << " " << planet.YPosition << " " << planet
.XVelocity << " "
133:         << planet.YVelocity << " " << planet.Mass << " " << planet.file
name << endl;
134:     return out;
135: }
```