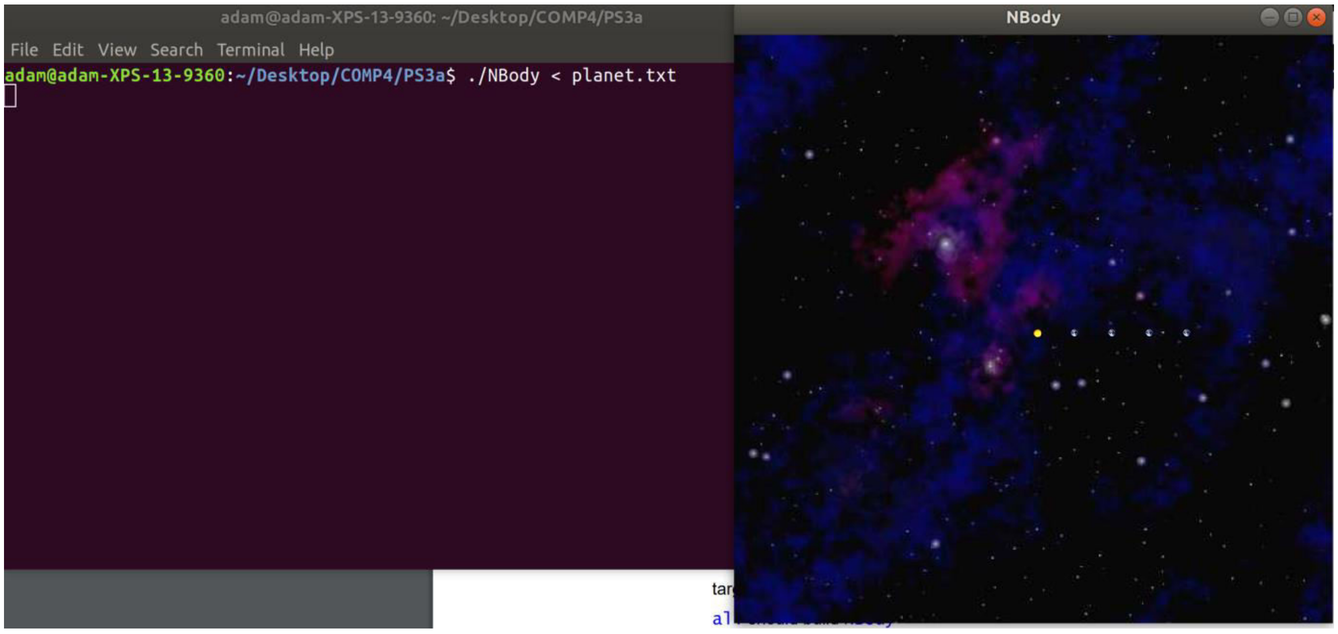
PS3 N-Body Simulation

In this project, I used physics and the SFML library to simulate celestial bodies and gravity graphically. One key OO design that was central to this assignment was the use of classes to create different bodies. In addition to that, I also used unique pointers to be able to draw multiple different sprites for each different celestial body in the simulation. To print the state of the universe at the end of the simulation I just outputted the current x and y positions, and the x and y velocities at the time the window is closed. To make the planets rotate counter clockwise, I just -= for all acceleration and velocities, because when it was +=, it was going clockwise. Used shared pointers from before. To play the song, I had to change the makefile to include -lsfml-audio in the LIBS, otherwise it didnt work. To display text I had to download a .ttf file to get text, and I also had to look online on how to do it. I also added music and a timer to the program as well.

One difficulty was that it took me forever to figure out how to make the planets move reasonably, and I was stuck the majority of the time with the planets just going up, or just going in one direction, or disappearing.


adam@adam-XPS-13-9360: ~/Desktop/COMP4/PS3a
File Edit View Search Terminal Help
NBody

```
adam@adam-XPS-13-9360:~/Desktop/COMP4/PS3a$ ./NBody < planet.txt
```

```
CC = g++
CFLAGS = -std=c++11 -c -g -Wall -Werror -pedantic
LIBS = -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio

all: NBody

NBody: main.o planets.o
	$(CC) main.o -o NBody $(LIBS)

main.o: main.cpp
	$(CC) -c $(CFLAGS) main.cpp

planets.o: planets.cpp planets.hpp
	$(CC) -c $(CFLAGS) planets.cpp planets.hpp

clean:
	rm *.o NBody *.gch
```

```cpp
#include <SFML/Audio.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>

#include <iostream>
#include <vector>
#include <string>
#include <memory>
#include <math.h>

#define G 6.67e-11

using namespace std;
using namespace sf;

class Body : public Drawable {
private:
    Texture texture;
    Vector2u winSize;

public:
    Sprite sprite;
    Vector2f F;
    string img_file;
    double R, x ,y, xVel, yVel, mass;

    Body();

    Vector2f getPos();

    double getM();

    void scale(Vector2u winSize, double R);

    virtual void draw(RenderTarget &target, RenderStates states) const;

    friend istream &operator>>(istream &in, Body &body);

    void time(double time);

    void move();

    void setV(double ax, double ay, double time);

    void setPos(double time);

    ~Body();
};

double getRadius(double body1_pos, double body2_pos);

double getForce(double mass1, double mass2, double r);

double dirF(double F, double dF, double r);



```

```cpp
#include "planets.hpp"

Body::Body() {}

Vector2f Body::getPos() {
    Vector2f pos(xVel, yVel);
    return pos;
}

double Body::getM() {
    return mass;
}

void Body::scale(Vector2u winSize, double R) {
    this->winSize = winSize;
    this->R = R;
}

void Body::draw(RenderTarget &target, RenderStates states) const {
    target.draw(sprite, states);
}

istream &operator >>(istream &in, Body &body) {
    in >> body.x >> body.y >> body.xVel >> body.yVel
        >> body.mass >> body.img_file;

    if (!body.texture.loadFromFile("nbody/" + body.img_file)) {
        cout << "Failed to load image " << body.img_file << endl;
        exit(EXIT_FAILURE);
    }
    body.sprite.setTexture(body.texture);
    return in;
}

void Body::time(double time) {
    double ax = F.x / mass;
    double ay = F.y / mass;

    setV(ax, ay, time);

    setPos(time);
}

void Body::setV(double ax, double ay, double time) {
    xVel -= (ax * time);
    yVel -= (ay * time);
}

void Body::setPos(double time) {
    x -= xVel * time;
    y -= yVel * time;
}

void Body::move() {
    double newX = ((x / R) * winSize.x/2) + winSize.x / 2;
    double newY = ((y / R) * winSize.y/2) + winSize.y / 2;
    Vector2f middle;
    middle.x = sprite.getTexture()->getSize().x / 2;
    middle.y = sprite.getTexture()->getSize().y / 2;
    sprite.setOrigin(middle);

    sprite.setPosition(newX, newY);
}

double getRadius(double body1_pos, double body2_pos) {
    return sqrt(pow(body1_pos, 2) + pow(body2_pos, 2));
}

double getForce(double mass1, double mass2, double r) {

    return (G * mass1 * mass2) / pow(r, 2);;
}

double dirF(double F, double dF, double r) {
    double fDir = F * dF / r;
    return fDir;
}

Body::~Body() {}
```

```
#include <sstream>

#include "planets.cpp"

#define BACKGROUND "background.jpg"

Vector2u WINSIZE, backSize;

int main(int argc, char* argv[]) {
    int N;
    double R;
    double T = atoi(argv[1]);
    double deltaT = atoi(argv[2]);
    vector<shared_ptr<Body>> bodies;
    double dx = 0,
           dy = 0,
           r = 0,
           f = 0,
           fX = 0,
           fY = 0;

    RenderWindow window(VideoMode(800, 800), "NBody");

    Sprite background;
    Texture texture;
    if (!texture.loadFromFile(BACKGROUND)) {
        cout << "Failed to load background" << endl;
        return EXIT_FAILURE;
    }

    Music music;
    if (!music.openFromFile("HEYYEYAAEYAAAEYAEYAA.ogg")) {
        cout << "Failed to load music" << endl;
        return EXIT_FAILURE;
    }
    music.play();

    Font font;
    font.loadFromFile("digital-7 (mono).ttf");
    Text text;
    text.setFont(font);
    text.setPosition(0, 0);
    text.setCharacterSize(24);
    stringstream timer;

    backSize = texture.getSize();
    WINSIZE = window.getSize(); //gets the window size

    double xScale = (double) WINSIZE.x / backSize.x;
    double yScale = (double) WINSIZE.y / backSize.y;

    background.setTexture(texture);
    background.setScale(xScale, yScale);

    window.setFramerateLimit(60);

    cin >> N;
    cin >> R;
    for (int i = 0; i < N; i++) {
        shared_ptr<Body> ptrBody(new Body());
        cin >> *ptrBody;
```

```cpp
        ptrBody->scale(WINSIZE, R);
        bodies.push_back(ptrBody);
    }

    Clock clock;
    clock.restart();

    while (window.isOpen())
    {
        Time ElapsedTime = clock.getElapsedTime();
        double timePassed = ElapsedTime.asSeconds();
        timer.str(string());
        timePassed *= deltaT;
        timer << "Time passed in seconds: " << timePassed;
        text.setString(timer.str().c_str());
        Event event;
        while (window.pollEvent(event))
        {
            if (event.type == Event::Closed || Keyboard::isKeyPressed(Keyboard::Escape))
                window.close();
        }

        window.clear();
        window.draw(background);
        window.draw(text);

        for (unsigned int i = 0; i < bodies.size(); i++) {
            fX = 0;
            fY = 0;
            for (unsigned int j = 0; j < bodies.size(); j++) {
                if (i != j) {
                    dx = bodies[j]->x - bodies[i]->x;
                    dy = bodies[j]->y - bodies[i]->y;
                    r = getRadius(dx, dy);
                    f = getForce(bodies[i]->getM(), bodies[j]->getM(), r);
                    fX += dirF(dx, f, r);
                    fY += dirF(dy, f, r);
                }
            }
            bodies[i]->F.x = fX;
            bodies[i]->F.y = fY;
            bodies[i]->time(deltaT);
            bodies[i]->move();
            window.draw(*bodies[i]);
        }

        if (timePassed > T || Keyboard::isKeyPressed(Keyboard::Escape) || event.type == Event::Closed) {
            for (unsigned int i = 0; i < bodies.size(); i++) {
                cout << " " << bodies[i]->x;
                cout << " " << bodies[i]->y;
                cout << " " << bodies[i]->xVel;
                cout << " " << bodies[i]->yVel;
                cout << " " << bodies[i]->mass;
                cout << " " + bodies[i]->img_file << endl;
            }
        }

        window.display();
    }
    return 0;
}
```