

PS2 N-Body Simulation

The purpose of this project was to mimic how pairwise forces affect particles in a universe. PS2a and PS2b were the two parts of this task. The first section of the task required you to create a representation of our galaxy using two classes: one to represent each planet and another to govern each planet. The initial part's sole purpose was to ensure that planets could be read from a text file and presented appropriately in the SFML window. The second section of the project involves simulating the effects of paired force using accurate physics calculations. The main function was required to accept data in the form of overall simulation time, time spent in one simulation step, and the input file containing information about each planet.

Key Concepts

The Universe class may be overloaded to allow for input redirection from a file to set up the planets for simulation. `CelestialBody` and `Universe` are both `sf::Drawable`, allowing them to be drawn to the window. Within the `Universe` class, each `CelestialBody` is created with a `std::unique_ptr` and stored in a `std::vector`. The planets are then rendered using the `draw` function on each `CelestialBody` in the vector in a loop in `main`. The majority of the calculations for this program are done in `Universe's step()` function.

What I accomplished

PS2a: `Universe` is a class which contains a vector of `CelestialBody(s)` and has a function to draw the planets by accessing a vector. For extra credit, I added a background image called "A Small Glimpse of The Cosmos:" and used smart pointers to avoid data leaks. The class `Universe` was a vector of `CelestialBody` which was used to hold all the planets that were fed into the program from `planets.txt`. I used a smart pointer to represent the universe, closing off the possibilities of memory issues(`unique_ptr <universe> u(new universe());`). The universe was allocated through a for loop. I had to make a pushback function in order to assign `CelestialBody(s)` to the vector.

PS2b: The `step` function was implemented in reference to the homework pdf. The `step` function is fed all the necessary forces via the file. Through each loop of said function, it sorts through and initializes each planet with the necessary x and y positions and velocities, acceleration, net force, force, gravitational force, and mass. Smart pointers were used to create new objects and pushback. Everything else was pretty much piggy backed off part a. I did fix my `Universe` class and implemented it properly. Though, looking back at it, I probably should still have kept it as its own header.

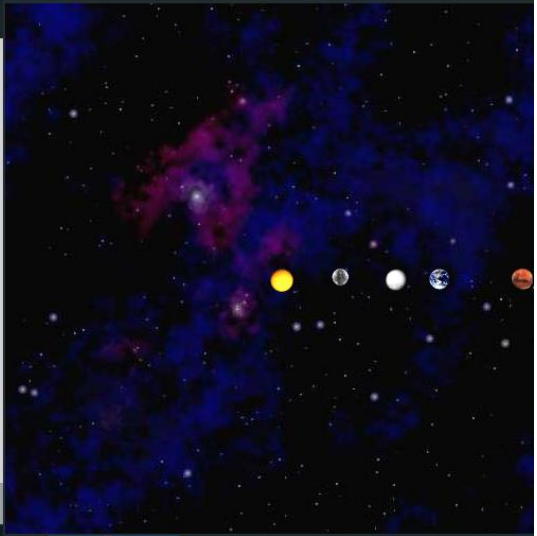
What I Learned

This task taught me a great deal. For starters, this was the first time I utilized smart pointers to initialize objects in a C++ application. I gained a lot of knowledge about the physics involved in the N-Body simulation utilizing the leapfrog finite difference approximation approach in addition to C++.

Output

```
Linux Lite Terminal -
File Edit View Terminal Tabs Help
current x Position: 3.079000e+02
current y Position: 2.500000e+02
current x Velocity: 0.000000e+00
current y Velocity: 4.790000e+04
Particle Mass: 3.302000e+23
Particle Name: mercury.gif
current x Position: 2.500000e+02
current y Position: 2.500000e+02
current x Velocity: 0.000000e+00
current y Velocity: 0.000000e+00
Particle Mass: 1.989000e+30
Particle Name: sun.gif
current x Position: 3.582000e+02
current y Position: 2.500000e+02
current x Velocity: 0.000000e+00
current y Velocity: 3.500000e+04
Particle Mass: 4.869000e+24
Particle Name: venus.gif
Loop # 0
Loop # 1
Loop # 2
Loop # 3
Loop # 4

```



```
1: C= g++
2: CFLAGS= -Wall -Werror -ansi -pedantic
3: GFLAGS= -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
4: c11 = -std=c++11
5:
6: all:
7:     make NBody
8:
9: NBody: CelestialBody.o main.o
10:     $(C) CelestialBody.o main.o -o NBody $(GFLAGS) $(c11)
11:
12: CelestialBody.o: CelestialBody.cpp CelestialBody.hpp
13:     $(C) -c CelestialBody.cpp -o CelestialBody.o $(CFLAGS) $(c11)
14:
15: main.o: main.cpp CelestialBody.hpp
16:     $(C) -c main.cpp -o main.o $(CFLAGS) $(c11)
17:
18: clean:
19:     rm *.o *NBody
```

```
1: #include <iostream>
2: #include <cstdlib>
3: #include <vector>
4:
5: #include <SFML/Graphics.hpp>
6: #include <SFML/Window.hpp>
7: #include <SFML/System.hpp>
8:
9:
10: #include "CelestialBody.hpp"
11:
12: using namespace std;
13:
14: int main(int argc, char* argv[]){
15:
16:     int cbodies;
17:
18:     double dt;
19:     double radius;
20:     double T;
21:     double time;
22:
23:     string filename;
24:
25:     sf::Clock clock;
26:
27:
28:     if (argc != 3){
29:
30:         cout << "\nThere are not enough arguments, exiting!" << endl;
31:         return -1;
32:     }
33:
34:     time = 0; // start time
35:
36:     filename = argv[0];
37:     T = strtod(argv[1], NULL);
38:     dt = strtod(argv[2], NULL);
39:
40:     cin >> cbodies;
41:     cin >> radius;
42:
43:
44:     Universe cb(radius, 500, cbodies, cin);
45:
46:     sf::RenderWindow window(sf::VideoMode(600, 600), "A Small Glimps of T
he Cosmos:");
47:
48:     while (window.isOpen()){
49:
50:         sf::Event event;
51:
52:         while (window.pollEvent(event)){ if (event.type == sf::Event::Clo
sed) window.close(); }
53:
54:         window.clear();
55:
56:         if (time < T){ // as long as time hasn't run out
57:
58:             sf::Time elapsed = clock.getElapsedTime();
59:
60:             cout << "\nElapsed time: " << elapsed.asSeconds( ) << " secon
ds." << endl;
61:
62:             cb.step(dt);
```

```
63:
64:         time += dt;
65:     }
66:
67:     window.draw(cb);
68:     window.display();
69: }
70:
71:     return 0;
72: }
```

```
1: #ifndef CELESTIALBODY_HPP
2: #define CELESTIALBODY_HPP
3:
4: #include <iostream>
5: #include <string>
6: #include <vector>
7: #include <memory>
8:
9: #include <SFML/Graphics.hpp>
10:
11: using namespace std;
12:
13:
14: class CelestialBody :public sf::Drawable {
15:
16:     private:
17:
18:         double winsize;
19:
20:         double xpos;
21:         double ypos;
22:
23:         double xvel;
24:         double yvel;
25:
26:         double mass;
27:         double radius;
28:
29:         double display_x;
30:         double display_y;
31:
32:         string filename;
33:
34:         sf::Sprite sprite;
35:         sf::Texture texture;
36:
37:     public:
38:         //constructors
39:         CelestialBody();
40:         CelestialBody(double x_pos, double y_pos, double x_vel, double y_vel,
double m, string name, double radius, double winsize);
41:         ~CelestialBody();
42:
43:         friend std::istream& operator >>(std::istream& input, CelestialBody&
ci);
44:         friend std::ostream& operator <<(std::ostream& out, CelestialBody& co
);
45:
46:         virtual void draw(sf::RenderTarget& target, sf::RenderStates states)c
onst;
47:
48:         //accessor functions
49:         double get_posx();
50:         double get_posy();
51:
52:         double get_velx();
53:         double get_vely();
54:
55:         double get_mass();
56:         string get_filename();
57:
58:         //mutators
59:         void set_x_y_pos(double x_input, double y_input);
60:
61:         void set_velx(double vx);
```

```
62:     void set_vely(double vy);
63:
64:     void set_radius(double radius);
65:     void set_window(double size);
66:
67:     void set_position();
68:
69: };
70:
71: class Universe : public sf::Drawable {
72:
73:     public:
74:
75:     Universe(); // basic constructor
76:     Universe(double radius, int window, int num_of_planets, istream &in);
77:
78:     virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const;
79:
80:     friend ostream &operator <<(std::ostream& out, const Universe& co);
81:
82:     void step(double seconds);
83:
84:     double get_r();
85:     int get_numPlanets();
86:
87:     void printInfo();
88:
89:     private:
90:
91:     double r;
92:
93:     int numplanets;
94:     int winsize;
95:
96:     vector <std::unique_ptr <CelestialBody>> planets;
97:
98: };
99:
100: #endif
```

```
1: #include "CelestialBody.hpp"
2:
3: #include <iostream>
4: #include <cmath>
5:
6:
7: using namespace std;
8:
9:
10: Universe::Universe() {
11:
12:     r = 0;
13:     winsize = 0;
14: }
15:
16: Universe::Universe(double radius, int window, int num_of_planets, istream
&in) {
17:
18:     int i;
19:
20:     r = radius;
21:     winsize = window;
22:     numplanets = num_of_planets;
23:
24:     for (i = 0; i < num_of_planets; i++) {
25:
26:         unique_ptr <CelestialBody> ptr(new CelestialBody());
27:
28:         CelestialBody();
29:         planets.push_back(move(ptr));
30:         planets[i]->set_radius(r);
31:         planets[i]->set_window(window);
32:         in >> *planets[i];
33:     }
34:
35: }
36:
37: void Universe::draw(sf::RenderTarget &target, sf::RenderStates states) co
nst {
38:
39:     int i;
40:
41:     for (i = 0; i < numplanets; i++) {
42:         target.draw(*planets.at(i), states);
43:     }
44: }
45:
46:
47: void Universe::step(double seconds) {
48:
49:     int i, k;
50:
51:     double ax;
52:     double ay;
53:
54:     double dx;
55:     double dy;
56:
57:     double force;
58:     double forcex;
59:     double forcey;
60:
61:     double fx;
62:     double fy;
63:
```



```
64:  double G;
65:
66:  double velx;
67:  double vely;
68:
69:  double x2;
70:  double y2;
71:
72:  double r;
73:
74:  for (i = 0; i < numplanets; i++){
75:      fx = 0;
76:      fy = 0;
77:
78:      for (k = 0; k < numplanets; k++){
79:
80:          if (k != i){
81:
82:              G = 6.67e-11; // gravitational constant
83:
84:              dx = planets[k]->get_posx() - planets[i]->get_posx();
85:              dy = planets[k]->get_posy() - planets[i]->get_posy();
86:
87:              r = sqrt(pow(dx, 2) + pow(dy, 2));
88:
89:              force = (G * planets[k]->get_mass() * planets[i]->get_mass()) / po
w(r, 2);
90:              forcex = force * (dx / r);
91:              forcey = force * (dy / r);
92:
93:              fy += forcey;
94:              fx += forcex;
95:
96:          }
97:      }
98:
99:      ax = fx / planets[i]->get_mass();
100:      ay = fy / planets[i]->get_mass();
101:
102:      velx = planets[i]->get_velx() + seconds * ax;
103:      vely = planets[i]->get_vely() + seconds * ay;
104:
105:      planets[i]->set_velx(velx);
106:      planets[i]->set_vely(vely);
107:
108:      x2 = (planets[i]->get_posx()) + velx * seconds;
109:      y2 = (planets[i]->get_posy()) + vely * seconds;
110:
111:      planets[i]->set_x_y_pos(x2, y2);
112:  }
113: }
114:
115: void Universe::printInfo(){
116:
117:     int i; // for loop
118:
119:     cout << numplanets << endl;
120:     cout << r << endl;
121:     for (i = 0; i < numplanets; i++){
122:         cout << planets[i]->get_posx() << " " << planets[i]->get_posy() << "
"
123:         << planets[i]->get_velx() << " " << planets[i]->get_vely() << " "
124:         << planets[i]->get_mass() << " " << planets[i]->get_filename() << e
ndl;
125:     } // print out x pos, y pos, velx, vely, mass, and name
```

```
126: }
127:
128: double Universe::get_r(){ return r; }
129:
130: int Universe::get_numPlanets(){ return numplanets; }
131:
132: CelestialBody::CelestialBody(){
133:
134:     winsize = 0;
135:     xpos = 0;
136:     ypos = 0;
137:     xvel = 0;
138:     yvel = 0;
139:     mass = 0;
140:     radius = 0;
141:     filename = "";
142: }
143:
144: CelestialBody::CelestialBody(double x_pos, double y_pos, double x_vel,
145:                               double y_vel, double m, string name,
146:                               double rad, double window_size){
147:
148:     double radx;
149:     double rady;
150:
151:     xpos = x_pos; // updated values of xpos
152:     ypos = y_pos; // updated values of ypos
153:     xvel = x_vel; // updated values of xvel
154:     yvel = y_vel; // updated values of yvel
155:
156:     mass = m; // update mass
157:     radius = rad; // update radius
158:     winsize = window_size; // update window size
159:     filename = name; // update filename
160:
161:     radx = (winsize / 2) * (xpos / radius) + (winsize / 2);
162:     rady = (winsize / 2) * (ypos / radius) + (winsize / 2);
163:
164:
165:     texture.loadFromFile(filename);
166:
167:
168:     sprite.setTexture(texture);
169:     sprite.setPosition(radx, rady);
170: }
171:
172: istream &operator >>(istream &in, CelestialBody &ci){
173:
174:     double radx;
175:     double rady;
176:
177:     in >> ci.xpos >> ci.ypos >> ci.xvel >> ci.yvel >> ci.mass >> ci.filename
e;
178:
179:     radx = (ci.winsize / 2) * (ci.xpos / ci.radius) + (ci.winsize / 2);
180:     rady = (ci.winsize / 2) * (ci.ypos / ci.radius) + (ci.winsize / 2);
181:
182:     ci.texture.loadFromFile(ci.filename);
183:
184:     ci.sprite.setTexture(ci.texture);
185:     ci.sprite.setPosition(radx, rady);
186:
187:
188:     return in; // return input
189: }
```

```
190:
191: void CelestialBody::draw(sf::RenderTarget &target, sf::RenderStates state
s) const { target.draw(sprite, states); }
192:
193: CelestialBody::~CelestialBody(){}
194:
195: double CelestialBody::get_posx(){ return xpos; } // return the xpos
196: double CelestialBody::get_posy(){ return ypos; } // return the ypos
197: double CelestialBody::get_velx(){ return xvel; } // return the xvel
198: double CelestialBody::get_vely(){ return yvel; } // return the yvel
199: double CelestialBody::get_mass(){ return mass; } // return the mass
200:
201: string CelestialBody::get_filename(){ return filename; }
202:
203: void CelestialBody::set_radius(double r){ radius = r; }
204: void CelestialBody::set_window(double size){ winsize = size; }
205: void CelestialBody::set_velx(double vx){ xvel = vx; }
206: void CelestialBody::set_vely(double vy){ yvel = vy; }
207: void CelestialBody::set_x_y_pos(double x_input, double y_input){
208:
209:     double radx;
210:     double rady;
211:
212:     xpos = x_input;
213:     ypos = y_input;
214:
215:     radx = (winsize / 2) * (xpos / radius) + (winsize / 2);
216:     rady = (winsize / 2) * (-ypos / radius) + (winsize / 2);
217:
218:     sprite.setPosition(sf::Vector2f(radx, rady));
219: }
```