

Ps3: Nbody simulation

Assignment Description:

For the 4th assignment of the semester we were tasked to create an Nbody simulation. An Nbody simulation is a physics simulation that takes an objects mass and distance to other objects in order to simulate gravitational forces. We used this type of simulation to make a recreation of the solar system. This program will simulate gravitational forces for a given system over a certain length of time per second. I also received extra points for adding music and creating a new universe for my simulation.

Key Concepts and Algorithms:

This assignment used real physics formulas to calculate accurate simulations of celestial bodies. The main formula used is $F = (G * M1 * M2) / R * R$ this formula calculates the forces on the bodies and uses them to calculate the new acceleration for the bodies. For this project we used 2 classes 1 class called CelestialBodies that holds all of the relevant data members and functions to calculate forces and 1 called Universe that generates and holds the CelestialBodies.

This assignment uses smart pointers which are an object that functions as a pointer and manages the memory allocation for the user. In this assignment we used a vector that hold shared pointers to CelestialBodies. This allows the Universe to hold a near infinite number of objects. These objects are given to the program via an input text file.

What I learned in this assignment:

During this assignment I learned how to use smart pointers a concept in C++ that I had no experience in before starting this assignment. I also learned how to input data into a program from an input file with `std::cin` using the input stream. Before this assignment the only experience I had with input text files was file pointers in C.

Ps3: Screen shot



Ps3a Source Code: Makefile

```
1 CFLAGS = -Wall -Werror -std=c++11 -pedantic
2 DEPS = -lboost_unit_test_framework
3
4
5 ps4a: test.o RingBuffer.o
6     g++ test.cpp RingBuffer.cpp headers/RingBuffer.h -o ps4a $(CFLAGS) $(DEPS)
7
8 test.o:
9     g++ test.cpp RingBuffer.cpp headers/RingBuffer.h -o test.o $(CFLAGS) $(DEPS)
10
11 RingBuffer.o:
12     g++ -c RingBuffer.cpp headers/RingBuffer.h $(CFLAGS)
13
14 clean:
15     rm ps4a
16     rm *.o
17     rm headers/*.gch
18     rm *.out
```

Ps3b Source Code: Makefile

```
1 CC= g++
2 CFLAGS= -Wall -Werror -std=c++11
3 DEPS= -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
4
5 KSGuitarSim:    KSGuitarSim.o StringSound.o RingBuffer.o
6     $(CC) KSGuitarSim.o StringSound.o RingBuffer.o -o KSGuitarSim $(DEPS)
7
8 main.o: main.cpp StringSound.h
9     $(CC) -c main.cpp StringSound.h $(CFLAGS) $(DEPS)
10
11 StringSound.o: StringSound.cpp StringSound.h
12     $(CC) -c StringSound.cpp StringSound.h $(CFLAGS) $(DEPS)
13
14 RingBuffer.o: RingBuffer.cpp RingBuffer.h
15     $(CC) -c RingBuffer.cpp RingBuffer.h $(CFLAGS) $(DEPS)
16
17 clean:
18     rm *.o
19     rm *.gch
20     rm KSGuitarSim
```

Ps3a Source Code: main.cpp

```
1 #include "nBody.hpp"
2 using namespace sf;
3 using namespace std;
4
5
6 int main(int argc, char* argv[]){
7     //Open Window
8     RenderWindow window(sf::VideoMode(500, 500), "PS3a");
9     //Background
10    Texture backgroundTexture;
11    Sprite background;
12    Vector2u textureSize;
13    Vector2u windowSize;
14    if(!backgroundTexture.loadFromFile("background.jpg"))
15        return -1;
16    else{
17        textureSize = backgroundTexture.getSize();
18        windowSize = window.getSize();
19        //setting scale of background
20        float x_scale = (float) windowSize.x / textureSize.x;
21        float y_scale = (float) windowSize.y / textureSize.y;
22        background.setTexture(backgroundTexture);
23        background.setScale(x_scale, y_scale);
24    }
25    string temp_particles;
26    string temp_radius;
27    cin >> temp_particles;
28    cin >> temp_radius;
29    int particles = atoi(temp_particles.c_str());
30    double radius = atof(temp_radius.c_str());
31    unique_ptr<Universe> a(new Universe(particles, radius, window));
32
33
34    while (window.isOpen())
35    {
36        Event event;
37        while (window.pollEvent(event))
38        {
39            if (event.type == Event::Closed)
40                window.close();
41        }
42
43        window.clear();
```

```

44         window.draw(background);
45         for(int i = 0; i < particles; i++){
46             a->vectorOfBodies[i]->draw(window);
47         }
48         window.display();
49
50     }
51
52     return 0;
53 }

```

Ps3b Source Code: main.cpp

```

1  #include "nBody.hpp"
2  using namespace sf;
3  using namespace std;
4
5
6  int main(int argc, char* argv[]){
7      //args
8      double T = atof(argv[1]);
9      double deltaT = atof(argv[2]);
10
11     //Open Window
12     RenderWindow window(sf::VideoMode(500, 500), "PS3a");
13     window.setFramerateLimit(60);
14     //Music
15     Music musicFile;
16     musicFile.openFromFile("DueloftheFates.ogg");
17     musicFile.play();
18     musicFile.setLoop(true);
19     //Clock
20     Font font;
21     if (!font.loadFromFile("arial.ttf"))
22     {
23         cout << "Font can't load." << endl;
24     }
25     Text text;
26     text.setFont(font);
27     text.setCharacterSize(15);
28     text.setFillColor(Color::White);
29     text.setString("Time: " + to_string(0));
30
31     //Background

```

```

32 Texture backgroundTexture;
33 Sprite background;
34 Vector2u textureSize;
35 Vector2u windowSize;
36 if(!backgroundTexture.loadFromFile("starfield.jpg"))
37     return -1;
38 else{
39     textureSize = backgroundTexture.getSize();
40     windowSize = window.getSize();
41     //setting scale of background
42     float x_scale = (float) windowSize.x / textureSize.x;
43     float y_scale = (float) windowSize.y / textureSize.y;
44     background.setTexture(backgroundTexture);
45     background.setScale(x_scale, y_scale);
46 }
47 string temp_particles;
48 string temp_radius;
49 cin >> temp_particles;
50 cin >> temp_radius;
51 int particles = atoi(temp_particles.c_str());
52 double radius = atof(temp_radius.c_str());
53 unique_ptr<Universe> a(new Universe(particles, radius, window));
54
55
56 for(double j = 0 ; j < T; j+= deltaT){
57
58     Event event;
59     while (window.pollEvent(event))
60     {
61         if (event.type == Event::Closed)
62             window.close();
63     }
64     window.clear();
65     window.draw(background);
66     text.setString("Time: " + to_string(j));
67     window.draw(text);
68     for(int i = 0; i < particles; i++){
69         a->vectorOfBodies[i]->draw(window);
70     }
71     a->update(a->vectorOfBodies);
72     a->move(a->vectorOfBodies, deltaT);
73     window.display();
74
75
76 }
77 for(int i = 0; i < particles; i++){

```

```

78         cout << a->vectorOfBodies[i];
79     }
80     return 0;
81 }

```

Ps3a Source Code: nBody.hpp

```

1  #include<iostream>
2  #include<sstream>
3  #include<string>
4  #include<cstdlib>
5  #include<vector>
6  #include<memory>
7  #include<SFML/Graphics.hpp>
8  using namespace sf;
9  using namespace std;
10
11 const int window_length = 500;
12 const int window_width = 500;
13
14 class CelestialBody{
15     public:
16     CelestialBody();
17     CelestialBody(double xcoordinate, double ycoordinate, double xvelocity, double yvelocity, double mass, string fileName, int radius);
18     friend istream& operator>> (istream &input, CelestialBody &body);
19     friend ostream& operator<< (ostream &output, CelestialBody &body);
20     void set_radius(float Radius);
21     void draw(RenderWindow &window);
22     //void setPosition(RenderWindow &window);
23     Sprite sprite;
24     Texture texture;
25     double x_coordinate;
26     double y_coordinate;
27     double x_velocity;
28     double y_velocity;
29     double mass;
30     string fileName;
31     int radius;
32 };
33
34 class Universe : public CelestialBody{
35     public:
36     Universe(int _particles, int radius, RenderWindow& window);
37     //void draw(RenderWindow &window);

```

```

38
39     vector< shared_ptr<CelestialBody> > vectorOfBodies;
40     int particles;
41 };

```

Ps3b Source Code: nBody.hpp

```

1  #include<iostream>
2  #include<sstream>
3  #include<string>
4  #include<cstdlib>
5  #include<vector>
6  #include<memory>
7  #include<cmath>
8  #include<SFML/Graphics.hpp>
9  #include<SFML/Audio.hpp>
10 #include<SFML/Graphics/Font.hpp>
11 #include<SFML/Graphics/Text.hpp>
12 using namespace sf;
13 using namespace std;
14
15 const int window_length = 500;
16 const int window_width = 500;
17
18 class CelestialBody{
19     public:
20     CelestialBody();
21     CelestialBody(double xcoordinate, double ycoordinate, double xvelocity, double yvelocity, double mass, string fileName);
22     friend istream& operator>> (istream &input, CelestialBody &body);
23     friend ostream& operator<< (ostream &output, shared_ptr<CelestialBody> &body);
24     void set_radius(float Radius);
25     void draw(RenderWindow &window);
26     void update(vector<shared_ptr<CelestialBody> > &body);
27     void move(vector<shared_ptr<CelestialBody> > &body, double delta_T);
28     Sprite sprite;
29     Texture texture;
30     double x_coordinate;
31     double y_coordinate;
32     double x_velocity;
33     double y_velocity;
34     double x_force;
35     double y_force;
36     double mass;
37     string fileName;

```



```

38     double radius;
39     int particles;
40 };
41
42 class Universe : public CelestialBody{
43     public:
44         Universe(int _particles, int radius, RenderWindow& window);
45
46         vector< shared_ptr<CelestialBody> > vectorOfBodies;
47 };

```

Ps3a Source Code: nBody.cpp

```

1  #include "nBody.hpp"
2  using namespace sf;
3  using namespace std;
4
5
6  //Constructors
7  CelestialBody::CelestialBody(){
8      x_coordinate = 0;
9      y_coordinate = 0;
10     x_velocity = 0;
11     y_velocity = 0;
12     mass = 0;
13     radius = 0;
14     fileName = " ";
15 }
16
17 CelestialBody::CelestialBody(double xcoordinate, double ycoordinate, double xvel
18 x_coordinate = xcoordinate;
19 y_coordinate = ycoordinate;
20 x_velocity = xvelocity;
21 y_velocity = yvelocity;
22 mass = _mass;
23 radius = _radius;
24 fileName = file_Name;
25 if(!texture.loadFromFile(fileName))
26     return;
27
28 sprite.setTexture(texture);
29 sprite.setPosition(Vector2f(x_coordinate, y_coordinate));
30
31 }

```

```

32
33 Universe::Universe(int _particles, int radius, RenderWindow& window){
34     particles = _particles;
35
36     for(int i = 0; i < particles; i++){
37         shared_ptr<CelestialBody> temp(new CelestialBody());
38
39         cin >> *temp;
40
41         temp->set_radius(radius);
42
43         temp->sprite.setPosition(250 - ((temp->x_coordinate / radius) * 2 + 250) + 2
44
45         this->vectorOfBodies.push_back(temp);
46
47         cout << *temp;
48     }
49 }
50
51 }
52 //Functions
53 void CelestialBody::draw(RenderWindow &window){
54     window.draw(this->sprite);
55     return;
56 }
57
58 void CelestialBody::set_radius(float Radius){
59     radius = Radius;
60     return;
61 }
62
63 //IO streams
64 istream& operator>>(istream &input, CelestialBody &body){
65     input >> body.x_coordinate;
66     input >> body.y_coordinate;
67     input >> body.x_velocity;
68     input >> body.y_velocity;
69     input >> body.mass;
70     input >> body.fileName;
71
72     if(!body.texture.loadFromFile(body.fileName)){
73         return input;
74     }
75
76     body.sprite.setTexture(body.texture);
77     body.sprite.setPosition(Vector2f(body.x_coordinate, body.y_coordinate));

```

```

78
79     return input;
80 }
81
82 ostream& operator<< (ostream &output, CelestialBody &body){
83     output << "X_Coordinate : " << body.x_coordinate << endl << "Y_Coordinate : " << body.y_coordinate << endl;
84     output << "X_Velocity : " << body.x_velocity << endl << "Y_Velocity : " << body.y_velocity << endl;
85     output << "Mass : " << body.mass << endl << "File : " << body.fileName << endl;
86     return output;
87 }

```

Ps3b Source Code: nBody.cpp

```

1  #include "nBody.hpp"
2  using namespace sf;
3  using namespace std;
4
5
6  //Constructors
7  CelestialBody::CelestialBody(){
8      x_coordinate = 0;
9      y_coordinate = 0;
10     x_velocity = 0;
11     y_velocity = 0;
12     x_force = 0;
13     y_force = 0;
14     mass = 0;
15     radius = 0;
16     fileName = " ";
17 }
18
19 CelestialBody::CelestialBody(double xcoordinate, double ycoordinate, double xvelocity, double yvelocity, double mass, double radius, string fileName){
20     x_coordinate = xcoordinate;
21     y_coordinate = ycoordinate;
22     x_velocity = xvelocity;
23     y_velocity = yvelocity;
24     mass = mass;
25     radius = radius;
26     fileName = fileName;
27     if(!texture.loadFromFile(fileName))
28         return;
29
30     sprite.setTexture(texture);
31     sprite.setPosition(Vector2f(x_coordinate, y_coordinate));

```

```

32
33 return;
34 }
35
36 Universe::Universe(int _particles, int radius, RenderWindow& window){
37     particles = _particles;
38
39     for(int i = 0; i < particles; i++){
40         shared_ptr<CelestialBody> temp(new CelestialBody());
41
42         cin >> *temp;
43
44         temp->set_radius(radius);
45
46         temp->sprite.setPosition(250 - ((temp->x_coordinate / radius) * 2 + 250) + 2
47
48         this->vectorOfBodies.push_back(temp);
49
50     }
51     return;
52 }
53
54 //Functions
55 void CelestialBody::draw(RenderWindow &window){
56     window.draw(this->sprite);
57     return;
58 }
59
60 void CelestialBody::set_radius(float Radius){
61     radius = Radius;
62     return;
63 }
64
65 void CelestialBody::update(vector<shared_ptr<CelestialBody> > &body){
66     for(int i = 0; i < particles; i++){
67         for(int j = 0; j < particles; j++){
68             if(i != j){
69                 double change_in_x = body[j]->x_coordinate - body[i]->x_coordinate;
70                 double change_in_y = body[j]->y_coordinate - body[i]->y_coordinate;
71                 double r = sqrt((change_in_x * change_in_x) + (change_in_y * change_in_y));
72                 double beeg_G = 6.67e-11;
73                 double T_force = ((beeg_G*body[i]->mass*body[j]->mass) / (r * r));
74                 body[i]->x_force = body[i]->x_force + (T_force * (change_in_x / r));
75                 body[i]->y_force = body[i]->y_force + (T_force * (change_in_y / r));
76
77             }

```

```

78         }
79     }
80     return;
81 }
82
83 void CelestialBody::move(vector<shared_ptr<CelestialBody> > &body, double delta_
84     for(int i = 0; i < particles; i++){
85         double x_acceleration = body[i]->x_force/body[i]->mass;
86         double y_acceleration = body[i]->y_force/body[i]->mass;
87         body[i]->x_velocity = body[i]->x_velocity + (x_acceleration* delta_T);
88         body[i]->y_velocity = body[i]->y_velocity + (y_acceleration* delta_T);
89         body[i]->x_coordinate = body[i]->x_coordinate + (body[i]->x_velocity* de
90         body[i]->y_coordinate = body[i]->y_coordinate + (body[i]->y_velocity* de
91         body[i]->sprite.setPosition(250 - ((body[i]->x_coordinate / body[i]->rad
92         body[i]->x_force = body[i]->y_force = 0;
93     }
94     return;
95 }
96
97 //IO streams
98 istream& operator>>(istream &input, CelestialBody &body){
99     input >> body.x_coordinate;
100    input >> body.y_coordinate;
101    input >> body.x_velocity;
102    input >> body.y_velocity;
103    input >> body.mass;
104    input >> body.fileName;
105
106    if(!body.texture.loadFromFile(body.fileName)){
107        return input;
108    }
109
110    body.sprite.setTexture(body.texture);
111    body.sprite.setPosition(Vector2f(body.x_coordinate, body.y_coordinate));
112    body.sprite.setOrigin(10,10);
113
114    return input;
115 }
116
117 ostream& operator<< (ostream &output, shared_ptr<CelestialBody> &body){
118     output << body->x_coordinate << "      " << body->y_coordinate << "
";
119     output << body->x_velocity << "      " << body->y_velocity << "
";
120     output << body->mass << "      " << body->fileName << endl;
121     return output;

```

