

```
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: }
```