PS2 Static and Dynamic N-Body Simulations

What is the assignment?

For part A, the assignment is basically setting up the solar system and positioning the planets to their positions that is also relative to the screen size so that their positions would be exactly where they should be even with different window screen sizes

For part B, the assignment is a continuation of the N-Body Simulation assignment of part A where we basically calculate the positions of the planets and scale it relative to the window size. Then we print or draw the planets to their respective positions within the universe. But in part B we basically simulate the universe in real life where the planets move around the sun and each object is affecting each individual planet.

What I learned in this assignment:

1) I learned how to utilize smart pointers for the first time and how they are handy for this assignment and potential upcoming projects.
2) I also learned how to use a for each loop for the first time as well.
3) Additionally, I learn how to scale the planets to their render window positions based on their normal or original positions.
4) I learned how to use SFML text and fonts and utilize them for this project.
5) I learned that debugging without a debugger is quite a tedious task.
6) I also learned how to create my own universe.

What I accomplished in this assignment:

1) I managed to draw the planets relative to the window screen size and override the draw function successfully.
2) I also successfully used a for each loop for the first time.
3) Additionally, I added a background image for extra credit as well.

4) I managed to get the planets rotate around the sun counterclockwise without it flying off the screen.
5) Additionally, I managed to display the elapsed time of the universe in the window using SFML text and font.
6) Furthermore, I successfully created my own universe called chaos_center.txt

Any Issues I had with this assignment:

For part A, I had issues logging into wsl. Managed to fix it by going to PowerShell and doing --wsl.exe shutdown I believe. Also, I could not figure out why the planets were not being drawn on the window and how to position them relative to the window screen size. I also had trouble with how to create vector of pointers of CelestialBody objects.

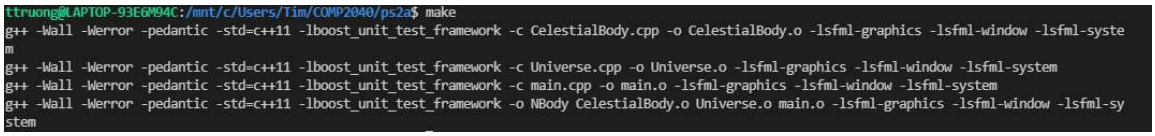
For part B, I could not figure out the math and physics behind it as my planets disappeared off the screen. Turns out I needed to account the original positions the planets would be in outer space instead of their positions relative to the screen before applying the forces that each and every planet affects one another. Because when going through my updateSetPos() function the planet's original/normal positions in outer space would be updated to fit/match in the new window screen size. But by using my updateSetPos() again, the planet's current positions were not using their normal positions, so the planet's updated positions would not be relative to the window screen and their updated positions would be very whack and they fly out of the screen. Because when updating the planet's positions, the planet's normal or original positions should be utilized instead of the positions relative to the screen size. That was my main problem because I added so many printf statements to calculate if my math was right (which it was), and I had to narrow down every possibility on why code is broken and asked tutors and hit my physics textbook to help cope with my sanity on working on this assignment. I also had to negate the force because my numbers were off.

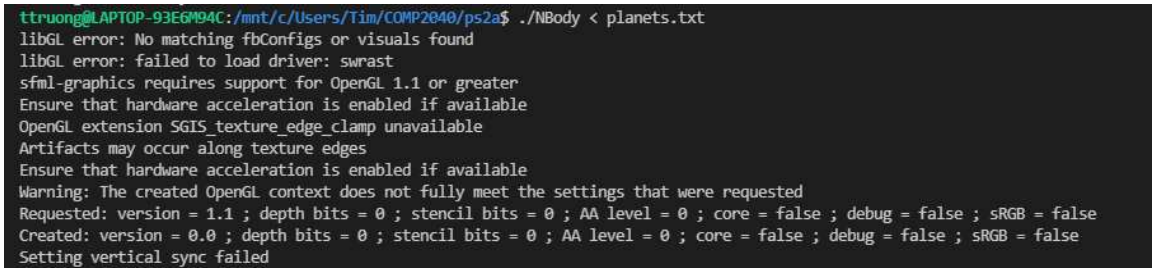
Key Algorithms, Data Structures, or OO Designs used for this assignment:

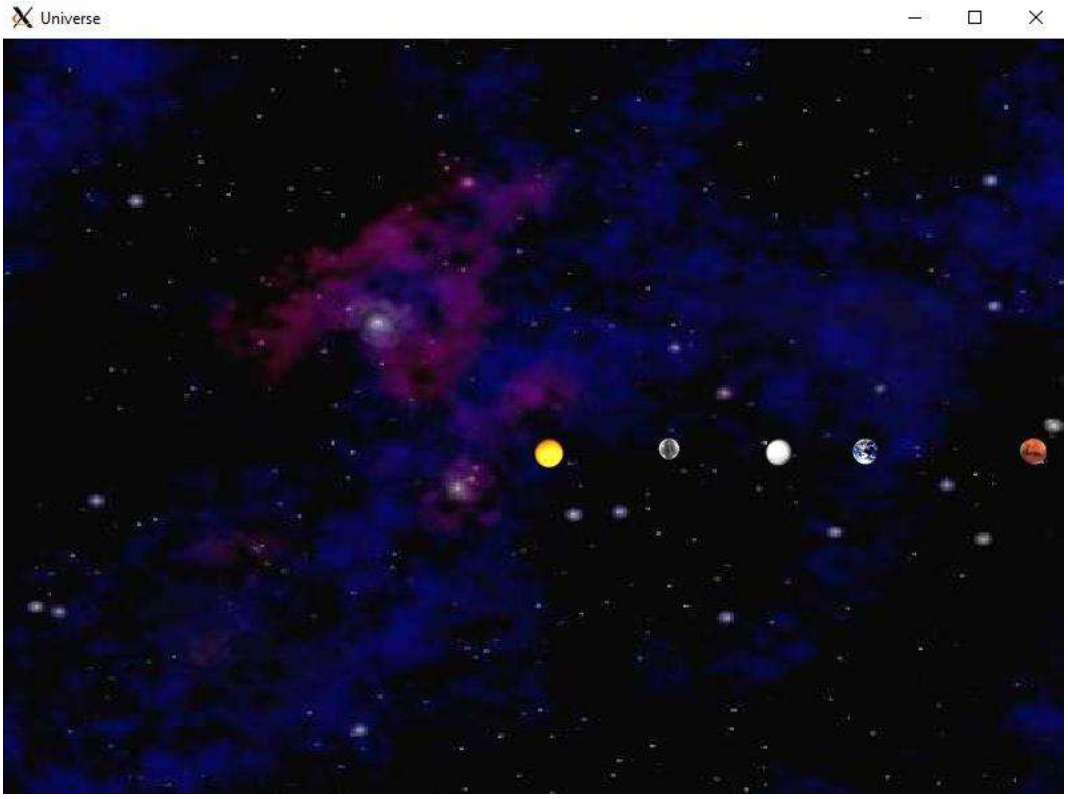
For part A, I utilized inheritance and getters and setters to help me progress in this assignment. By inheriting the sf::Drawable, I can override the draw function so that it draws out CelestialBody objects inside a vector of pointers to CelestialBody. And by utilizing getter and setter member functions to update x_position and y_position of the planets relative to the window size or universe size rather.

For part B, in order to successfully complete this part of the assignment, I used a lot of getter and setter member functions to obtain the planet's normal positions outside of the universe. After that I used their normal positions to calculate their next positions, velocities, acceleration, etc. Then I called updateSetPos() in order to set the planet's positions relative to the window screen size. Then finally I continually draw the planets on the window screen to simulate the planets rotating around the sun. This was achievable by using inheritance, getters and setters, and smart pointers which I used in this assignment.

Screenshot(s):

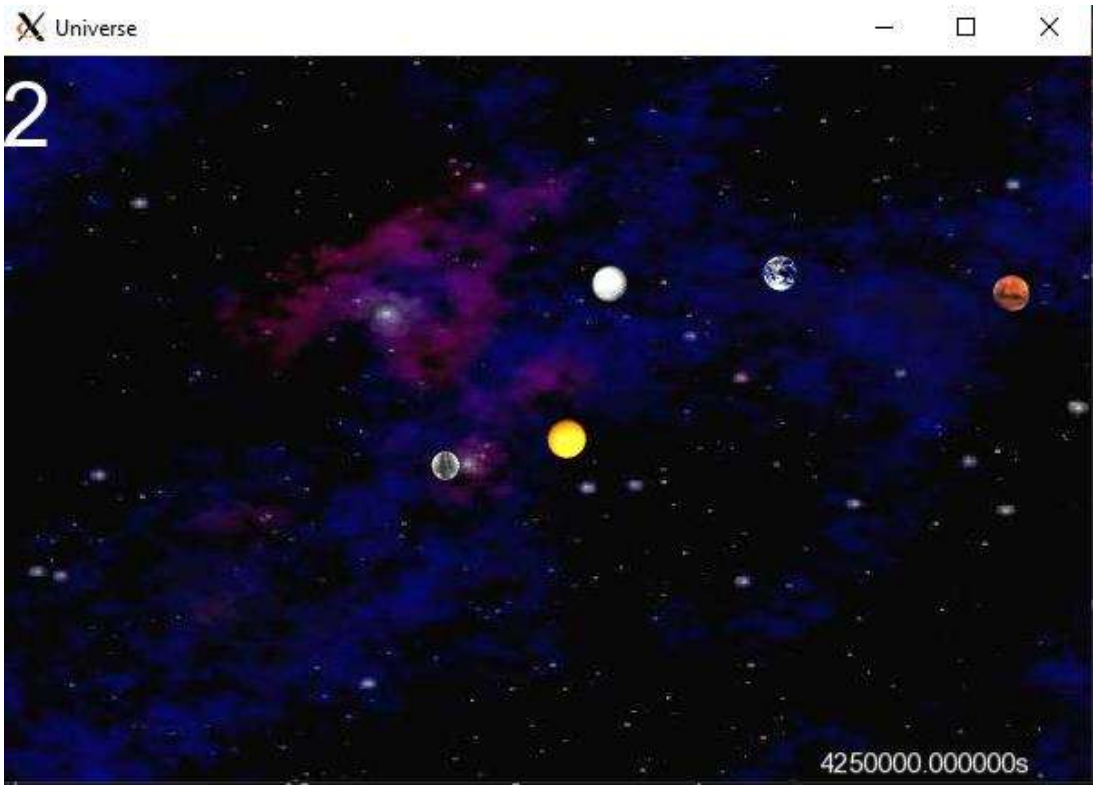


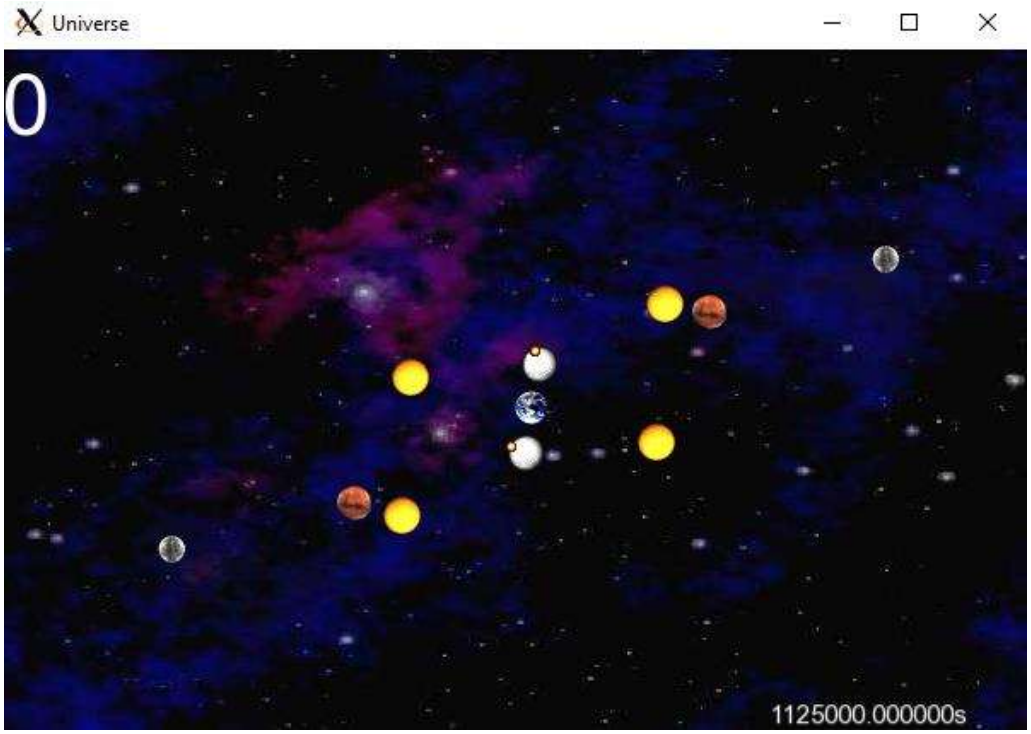





```
ttruong@LAPTOP-93E6M94C:/mnt/c/Users/Tim/COMP2040/ps2b$ make
g++ -Wall -Werror -pedantic -std=c++11 -lboost_unit_test_framework -c CelestialBody.cpp -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
g++ -Wall -Werror -pedantic -std=c++11 -lboost_unit_test_framework -c Universe.cpp -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
g++ -Wall -Werror -pedantic -std=c++11 -lboost_unit_test_framework -c main.cpp -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
g++ -Wall -Werror -pedantic -std=c++11 -lboost_unit_test_framework -o NBody CelestialBody.o Universe.o main.o -lsfml-graphics -lsfml-window -lsfml-sy
stem -lsfml-audio
```

```
ttruong@LAPTOP-93E6M94C:/mnt/c/Users/Tim/COMP2040/ps2b$ ./NBody 157788000.0 25000.0 < planets.txt
libGL error: No matching fbConfigs or visuals found
libGL error: failed to load driver: swrast
sfml-graphics requires support for OpenGL 1.1 or greater
Ensure that hardware acceleration is enabled if available
OpenGL extension SGIS_texture_edge_clamp unavailable
Artifacts may occur along texture edges
Ensure that hardware acceleration is enabled if available
Warning: The created OpenGL context does not fully meet the settings that were requested
Requested: version = 1.1 ; depth bits = 0 ; stencil bits = 0 ; AA level = 0 ; core = false ; debug = false ; sRGB = false
Created: version = 0.0 ; depth bits = 0 ; stencil bits = 0 ; AA level = 0 ; core = false ; debug = false ; sRGB = false
Setting vertical sync failed
```


Universe
2
4250000.000000s


Universe
0
1125000.000000s

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

```
 1: // Copyright [2022] <Tim Truong>
 2: #include "CelestialBody.h"
 3: #include "Universe.h"
 4: #include <iomanip>
 5:
 6: /*************************************************************
 7:  *Name: Tim Truong
 8:  *Course name: COMP.2040
 9:  *Assignment: PS2b - Dynamic N-Body Simulation
10:  *Instructor's name: Dr. James Daly
11:  *Date: 2/22/22
12:  *Sources Of Help: Tutor, StackOverflow, SFML Tutorial Website,
13:  My Physics Notes
14: *************************************************************/
15: /*************************************************************
16:  *Note: PSXa Redo Assignment for PS2b Dynamic N-Body Simulation
17: *************************************************************/
18: const double VIEWPORT_WIDTH = 600;
19: const double VIEWPORT_HEIGHT = 400;
20: const double UNIV_TIME_WIDTH_FACTOR = 1.333;
21: const double UNIV_TIME_HEIGHT_FACTOR = 1.052;
22:
23: int main(int argc, char* argv[]) {
24:     double changeInT, T, current_time = 0;
25:
26:     if (argc < 3 || argc > 3) {
27:         exit(1);
28:     } else {
29:         T = atof(argv[1]);
30:         changeInT = atof(argv[2]);
31:     }
32:
33:     sf::Text text, secs;
34:     sf::Font MyFont;
35:
36:     if (!MyFont.loadFromFile("arial.TTF")) {
37:         return -1;
38:     }
39:
40:     Universe u;
41:     sf::Image background;
42:
43:     if (!background.loadFromFile("starfield.jpg")) {
44:         return -1;
45:     }
46:
47:     std::cin >> u;
48:
49:     sf::Sprite sprite;
50:     sf::Texture texture;
51:     sf::Vector2f size;
52:
53:     texture.loadFromImage(background);
54:     sprite.setTexture(texture);
55:     size.x = background.getSize().x;
56:     size.y = background.getSize().y;
57:
58:     sf::RenderWindow window(sf::VideoMode(VIEWPORT_WIDTH,
59:     VIEWPORT_HEIGHT), "Universe");
60:     float ScaleX, ScaleY;
61:
62:     if (size.x != VIEWPORT_WIDTH || size.y != VIEWPORT_HEIGHT) {
63:         ScaleX = static_cast<float>(VIEWPORT_WIDTH / size.x);
64:         ScaleY = static_cast<float>(VIEWPORT_HEIGHT / size.y);
65:         sprite.setScale(ScaleX, ScaleY);
```

```
    }

    size.x = window.getSize().x;
    size.y = window.getSize().y;

    u.set_windowSize(size);

    for (int i = 0; i < u.get_num_of_planets(); i++) {
        std::shared_ptr<CelestialBody> bodies(new CelestialBody);
        std::cin >> *bodies;
        // std::cout << *bodies;
        bodies->setNormXPos(bodies->get_x_pos());
        bodies->setNormYPos(bodies->get_y_pos());
        bodies->setRadius(u.get_radius());
        bodies->setWindowSize(u.get_windowSize());
        u.initVector(bodies);
    }

    window.setFramerateLimit(60);

    int i = 0;
    text.setFont(MyFont);
    text.setString(std::to_string(i));
    text.setCharacterSize(50);

    while (window.isOpen()) {
        sf::Event event;
        while (window.pollEvent(event)) {
            if (event.type == sf::Event::Closed) {
                window.close();
            }
            if (sf::Keyboard::isKeyPressed(sf::Keyboard::Escape)) {
                window.close();
            }
            if (sf::Keyboard::isKeyPressed(sf::Keyboard::Backspace)) {
                window.close();
            }
        }

        window.clear(sf::Color::Black);

        window.draw(sprite);

        for (int i = 0; i < u.get_num_of_planets(); i++) {
            u.planets[i]->updateSetPos(u.planets[i]->getRadius(),
            u.planets[i]->getWindowSize());
            // std::cout << *u.planets[i];
        }

        window.draw(u);

        if (current_time < T) {
            secs.setFont(MyFont);
            secs.setString(std::to_string(current_time) + 's');
            secs.setCharacterSize(15);
            secs.setPosition(VIEWPORT_WIDTH / UNIV_TIME_WIDTH_FACTOR,
            VIEWPORT_HEIGHT / UNIV_TIME_HEIGHT_FACTOR);

            i++;
            u.ApplyForce();
            u.step(changeInT);

            window.draw(text);
            window.draw(secs);

```

```
131:             current_time += changeInT;
132:
133:             text.setString(std::to_string(i / 60));
134:         } else {
135:             std::cout.precision(4);
136:             std::cout << std::scientific;
137:
138:             for (int i = 0; i < u.get_num_of_planets(); i++) {
139:                 std::cout << *u.planets[i];
140:             }
141:             window.clear();
142:
143:             window.close();
144:         }
145:         window.display();
146:     }
147:
148:     return 0;
149: }
```

```
 1: // Copyright [2022] <Tim Truong>
 2: #pragma once
 3: #ifndef COMP2040_PS2B_CELESTIALBODY_H_
 4: #define COMP2040_PS2B_CELESTIALBODY_H_
 5:
 6: #include <iostream>
 7: #include <memory>
 8: #include <cmath>
 9: #include <string>
10: #include <SFML/System.hpp>
11: #include <SFML/Window.hpp>
12: #include <SFML/Graphics.hpp>
13: #include <SFML/Audio.hpp>
14:
15: class CelestialBody : public sf::Drawable {
16:  public:
17:         CelestialBody();
18:         CelestialBody(double x, double y, double xVel, double yVel,
19:         double Mass, std::string pic,
20:         double radius, sf::Vector2f windowSize);
21:         friend std::istream& operator>>(std::istream& in,
22:         CelestialBody& body);
23:         friend std::ostream& operator<<(std::ostream& out,
24:         CelestialBody& body);
25:         void updateSetPos(double radius, sf::Vector2f windowSize);
26:         double get_x_pos(void) const;
27:         double get_y_pos(void) const;
28:         double getNormXPos(void) const;
29:         double getNormYPos(void) const;
30:         double get_x_vel(void) const;
31:         double get_y_vel(void) const;
32:         double get_mass(void) const;
33:         double get_x_acc(void) const;
34:         double get_y_acc(void) const;
35:         double get_x_force(void) const;
36:         double get_y_force(void) const;
37:         double getRadius(void) const;
38:         sf::Vector2f getWindowSize(void) const;
39:         std::string get_fileName(void) const;
40:         void set_x_pos(double xpos);
41:         void set_y_pos(double ypos);
42:         void setNormXPos(double OG_XPos);
43:         void setNormYPos(double OG_YPos);
44:         void setPos(double xpos, double ypos);
45:         void updatePos(double seconds);
46:         void set_x_vel(double xvel);
47:         void set_y_vel(double yvel);
48:         void updateVel(double seconds);
49:         void setVel(double xvel, double yvel);
50:         void set_mass(double planet_mass);
51:         void set_x_acc(double xAcc);
52:         void set_y_acc(double yAcc);
53:         void updateAccel(void);
54:         void set_x_force(double xForce);
55:         void set_y_force(double yForce);
56:         void setRadius(double R);
57:         void setWindowSize(sf::Vector2f ViewPortSize);
58:         void set_fileName(std::string fileName);
59:
60:  private:
61:         virtual void draw(sf::RenderTarget& target, // NOLINT
62:         sf::RenderStates states) const;
63:         double x_pos, y_pos, x_vel, y_vel, x_acc, y_acc, x_force,
64:         y_force, mass;
65:         double normPosX, normPosY;
```

```
66:         double radius;
67:         sf::Vector2f window_size;
68:         std::string planet_gif;
69:         sf::Image image;
70:         sf::Sprite sprite;
71:         sf::Texture texture;
72: };
73:
74: #endif  // COMP2040_PS2B_CELESTIALBODY_H_
```

```
 1: // Copyright [2022] <Tim Truong>
 2: #include "CelestialBody.h"
 3: #include <string>
 4:
 5: CelestialBody::CelestialBody() {
 6: }
 7:
 8: CelestialBody::CelestialBody(double x, double y, double xVel,
 9: double yVel, double Mass, std::string pic,
10: double radius, sf::Vector2f windowSize) {
11:     if (!image.loadFromFile(pic)) {
12:         return;
13:     }
14:
15:     texture.loadFromImage(image);
16:     sprite.setTexture(texture);
17:
18:     x_pos = x;
19:     y_pos = y;
20:     x_vel = xVel;
21:     y_vel = yVel;
22:     mass = Mass;
23:
24:     sprite.setPosition(x_pos, y_pos);
25:     updateSetPos(radius, windowSize);
26: }
27:
28: std::istream& operator>>(std::istream& in, CelestialBody& body) {
29:     in >> body.x_pos >> body.y_pos >> body.x_vel >> body.y_vel >>
30:     body.mass >> body.planet_gif;
31:
32:     body.image.loadFromFile(body.planet_gif);
33:     body.texture.loadFromImage(body.image);
34:     body.sprite.setTexture(body.texture);
35:     body.sprite.setPosition(body.get_x_pos(), body.get_y_pos());
36:
37:     return in;
38: }
39:
40: std::ostream& operator<<(std::ostream& out, CelestialBody& body) {
41:     out << body.normPosX << " " << body.normPosY << " " << body.x_vel
42:     << " " << body.y_vel << " " << body.mass << " "
43:     << body.planet_gif << std::endl;
44:
45:     return out;
46: }
47:
48: void CelestialBody::draw(sf::RenderTarget& target,
49: sf::RenderStates states)
50: const {
51:     target.draw(sprite, states);
52:
53:     return;
54: }
55:
56: void CelestialBody::updateSetPos(double radius,
57: sf::Vector2f windowSize) {
58:       // std::cout << "POS: " << get_x_pos() << " " << get_y_pos()
59:       // << std::endl;
60:       set_x_pos((radius + getNormXPos()) * (windowSize.x /
61:       (radius * 2)));
62:       set_y_pos((radius - getNormYPos()) * (windowSize.y /
63:       (radius * 2)));
64:       // std::cout << "POS: " << get_x_pos() << " " << get_y_pos()
65:       // << std::endl;
```

```
   66:       setPos(get_x_pos(), get_y_pos());
   67: }
   68:
   69: double CelestialBody::get_x_pos(void) const {
   70:     return x_pos;
   71: }
   72:
   73: double CelestialBody::get_y_pos(void) const {
   74:     return y_pos;
   75: }
   76:
   77: double CelestialBody::getNormXPos(void) const {
   78:     return normPosX;
   79: }
   80:
   81: double CelestialBody::getNormYPos(void) const {
   82:     return normPosY;
   83: }
   84:
   85: double CelestialBody::get_x_vel(void) const {
   86:     return x_vel;
   87: }
   88: double CelestialBody::get_y_vel(void) const {
   89:     return y_vel;
   90: }
   91: double CelestialBody::get_mass(void) const {
   92:     return mass;
   93: }
   94:
   95: double CelestialBody::get_x_acc(void) const {
   96:     return x_acc;
   97: }
   98:
   99: double CelestialBody::get_y_acc(void) const {
  100:     return y_acc;
  101: }
  102:
  103: double CelestialBody::get_x_force(void) const {
  104:     return x_force;
  105: }
  106:
  107: double CelestialBody::get_y_force(void) const {
  108:     return y_force;
  109: }
  110:
  111: double CelestialBody::getRadius(void) const {
  112:     return radius;
  113: }
  114:
  115: sf::Vector2f CelestialBody::getWindowSize(void) const {
  116:     return window_size;
  117: }
  118:
  119: std::string CelestialBody::get_fileName(void) const {
  120:     return planet_gif;
  121: }
  122:
  123: void CelestialBody::set_x_pos(double xpos) {
  124:     x_pos = xpos;
  125: }
  126:
  127: void CelestialBody::set_y_pos(double ypos) {
  128:     y_pos = ypos;
  129: }
  130:
```

```
131: void CelestialBody::setNormXPos(double OG_XPos) {
132:     normPosX = OG_XPos;
133: }
134:
135: void CelestialBody::setNormYPos(double OG_YPos) {
136:     normPosY = OG_YPos;
137: }
138:
139: void CelestialBody::setPos(double xpos, double ypos) {
140:     x_pos = xpos;
141:     y_pos = ypos;
142:
143:     sprite.setPosition(x_pos, y_pos);
144: }
145:
146: void CelestialBody::updatePos(double deltaT) {
147:     // std::cout << "POS: " << x_pos << " " << y_pos << std::endl;
148:     // std::cout << "NPOS: " << normPosX << " " << normPosY << std::endl;
149:     // std::cout << "VELO: " << x_vel << " " << y_vel << std::endl;
150:     normPosX = normPosX + (deltaT * x_vel);
151:     normPosY = normPosY + (deltaT * y_vel);
152:     // std::cout << "POS: " << x_pos << " " << y_pos << std::endl;
153:     // std::cout << "NPOS: " << normPosX << " " << normPosY << std::endl;
154: }
155:
156: void CelestialBody::set_x_vel(double xvel) {
157:     x_vel = xvel;
158: }
159:
160: void CelestialBody::set_y_vel(double yvel) {
161:     y_vel = yvel;
162: }
163:
164: void CelestialBody::setVel(double xvel, double yvel) {
165:     x_vel = xvel;
166:     y_vel = yvel;
167: }
168:
169: void CelestialBody::updateVel(double deltaT) {
170:     // std::cout << "VELO: " << x_vel << " " << y_vel << std::endl;
171:     x_vel = x_vel + (deltaT * x_acc);
172:     // x_vel *= -1;
173:     y_vel = y_vel + (deltaT * y_acc);
174:     // y_vel *= 1;
175:     // std::cout << "VELO: " << x_vel << " " << y_vel << std::endl;
176: }
177:
178: void CelestialBody::set_mass(double planet_mass) {
179:     mass = planet_mass;
180: }
181:
182: void CelestialBody::set_x_acc(double xAcc) {
183:     x_acc = xAcc;
184: }
185:
186: void CelestialBody::set_y_acc(double yAcc) {
187:     y_acc = yAcc;
188: }
189:
190: void CelestialBody::updateAccel(void) {
191:     // std::cout << "FNET: " << get_x_force() << " " << get_y_force()
192:     // << std::endl;
193:     // std::cout << get_mass() << std::endl;
194:     x_acc = x_force / mass;
195:     // x_acc *= -1;
```

```
196:     y_acc = y_force / mass;
197:     // y_acc *= -1;
198: }
199:
200: void CelestialBody::set_x_force(double xForce) {
201:     x_force = xForce;
202: }
203:
204: void CelestialBody::set_y_force(double yForce) {
205:     y_force = yForce;
206: }
207:
208: void CelestialBody::setRadius(double R) {
209:     radius = R;
210: }
211:
212: void CelestialBody::setWindowSize(sf::Vector2f ViewPortSize) {
213:     window_size = ViewPortSize;
214: }
215:
216: void CelestialBody::set_fileName(std::string fileName) {
217:     planet_gif = fileName;
218: }
```

```
 1: // Copyright [2022] <Tim Truong>
 2: #pragma once
 3: #ifndef COMP2040_PS2B_UNIVERSE_H_
 4: #define COMP2040_PS2B_UNIVERSE_H_
 5:
 6: #include "CelestialBody.h"
 7: #include <vector>
 8: #include <memory>
 9:
10: class Universe : public sf::Drawable {
11:  public:
12:         friend std::istream& operator>>(std::istream& in, Universe& u);
13:         void initVector(std::shared_ptr<CelestialBody> bodies);
14:         sf::Vector2<double> pairWiseForce
15:         (std::shared_ptr<CelestialBody> body1,
16:         std::shared_ptr<CelestialBody> body2);
17:         sf::Vector2<double> ApplyForce(void);
18:         int get_num_of_planets(void) const;
19:         double get_radius(void) const;
20:         sf::Vector2f get_windowSize(void) const;
21:         void set_num_of_planets(int numPlanets);
22:         void set_radius(double radius);
23:         void set_windowSize(sf::Vector2f size);
24:         void step(double seconds);
25:         std::vector<std::shared_ptr<CelestialBody>> planets;
26:  private:
27:         virtual void draw(sf::RenderTarget& target, // NOLINT
28:         sf::RenderStates states) const;
29:         const double GRAVITATIONAL_CONST = 6.67E-11;
30:         int num_of_planets;
31:         double univ_radius;
32:         sf::Vector2f windowSize;
33: };
34:
35: #endif  // COMP2040_PS2B_UNIVERSE_H_
```

```
 1: // Copyright [2022] <Tim Truong>
 2: #include "Universe.h"
 3: #include "CelestialBody.h"
 4: #include <memory>
 5:
 6: std::istream & operator>>(std::istream& in, Universe& u) {
 7:     in >> u.num_of_planets >> u.univ_radius;
 8:
 9:     return in;
10: }
11:
12: void Universe::initVector(std::shared_ptr<CelestialBody> bodies) {
13:     planets.push_back(bodies);
14:
15:     return;
16: }
17:
18: void Universe::draw(sf::RenderTarget& target,
19: sf::RenderStates states) const {
20:     for (auto body : planets) {
21:         target.draw(*body);
22:     }
23:
24:     return;
25: }
26:
27: sf::Vector2<double> Universe::pairWiseForce
28: (std::shared_ptr<CelestialBody> body1,
29: std::shared_ptr<CelestialBody> body2) {
30:     sf::Vector2<double> planet_force;
31:     // std::cout << "X_POS: " << body1->getNormXPos() << " "
32:     // << body2->getNormXPos() << std::endl;
33:     // std::cout << "Y_POS: " << body1->getNormYPos() << " "
34:     // << body2->getNormYPos() << std::endl;
35:     double deltaX = body1->getNormXPos() - body2->getNormXPos();
36:     double deltaY = body1->getNormYPos() - body2->getNormYPos();
37:     // std::cout << "deltaX: " << x << std::endl;
38:     // std::cout << "deltaY: " << y << std::endl;
39:
40:     double r2 = (deltaX * deltaX) + (deltaY * deltaY);
41:     double r = sqrt(r2);
42:     // std::cout << "r2: " << r2 << std::endl;
43:     // std::cout << "r: " << r << std::endl;
44:     // std::cout << GRAVITATIONAL_CONST << std::endl;
45:     // std::cout << body1->get_mass() << " "
46:     // << body2->get_mass() << std::endl;
47:     double Force = (((GRAVITATIONAL_CONST) * body1->get_mass()
48:     * body2->get_mass()) / (r2));
49:     // std::cout << "Force: " << Force << std::endl;
50:     Force *= -1;
51:     planet_force.x = Force * (deltaX / r);
52:     planet_force.y = Force * (deltaY / r);
53:     // std::cout << "FORCEX: " << planet_force.x << std::endl;
54:     // std::cout << "FORCEY: " << planet_force.y << std::endl;
55:
56:     return planet_force;
57: }
58:
59: void Universe::step(double seconds) {
60:     for (int i = 0; i < get_num_of_planets(); i++) {
61:         planets[i]->updateAccel();
62:         planets[i]->updateVel(seconds);
63:         planets[i]->updatePos(seconds);
64:         planets[i]->updateSetPos(univ_radius, windowSize);
65:     }
```

```
    66:
    67:     return;
    68: }
    69:
    70: sf::Vector2<double> Universe::ApplyForce(void) {
    71:     double netForceX = 0, netForceY = 0;
    72:     sf::Vector2<double> planet_force;
    73:     planet_force.x = 0;
    74:     planet_force.y = 0;
    75:
    76:     for (int i = 0; i < get_num_of_planets(); i++) {
    77:         for (int j = 0; j < get_num_of_planets(); j++) {
    78:             if (i != j) {
    79:                 planet_force = pairWiseForce(planets[i], planets[j]);
    80:                 // std::cout << planet_force.x << " "
    81:                 // << planet_force.y << std::endl;
    82:                 // std::cout << netForceX << " "
    83:                 // << netForceY << std::endl;
    84:                 netForceX = netForceX + planet_force.x;
    85:                 netForceY = netForceY + planet_force.y;
    86:                 // std::cout << netForceX << " " << netForceY
    87:                 // << std::endl;
    88:             }
    89:             // std::cout << "FX: " << netForceX << " "
    90:             // << netForceY << std::endl;
    91:         }
    92:         // std::cout << netForceX << " " << netForceY << std::endl;
    93:         // std::cout << planet_force.x << " " << planet_force.y <<
    94:         // std::endl;
    95:         planets[i]->set_x_force(netForceX);
    96:         planets[i]->set_y_force(netForceY);
    97:         netForceX = 0;
    98:         netForceY = 0;
    99:         planet_force.x = 0;
   100:         planet_force.y = 0;
   101:     }
   102:
   103:     return planet_force;
   104: }
   105:
   106: int Universe::get_num_of_planets(void) const {
   107:     return num_of_planets;
   108: }
   109:
   110: double Universe::get_radius(void) const {
   111:     return univ_radius;
   112: }
   113:
   114: sf::Vector2f Universe::get_windowSize(void) const {
   115:     return windowSize;
   116: }
   117:
   118: void Universe::set_num_of_planets(int numPlanets) {
   119:     num_of_planets = numPlanets;
   120: }
   121:
   122: void Universe::set_radius(double radius) {
   123:     univ_radius = radius;
   124: }
   125:
   126: void Universe::set_windowSize(sf::Vector2f size) {
   127:      windowSize = size;
   128: }
```