Particle Simulator OOP Assignment

Contents

1. Introduction

For this project, I decided to revisit an old project/interest of mine – physics simulation! It started out as just a gravity simulator, but I then realised it could be extended to a particle simulator. It has inter-particle gravitational forces, inter-particle collisions, wall collisions and local gravity (as if on earth). These forces can be turned on/off so that the user can make pretty much any scenario. The package is written in a way such that it can be used as a module, and custom scenarios can be written by the user. I have also written code for a GUI for it so that it can be easily demonstrated.

The motivation comes from the fact that my previous projects were written in python, but Java is faster than python, so I wanted to do it in Java to make it faster.

Under the hood, it uses the Barnes-Hut algorithm to improve performance. Brute force calculation of inter particle forces/collision checking is O(n^2) whereas the Barnes-Hut's Algorithm is O(n log(n)) meaning bigger simulations can be run.

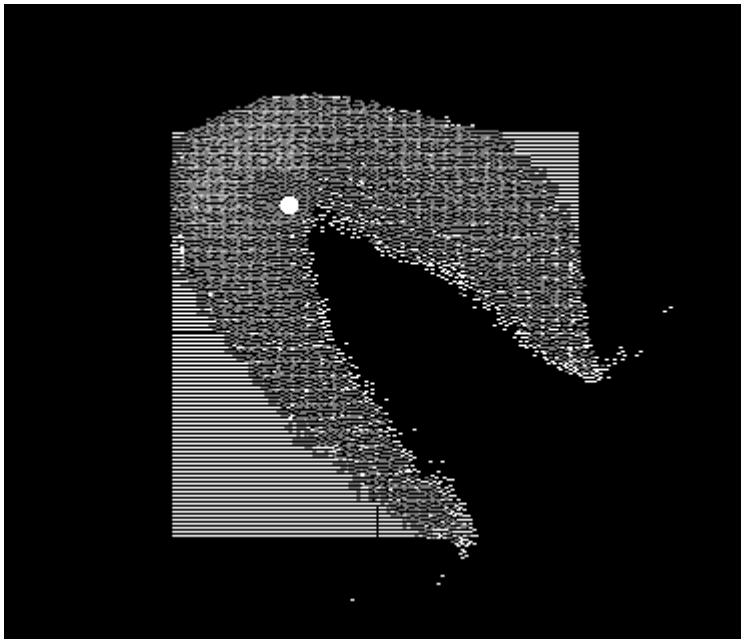
If you are interested in how the algorithm works, there is a pdf in this directory – or you can get it directly from my GitHub repo.

Overall, I'm quite proud of this project as it has the potential to make some beautiful visuals. See Below.

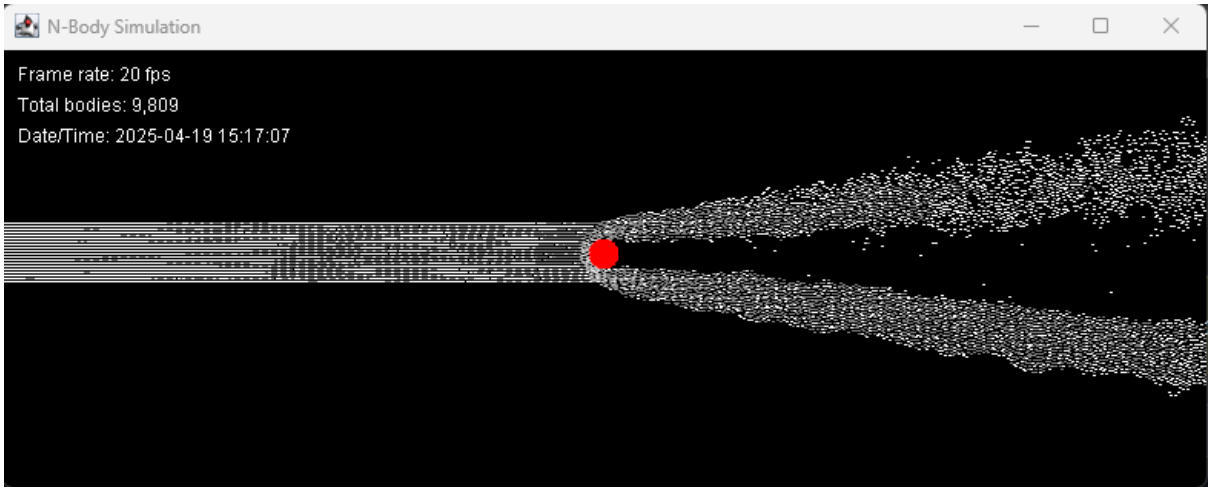
I highly recommend running and playing with this yourself. Use the following command:

java -XX:ActiveProcessorCount=HOW_MANY_CORES_YOU_HAVE -cp bin GUI

Ball passing through a grid of small light particles



A Stream of light particles colliding with a relatively massive ball, mimicking airflow

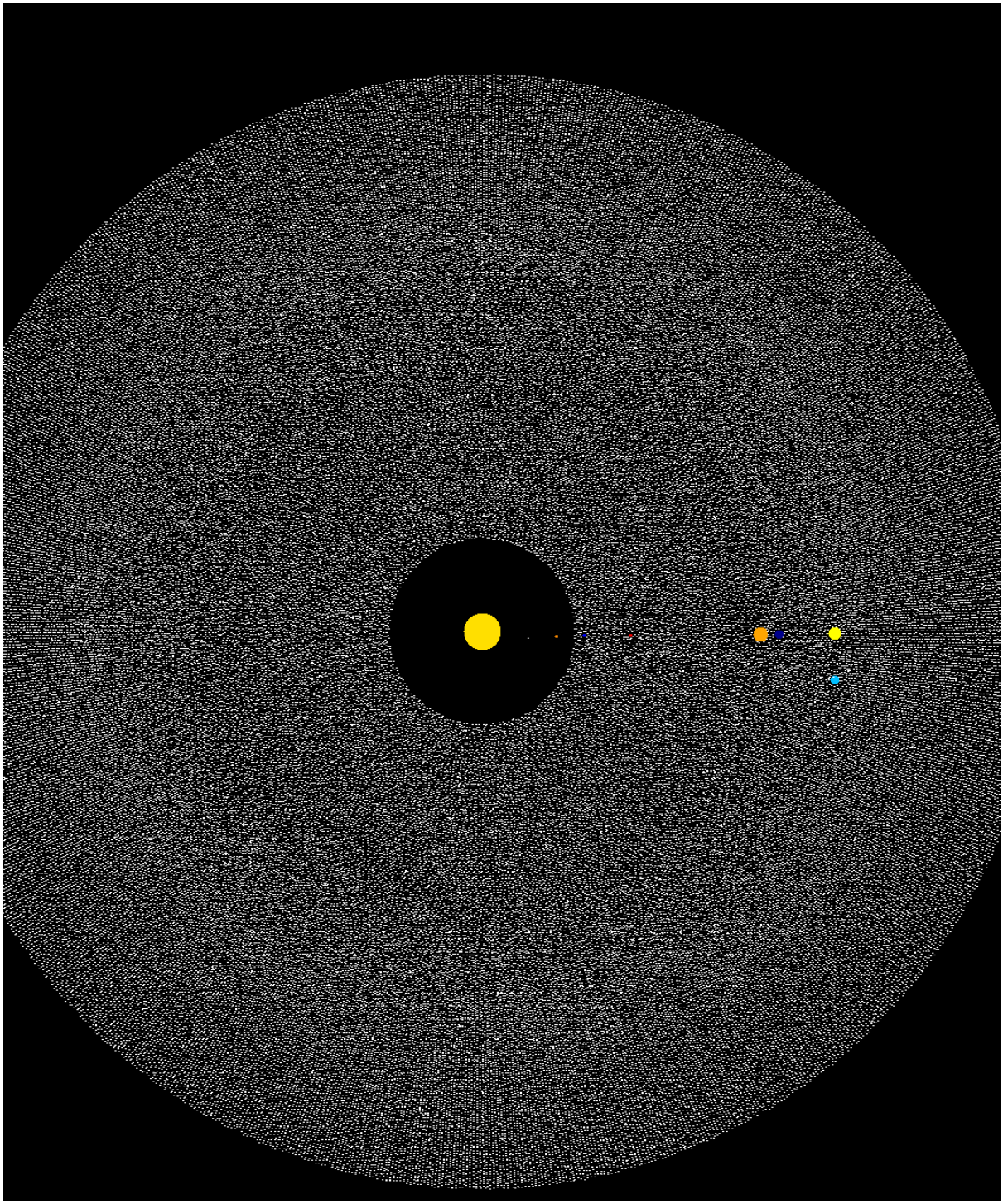


Below is very simple code to make the wind stream simulation, showcasing how a user of this package an easily use it to make their own custom scenario.

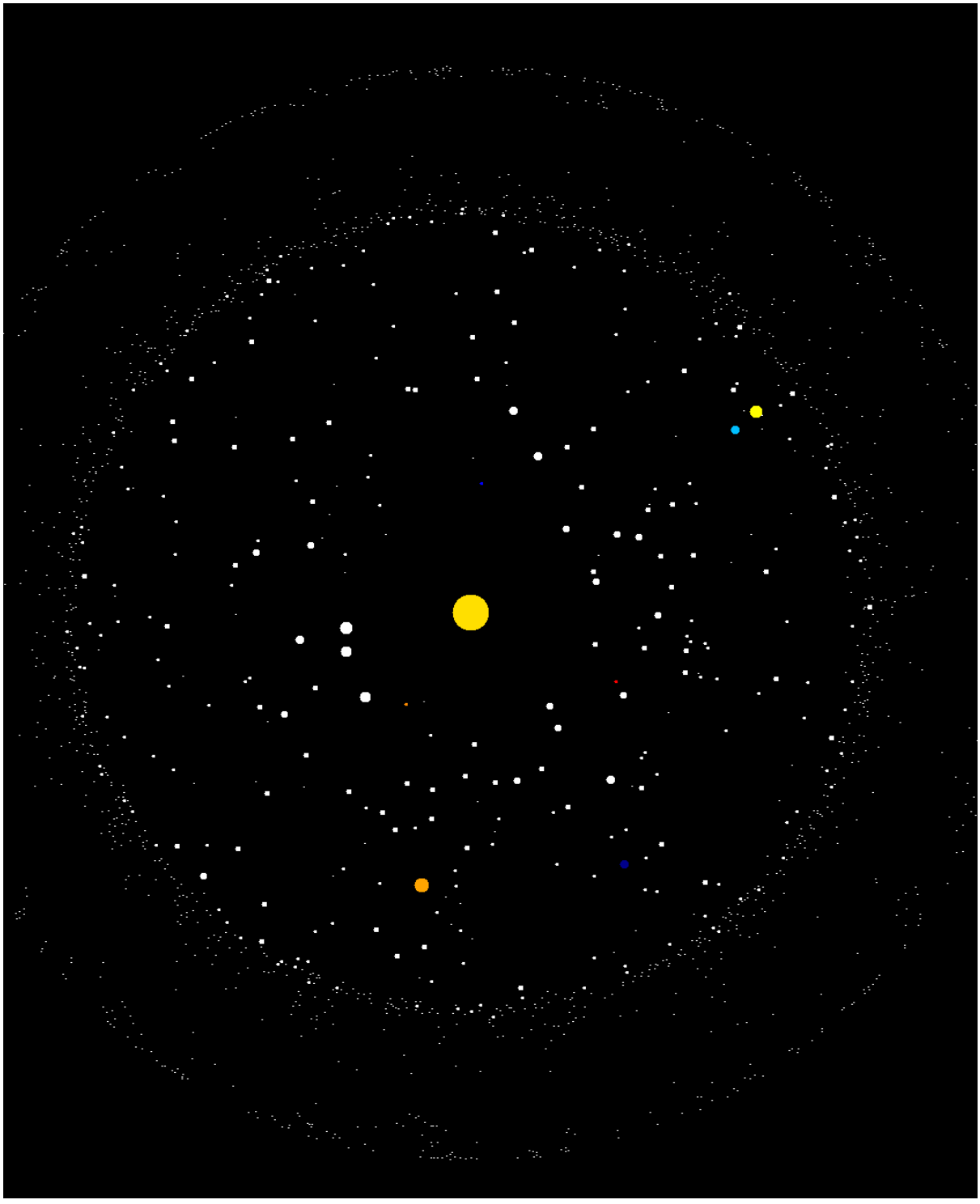
```
public static void windStream(JCheckBox drawQuadTree, JCheckBox StickyCollisions) {
    // 50k particles moving from left to right. There is a ball in the middle.
    //Ball has radius 10. So the stream should have height 20.

    double centreX, centreY,heightOfStream;
    int radiusBall = 10;
    heightOfStream = radiusBall*2;
    centreX = 168;
    centreY = 394.5;

    List<Body> bodies = new ArrayList<>();
```

```
        Body Ball = new Body(centreX,centreY,0,0,100,Color.RED);
        Ball.overRiddenRadius = radiusBall;
        bodies.add(Ball);
        Ball.elastic = 1.0;

        int numBodies = 20_000;
        double mass =1.0/numBodies;
        int overRiddenRadius = 1;
        double x = 0;
        double y = centreY-heightOfStream*overRiddenRadius;

        Random rand = new Random();
        rand.setSeed(0);
        for(int i = 0; i < numBodies/heightOfStream; i++){
            for(int j = 0;j<heightOfStream;j++){
                x = (j==0)?x-overRiddenRadius*2:x; // increment as j increases. reset to 0 when j = 0
                y = (j>0)?y+overRiddenRadius*2:centreY-heightOfStream*overRiddenRadius; //increment as
i increases. No need to reset
                double vx = 15;
                double vy = 0;
                Color color = Color.WHITE;
                Body b = new Body(x, y, vx, vy, mass, color);
                b.overRiddenRadius = overRiddenRadius;
                b.changeColorOnCollision = true;
                b.SwitchColor = new Color(255,255,255,125);
                b.elastic = 0.0;
                bodies.add(b);
            }
        }

        Simulation sim = new Simulation(bodies, .1, Double.POSITIVE_INFINITY,1000,drawQuadTree,
StickyCollisions);
        sim.interParticleCollisions = true;
        sim.graviationalForceField = false;
        sim.wallCollisions = false;
        sim.parallel = true;
        sim.oneLoop = true;
        sim.simulate();

    }
```

Formation of a solar system



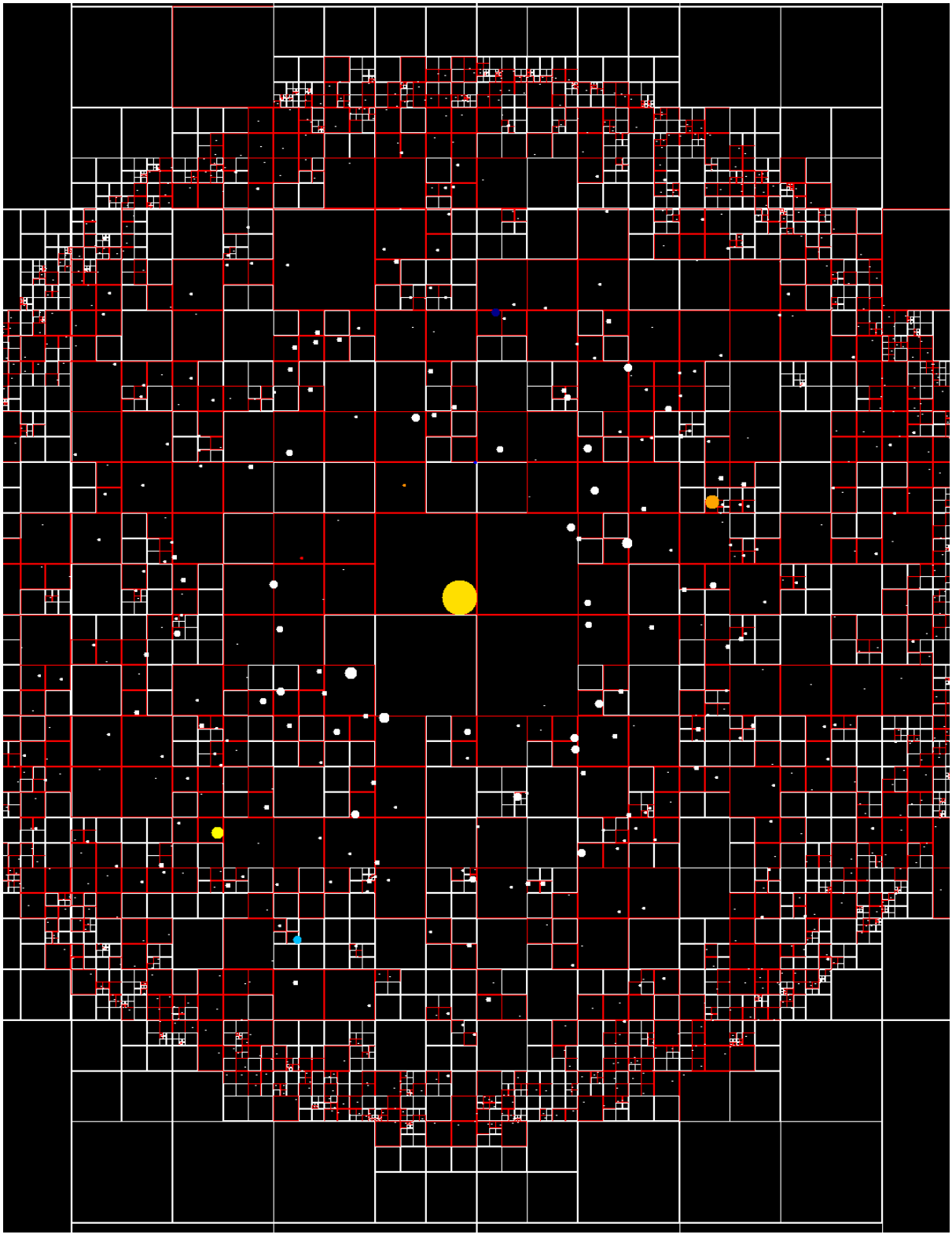
Initially, and



After some time...

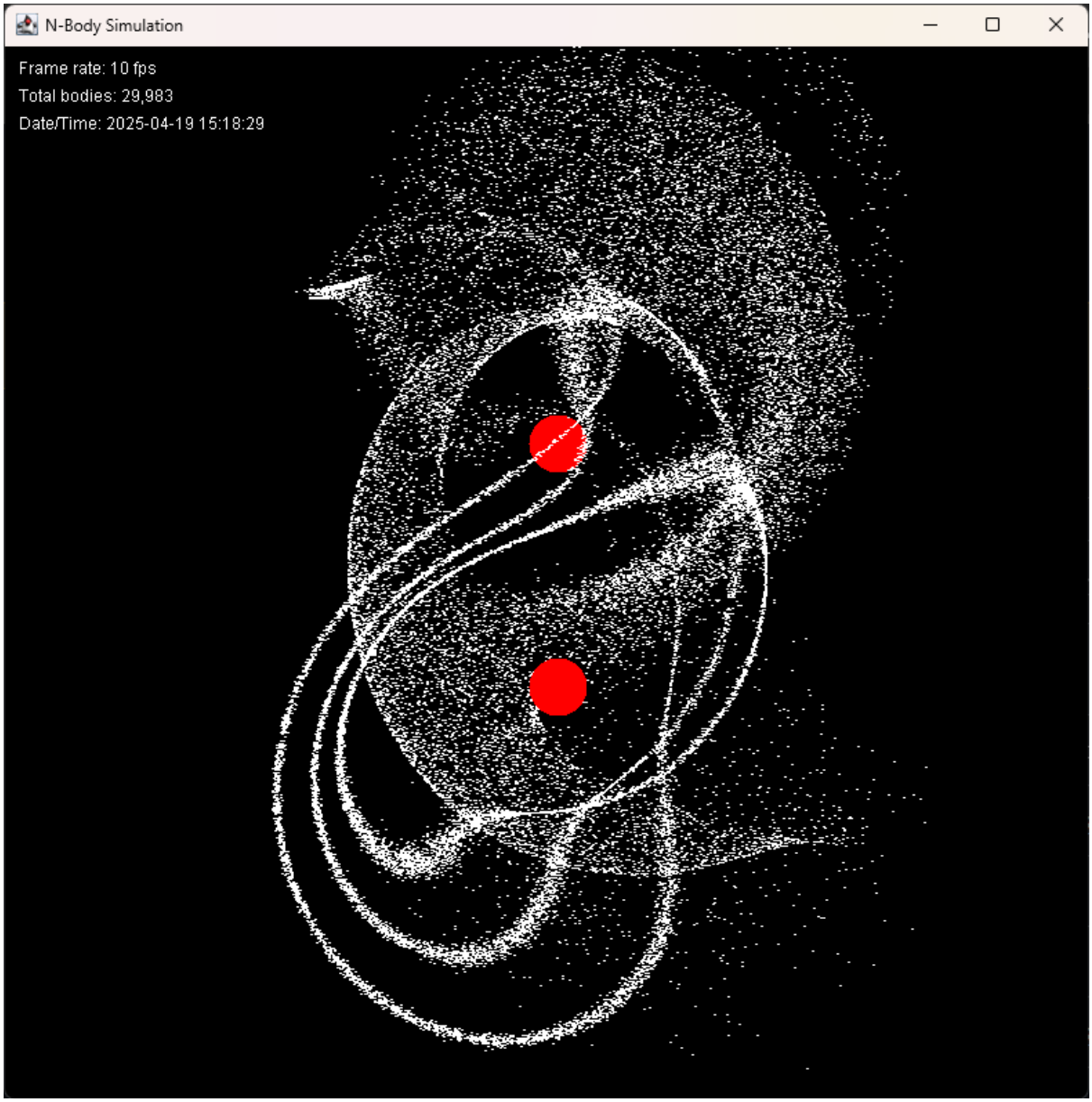
Most particles stick together and become satellites of the star, while the rest make their way out to wider orbits and form a sort of asteroid belt

Below is the same image but with a visualisation of the quad tree used in the BH algorithm.

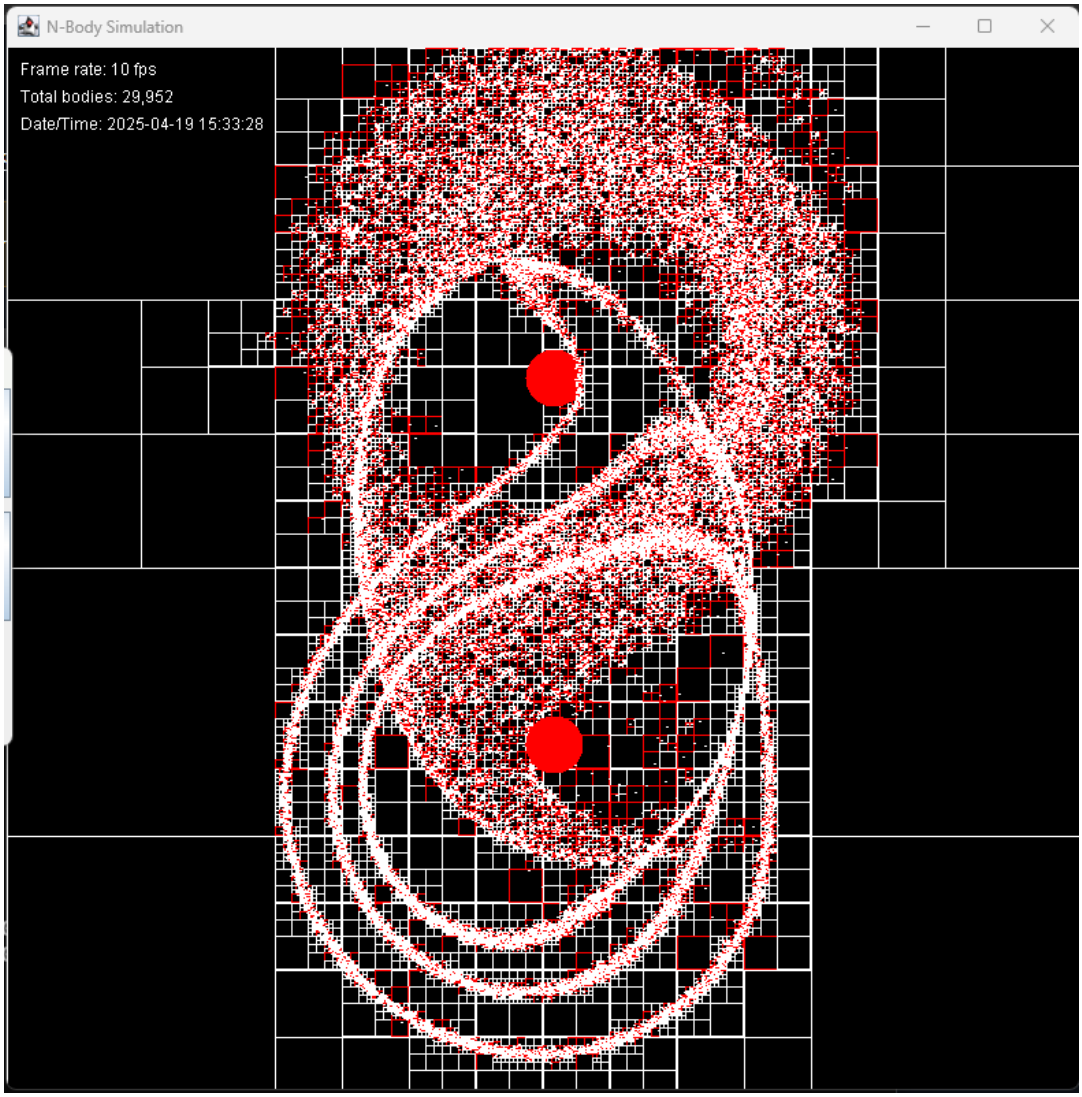


Red squares have are leaves whereas white squares are branches.

Two Galaxies colliding



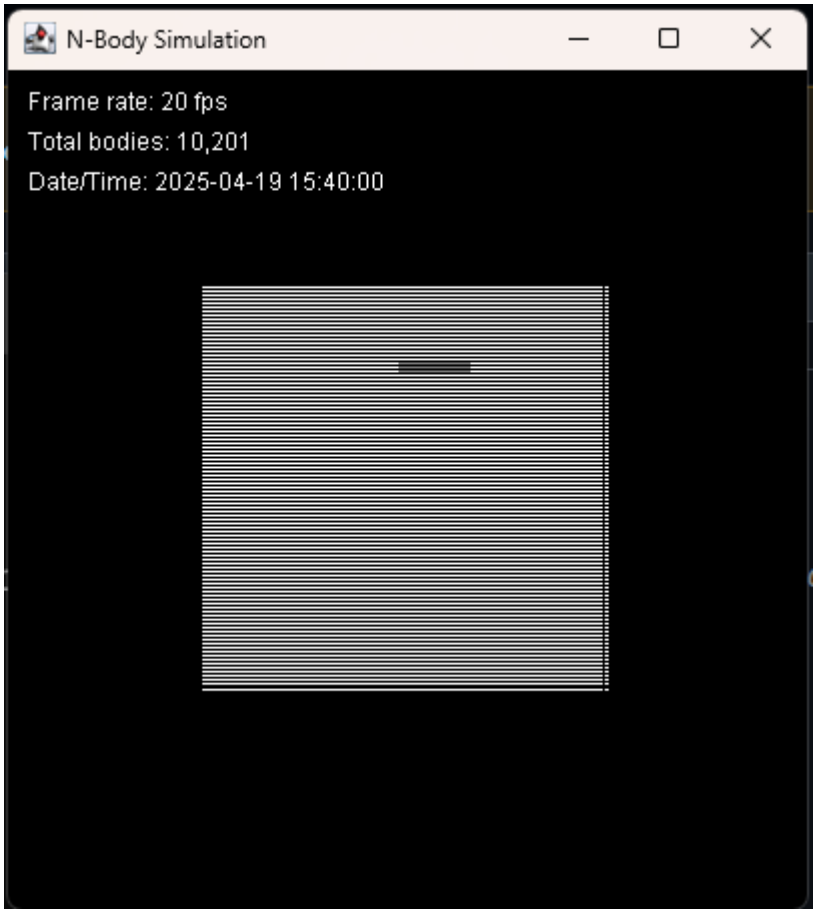
Below with tree visualized



2. User Stories

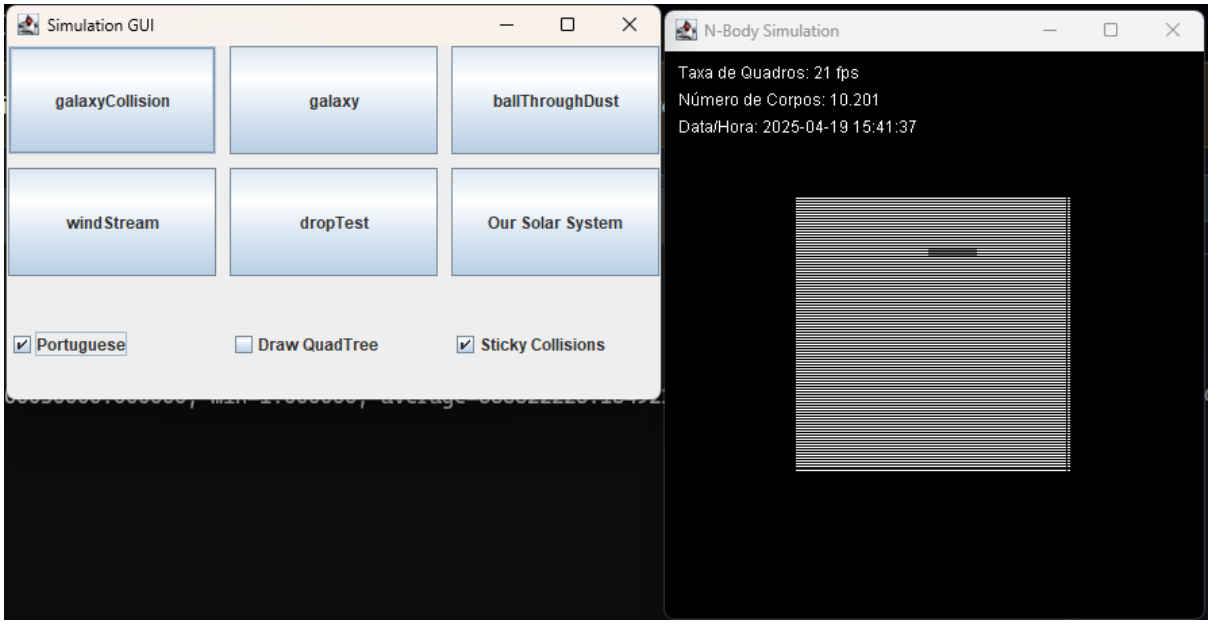
The user can use this package in their project. I will not speculate on how they might use it, as I believe it is beyond the scope of this assignment. The user can also run an interactive GUI with 6 pre-written scenarios. Below I will talk about these and I will also mention some of the functionality incorporated into the GUI.

Starting with the following base case. I am using the ballThroughDust Scenario for consistency between screenshots.



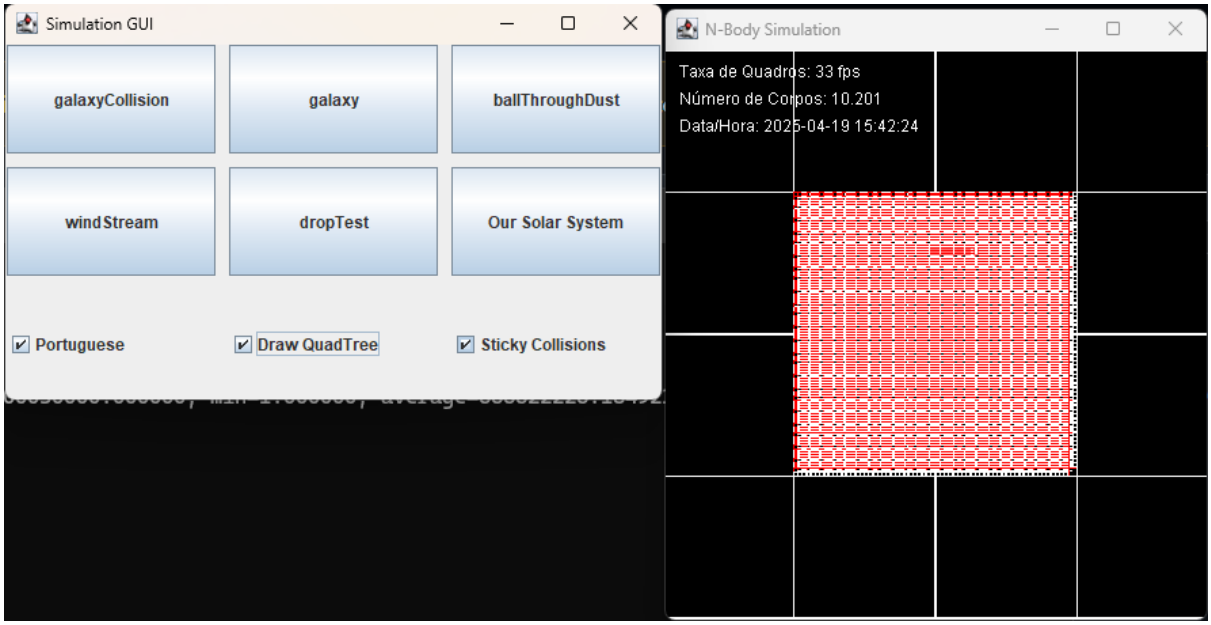
2.1. Locale – English/Portuguese

The user can press the Portuguese checkbox to switch the Locale from English to Portuguese and vice-versa. I chose Portuguese randomly.



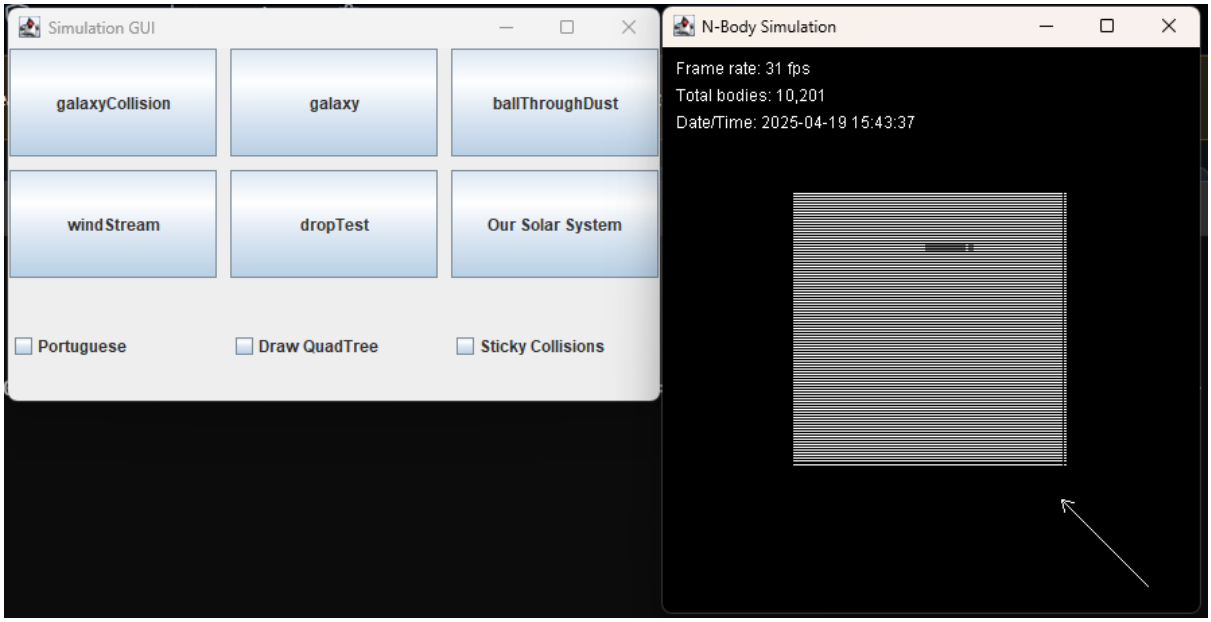
2.2. DrawQuadTree

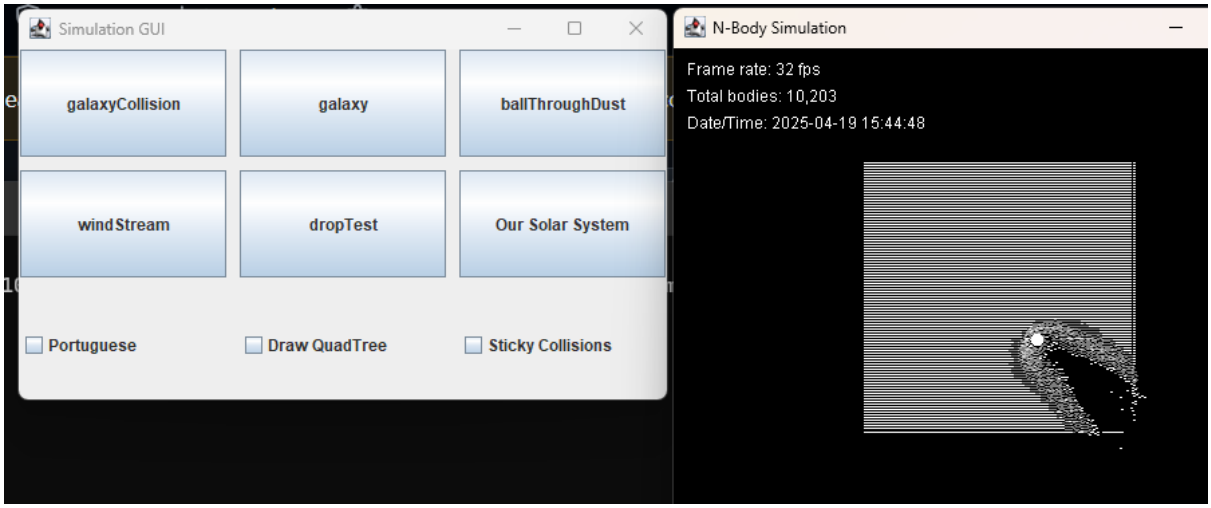
The User can check the Draw Quadtree Check box to turn on/off the visualization of the Barnes-hut algorithm



2.3. Shooting balls

The user can shoot balls around the “universe” by clicking, dragging and releasing on he GUI canvas.





This was actually hard to screenshot, but you'll see it in the video.

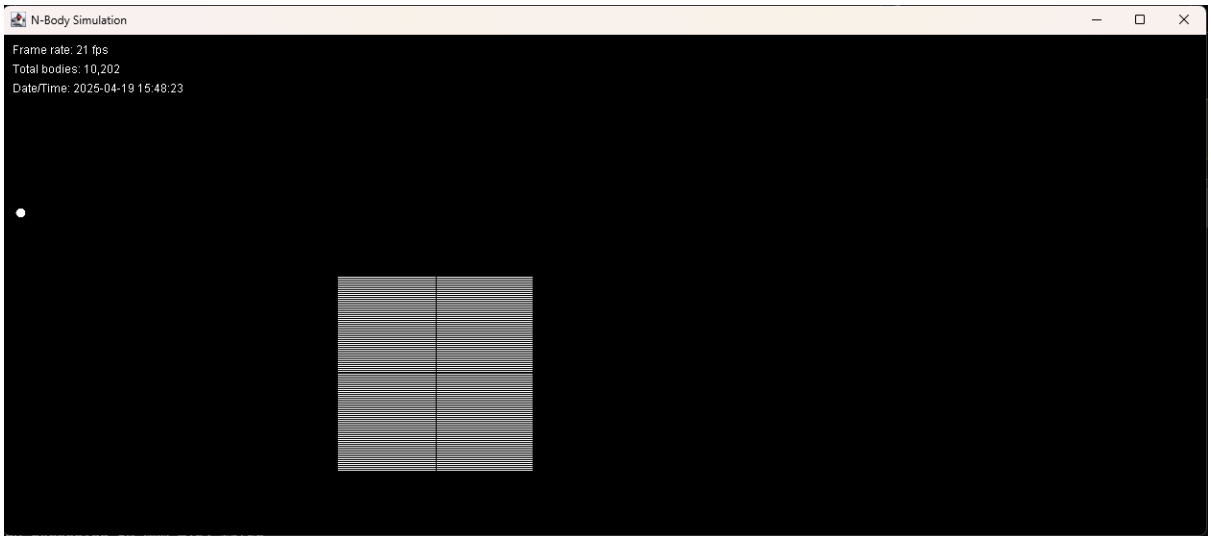
2.4. StickyCollisions

If the user ticks the Sticky Collisions box on the GUI, then the balls added through clicking and dragging on the GUI will be StickyBody objects, as opposed to Body objects. StickyBody Objects merge when they collide, unlike their parent class, Body, which bounce off each other according to Newtonian mechanics.

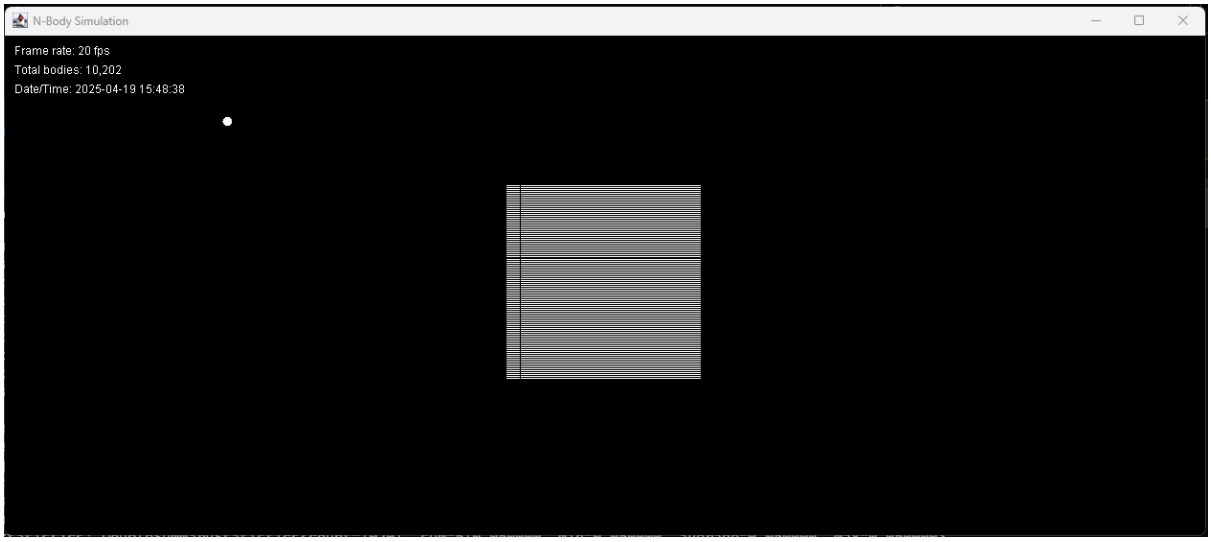
2.5. Recentre

The user can recentre all the bodies on the screen by double right clicking on the Canvas. This is achieved by subtracting the centre of mass from each bodies position.

Before

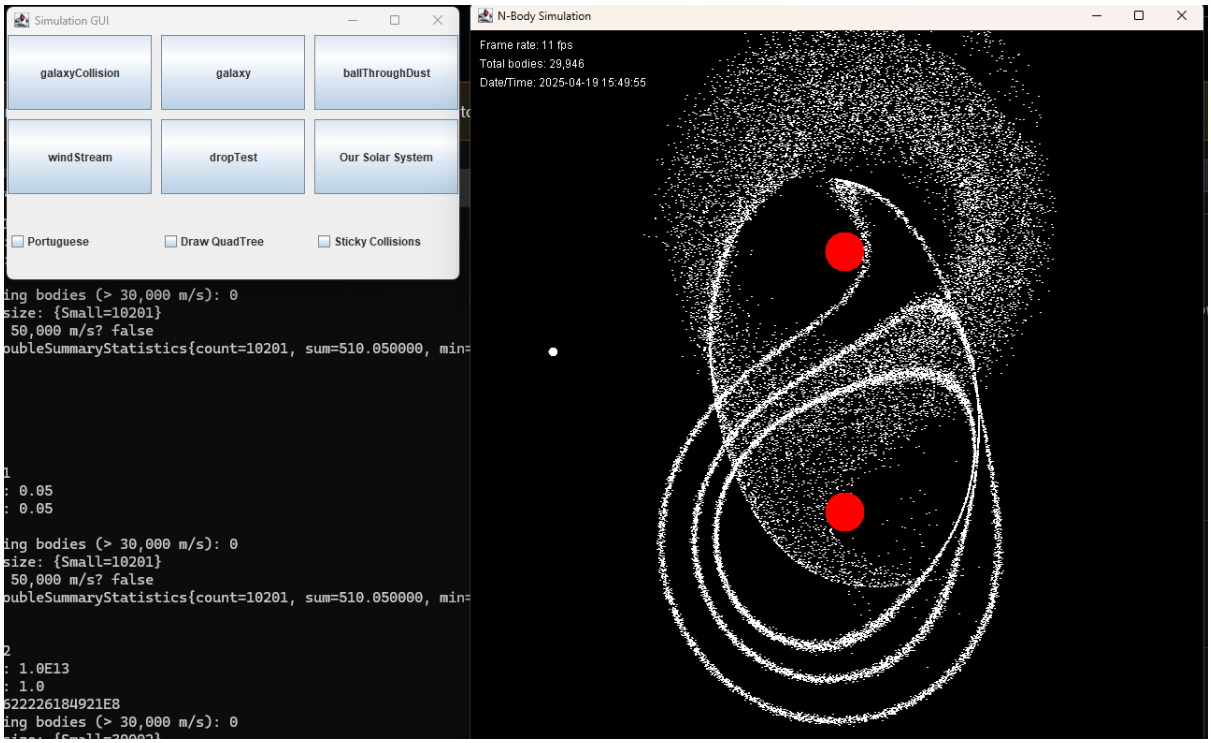


After



2.6. GalaxyCollision

The user can play and interact with the Galaxy Collision simulation by pressing the galaxyCollision button on the GUI

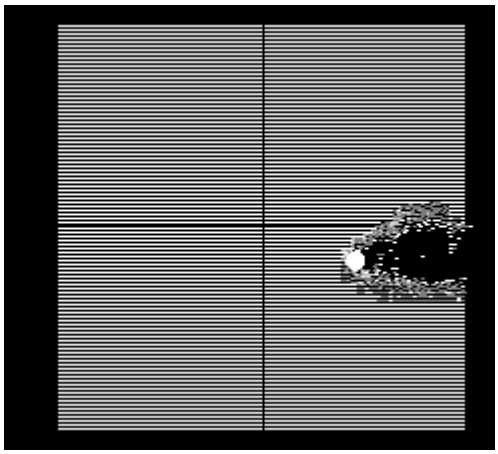


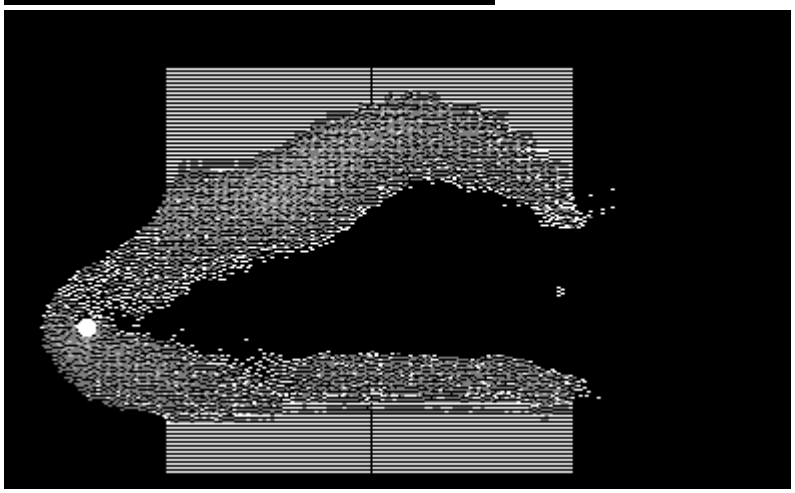
2.7. Galaxy

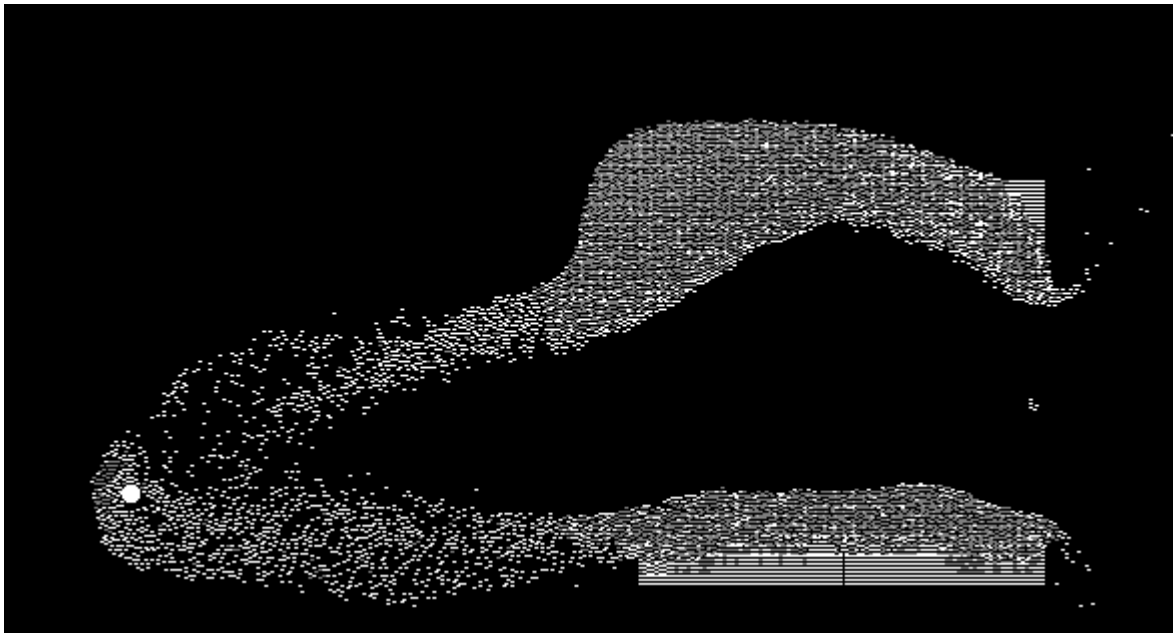
Similar to above just slightly different – honestly not that impressive.

2.8. ballThroughDust

The user can load the ballThroughDust simulation. This is poorly named as of now as the user has to shoot the balls themselves – but in an earlier iteration before I added the drag and shoot functionality, I programmed a ball to just fall through it. This is one of my favourite simulations and I wasted a lot of time just playing with it when I should have been coding.

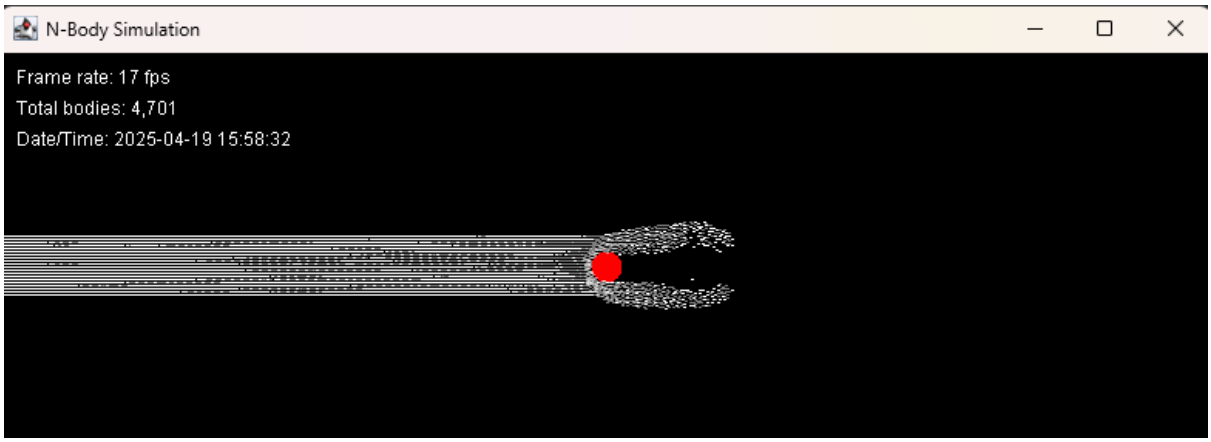


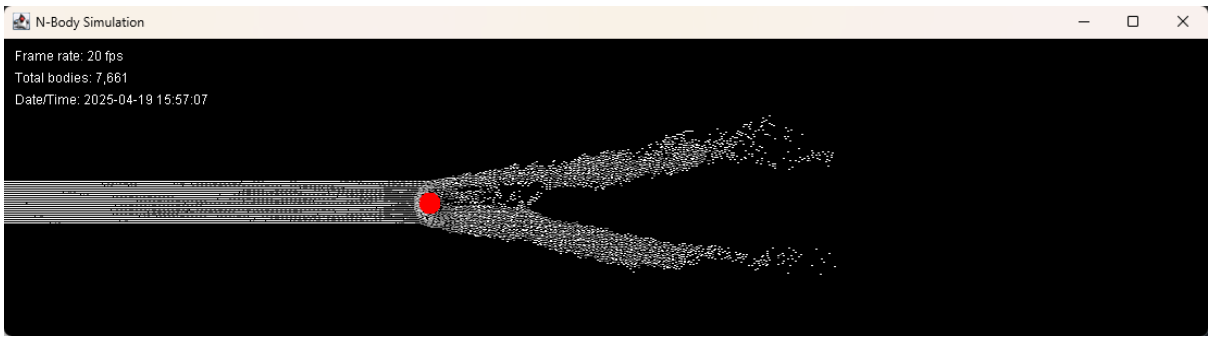


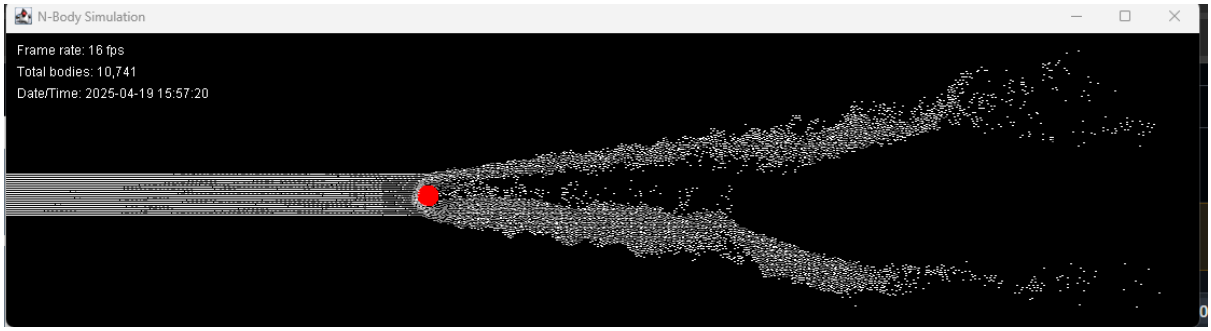


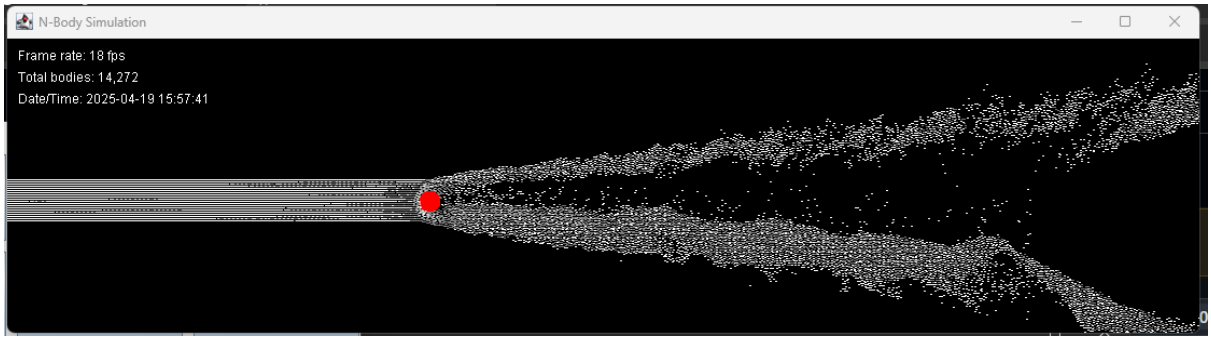
2.9. windStream

This is another interesting one. If you play with the masses, radii and elasticity of ‘dust particles’ and shoot them at a relatively massive object, it mimics gas flowing around a ball in a vacuum.



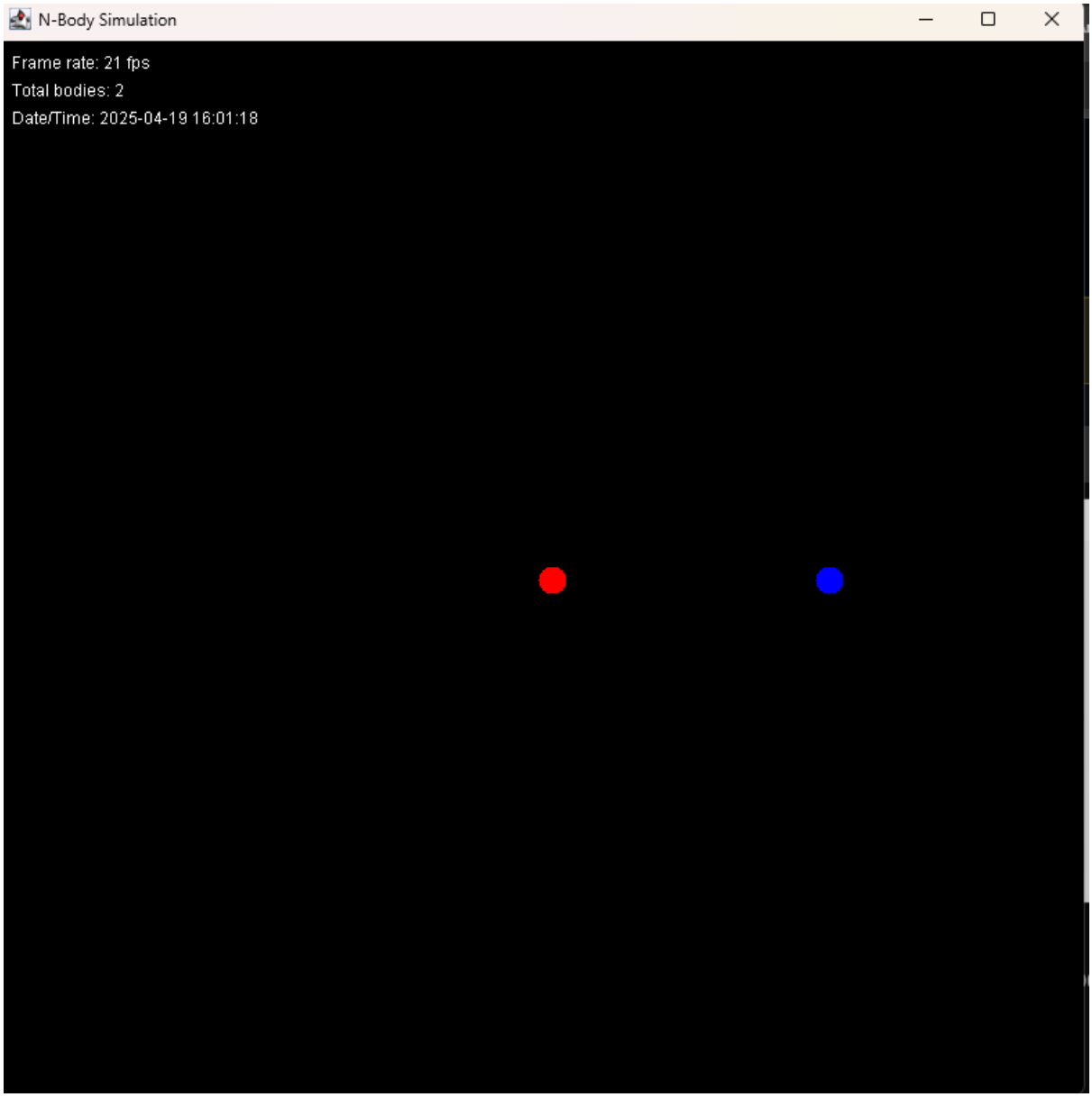


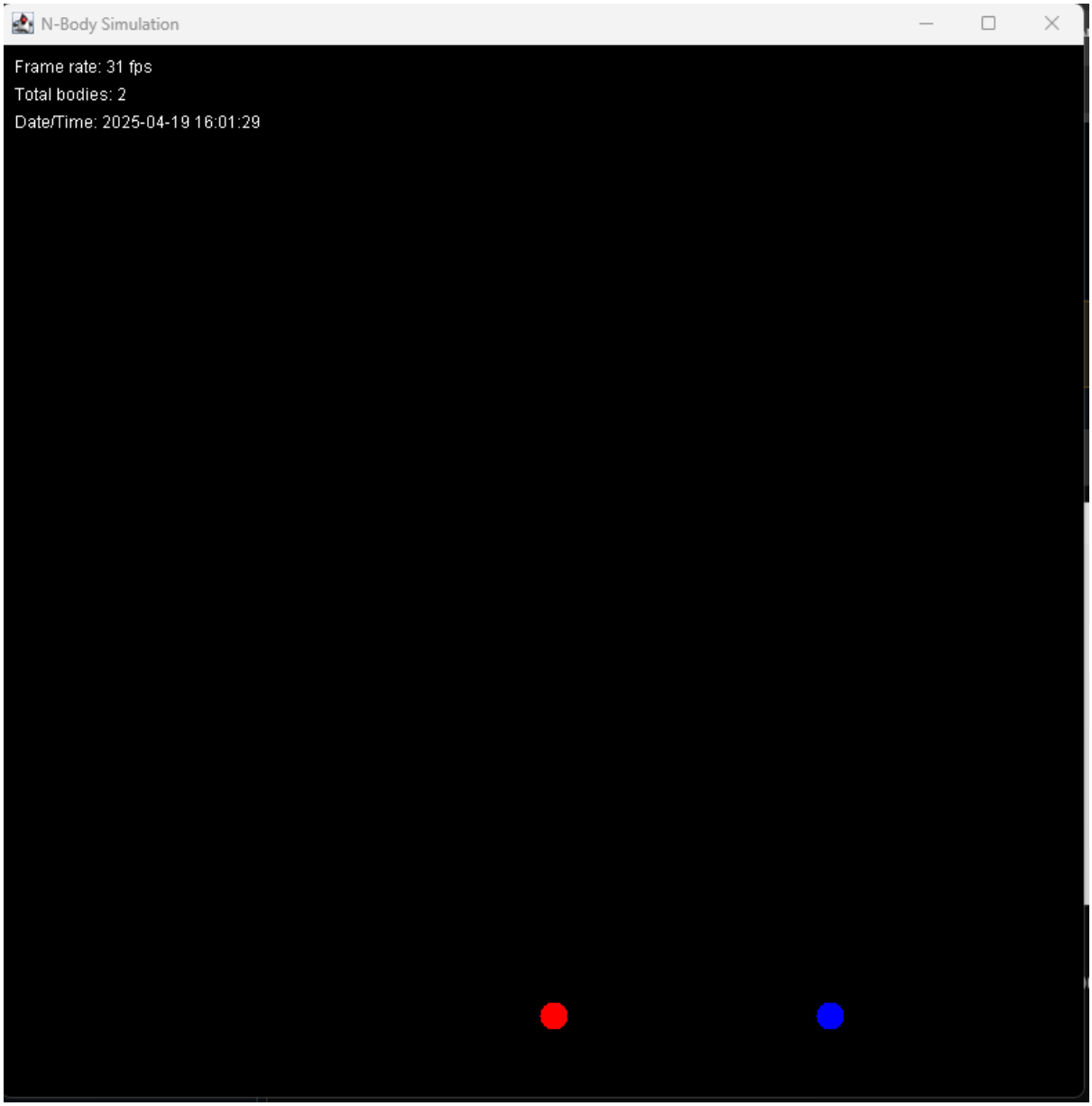




2.10. dropTest

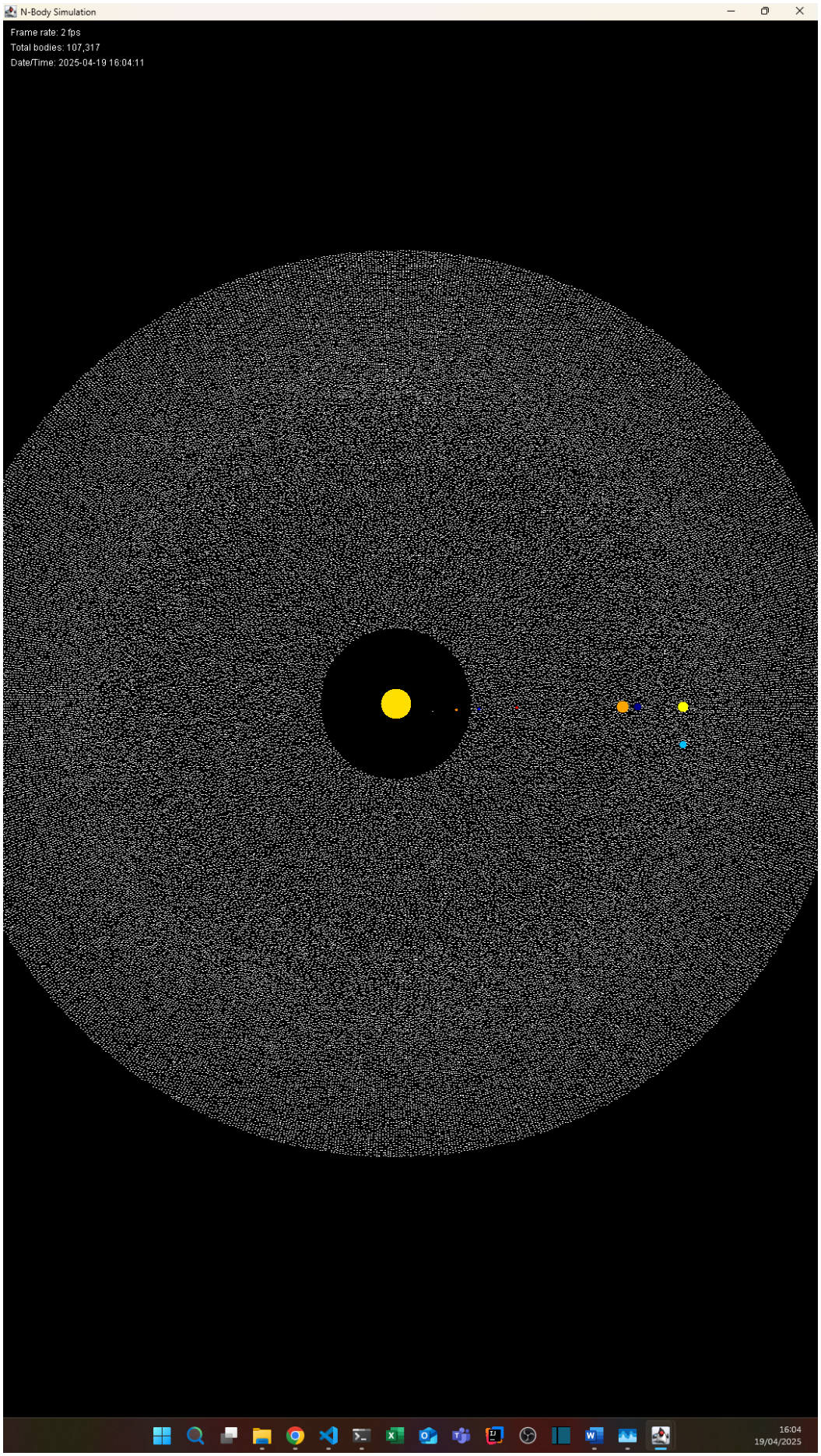
This is just a scenario showing local gravity (acceleration due to gravity on earth) and wall collisions at work. The test is to see if two balls of very different mass will fall at the same rate and maintain a bounce. The balls also have imperfect elasticity and so lose energy over time.


N-Body Simulation
Frame rate: 21 fps
Total bodies: 2
Date/Time: 2025-04-19 16:01:18

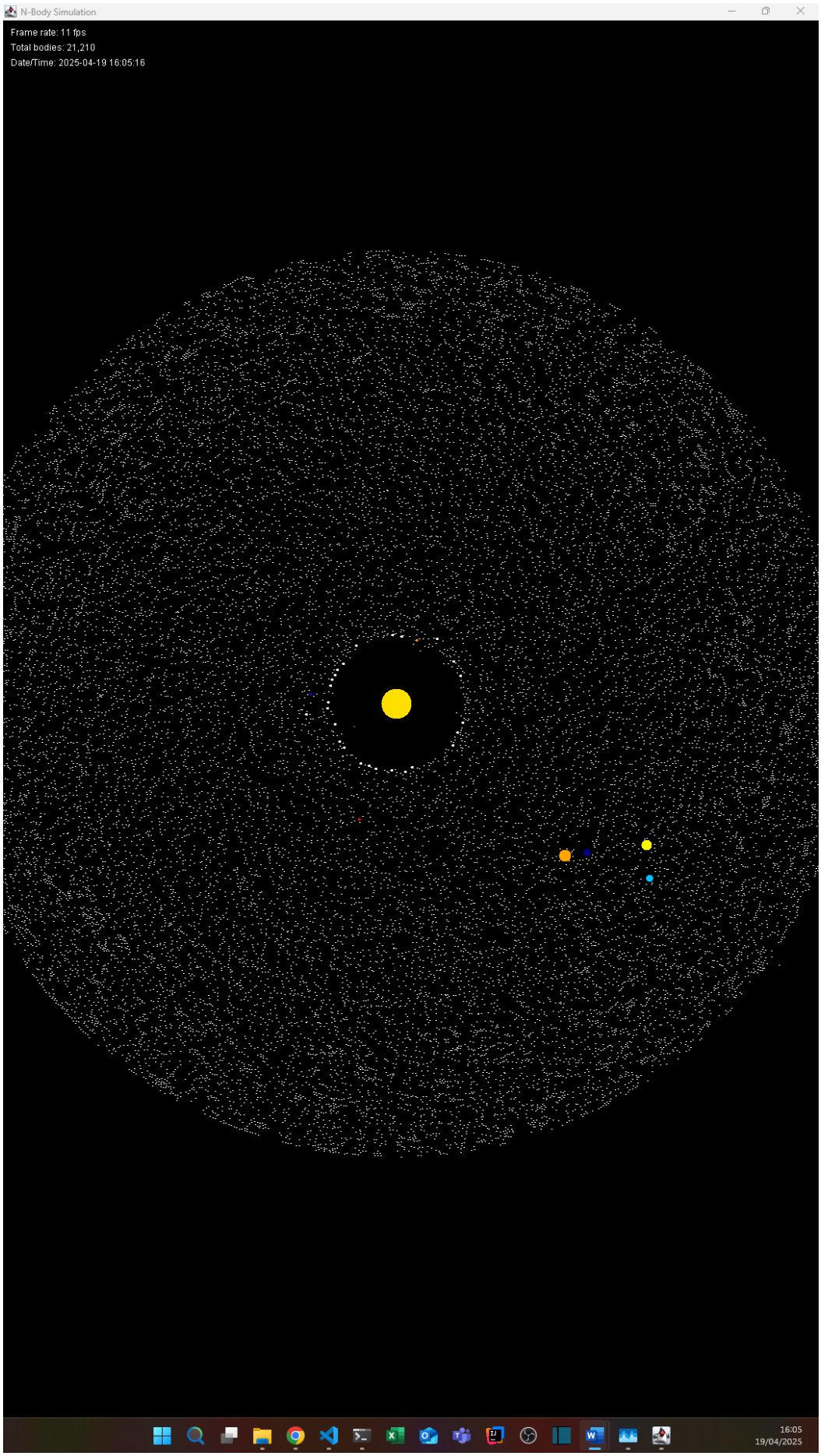


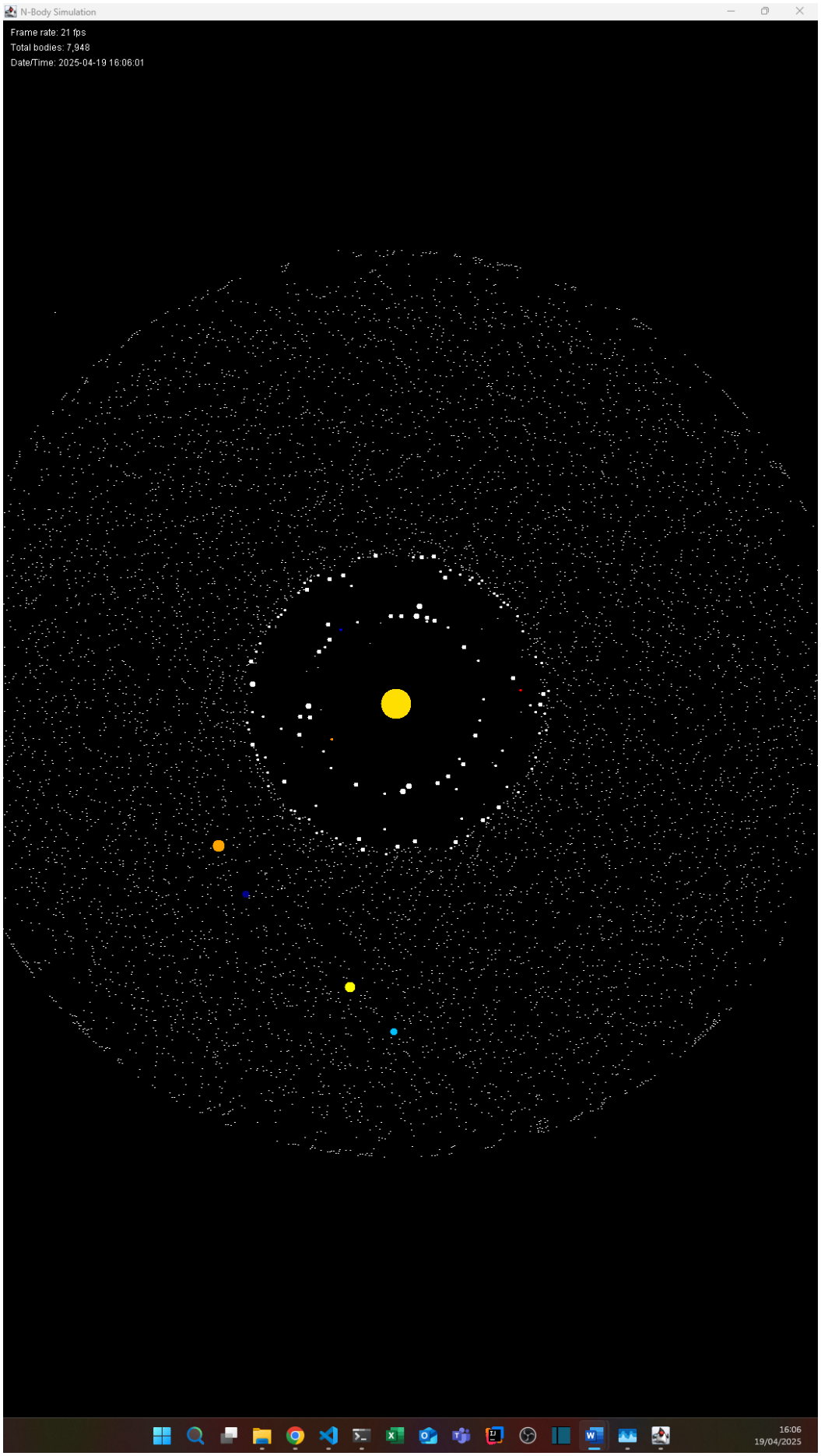
2.11. Our Solar System

This is another good scenario. I tried to replicate our own solar system, using initial conditions, but it didn't work very well. However, it still looks great. It starts out with 1,000,000 StickyBodys, which rapidly declines

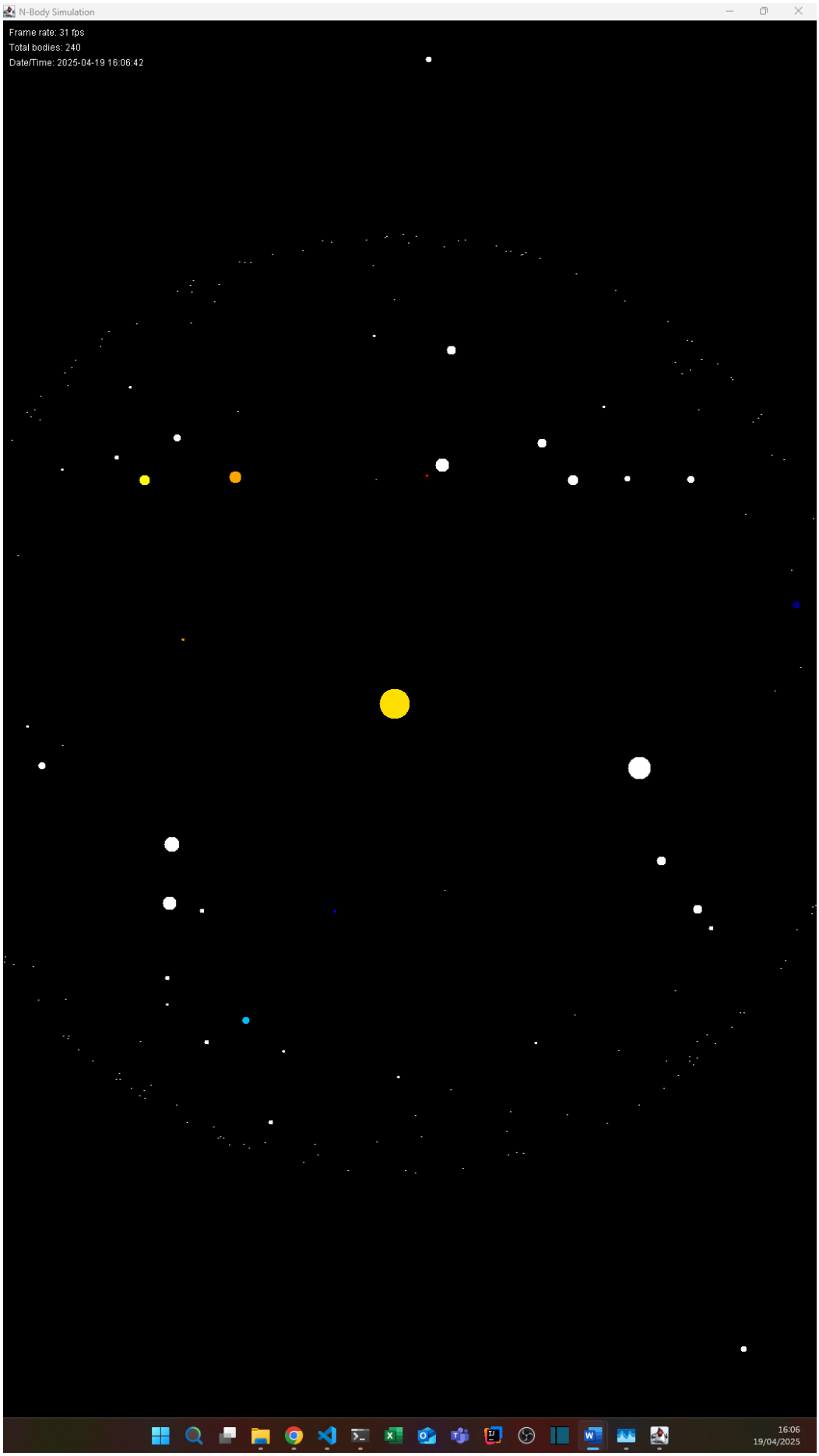

N-Body Simulation
Frame rate: 2 fps
Total bodies: 107,317
Date/Time: 2025-04-19 16:04:11
16:04
19/04/2025

They all start to collide and form moons and planets, while some particles get pushed out into wider orbits, forming an asteroid like belt


N-Body Simulation
Frame rate: 11 fps
Total bodies: 21,210
Date/Time: 2025-04-19 16:05:16
16:05
19/04/2025


N-Body Simulation
Frame rate: 21 fps
Total bodies: 7,948
Date/Time: 2025-04-19 16:06:01
16:06
19/04/2025

But it eventually settles down into a stable solar system, after 30 minutes or so.


N-Body Simulation
Frame rate: 31 fps
Total bodies: 240
Date/Time: 2025-04-19 16:06:42
16:06
19/04/2025

Sometimes, with some look, you get moons orbiting planets. You can also force it by adding particles to the simulation manually.

3. Evaluation

3.1. Fundamentals

3.1.1. Lambdas

When the user loads a simulation, statistics are printed out the terminal, basically summarizing the particles on the screen.

```
Total bodies: 20001
Heaviest body mass: 100.0
Lightest body mass: 5.0E-5
Average Mass: 0.0050497475126243686
Number of fast-moving bodies (> 30,000 m/s): 0
Bodies grouped by size: {Small=20001}
Any body exceeding 50,000 m/s? false
Mass statistics: DoubleSummaryStatistics{count=20001, sum=101.000000, min=0.000050, average=0.005050, max=100.000000}
Number of cores: 7
```

To do this, Consumer, Predicate and Function are used.

3.1.2. Streams

As above with the statistics, Stream are used along with both terminal and intermediate operations such as group by, filter, min max etc.

3.1.3. Switch expressions and pattern matching

Sticky body's need to know if they are colliding with another body or another sticky body – to this, a pattern matching switch expression is used.

```
public void collide(Body other){
    switch (other) {
        case StickyBody sb -> privateCollide(this, sb);
        default -> super.privateCollide(this, other);
    }

}
```

3.1.4. Sealed classes and interfaces

To read in data from a text file, I made a DataFrame object. It uses two objects, Column and Row. Which both extend a sealed abstract class, the class implements a sealed interface. The purpose of this is to make them iterable.

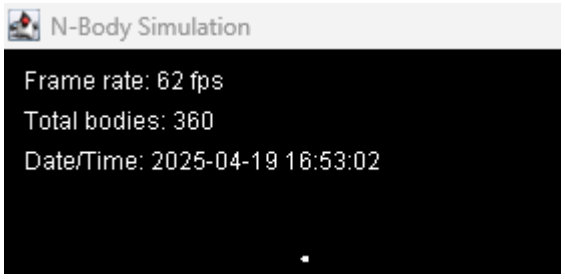
```
final public class Column<T extends Number> extends DataFrameSubClass<T> {
```

```
final public class Row extends DataFrameSubClass<Object> {
```

```java
public sealed interface DataFrameInterface<T extends Object> extends Iterable<T> permits DataFrameSubClass{
    public T[] getValues();

    @Override
    public default Iterator<T> iterator(){
        return new Iterator<T>(){
            private int index = 0;

            public boolean hasNext() {
                return index < getValues().length;
            }

            public T next() {
                return getValues()[index++];
            }
        };
    }
}

sealed abstract class DataFrameSubClass<T extends Object> implements DataFrameInterface<T> permits Column, Row {
    public String name;
    public T[] values;

    public T[] getValues(){
        return this.values;
    };
}
```

This interface is just an Iterable<T> wrapper essentially and is quite redundant. I just wanted to tick it off this list

3.1.5. Date/Time API



GUI displays the current datetime.

3.1.6. Records

I didn't use records in this as I thought there was no need. All my objects have to be mutable.

3.2. Advanced

3.2.1. Collections/generics

You will find these in the statistics.java file

```java
public class Statistics {
    public static void print(Simulation s) {
        List<Body> bodies = s.bodies;

        // Count total bodies
        long totalBodies = bodies.stream().count();
        System.out.println("Total bodies: " + totalBodies);

        // Find heaviest and lightest body (Using mass getter)
        //Consumer
        Consumer<Body> printHeaviestBodyMass = body -> System.out.println("Heaviest body mass: " + body
        bodies.stream().max(Comparator.comparing(Body::getMass))
            .ifPresent(printHeaviestBodyMass);
        bodies.stream().min(Comparator.comparing(Body::getMass))
            .ifPresent(b -> System.out.println("Lightest body mass: " + b.getMass()));

        // Average mass
        double avgMass = bodies.stream().mapToDouble(Body::getMass).average().orElse(other:0);
        System.out.println("Average Mass: " + avgMass);
```

3.2.2. Concurrency

I use stream.ParallelStream to update physics in some of the scenarios. In the bigger scenarios I use my custome oneLoop method.

```java
public ExecutorService executor = Executors.newFixedThreadPool(Runtime.getRuntime().availableProcessors());
```

```java
    public void updateOneLoop() throws InterruptedException{
        // gravity
        // collsions
        // wall collisions
        double com_x = 0;
        double com_y = 0;

        if(this.reCenter){
            double offset = 0;
            double _com_x = 0;
            double _com_y = 0;
            double total_mass = 0;
            for(Body body:this.bodies){
                offset = Math.max(body.scaledRadius(), offset);
                _com_x += body.getX()*body.getMass();
                _com_y += body.getY()*body.getMass();
                total_mass += body.getMass();
            }
            com_x = _com_x / total_mass+offset;
            com_y = _com_y / total_mass+offset;
        }

        // not alot of freedom with this one, but it's faster so is reserved for big sims.
        for (Body body : this.bodies) {
            final double com_x_ = com_x;
            final double com_y_ = com_y;
```

```
            executor.submit(() -> {
                if (this.graviationalForceField){
                    this.tree.updateForce(body);
                }
                if (this.interParticleCollisions){
                    this.tree.updateCollisions(body, this.collisionThreshold);
                }
                if (this.wallCollisions){
                    double x = body.getX();
                    double y = body.getY();
                    double vx = body.getVx();
                    double vy = body.getVy();
                    double r = body.scaledRadius();
                    double correctDirection = 0;
                    int width = this.getWidth();
                    int height = this.getHeight();
                    if(x - r < 0 || x + r > width){
                        correctDirection = x - r< 0 ? 1 : -1;
                        vx = correctDirection*Math.abs(vx)*body.elastic;
                        x = x - r < 0 ? r+1 : width-r-1;
                    }if(y - r < 0 || y + r > height){
                        correctDirection = y - r < 0 ? 1 : -1;
                        vy = correctDirection*Math.abs(vy)*body.elastic;
                        y = y - r < 0 ? r+1 : height-r-1;
                    }
                    body.setPosition(x, y);
                    body.setVelocity(vx, vy);
                }
                if (this.reCenter){
                    body.setPosition(body.getX() - com_x_ + this.getWidth()/2, body.getY() - com_y_ +
this.getHeight()/2);
                }
                body.update(this.dt);
                body.resetForce();
            });
        }

        // initially, I would let the executor shutdown, which will give an accurate simulation.
        // But if I don't shut it down, I don't have the overhead of recreating a new executor every
time.
        // Downside is that more jobs get pushed to it while it's still working, so it can lead to
some weird behavior.
        // upside is that the animations are still beautiful and smooth to the naked eye, while
imrpoving FPS.

        // executor.shutdown();
        // executor.awaitTermination(1, TimeUnit.MINUTES);
        // executor.
```

```
    }
```

This pushed updates through an executor service through all the cores on the CPU. It doesn't shutdown after each call, which is inaccurate, however it improves performance as there is no overhead from garbage collection and reinitiating each time.

3.2.3. NIO2

I used NIO2 to read in from a text file for initial conditions for the Our Solar System, I made a custom TextFileReader.

```
public class TextFileReader {
    public List<String> lines;

    public void read(Path filePath) {
        try {
            List<String> lines = Files.readAllLines(filePath);
            this.lines = lines;
        } catch (IOException e) {
            System.err.println("Error reading file: " + e.getMessage());
        }
    }
    public static Path sourceDirectory() {
        try {
            Path sourceDir = new File(TextFileReader.class.getProtectionDomain()
                                    .getCodeSource()
                                    .getLocation()
                                    .toURI()).toPath().getParent();

            return sourceDir;
        } catch (Exception e) {
            System.err.println("Error getting source directory: " + e.getMessage());
            return null;
        }
    }
    public static void main(String[] args) {
        Path filePath = sourceDirectory();
        filePath = filePath.resolve("data/SolarSystem.txt");
        TextFileReader reader = new TextFileReader();
        reader.read(filePath);

        DataFrame df = new DataFrame(reader.lines.toArray(new String[0]),
                                        new Class[] {Double.class, Double.class, Double.class,
Double.class, Double.class,Integer.class, Integer.class, Integer.class, Integer.class});
        System.out.println(df);
    }
}
```

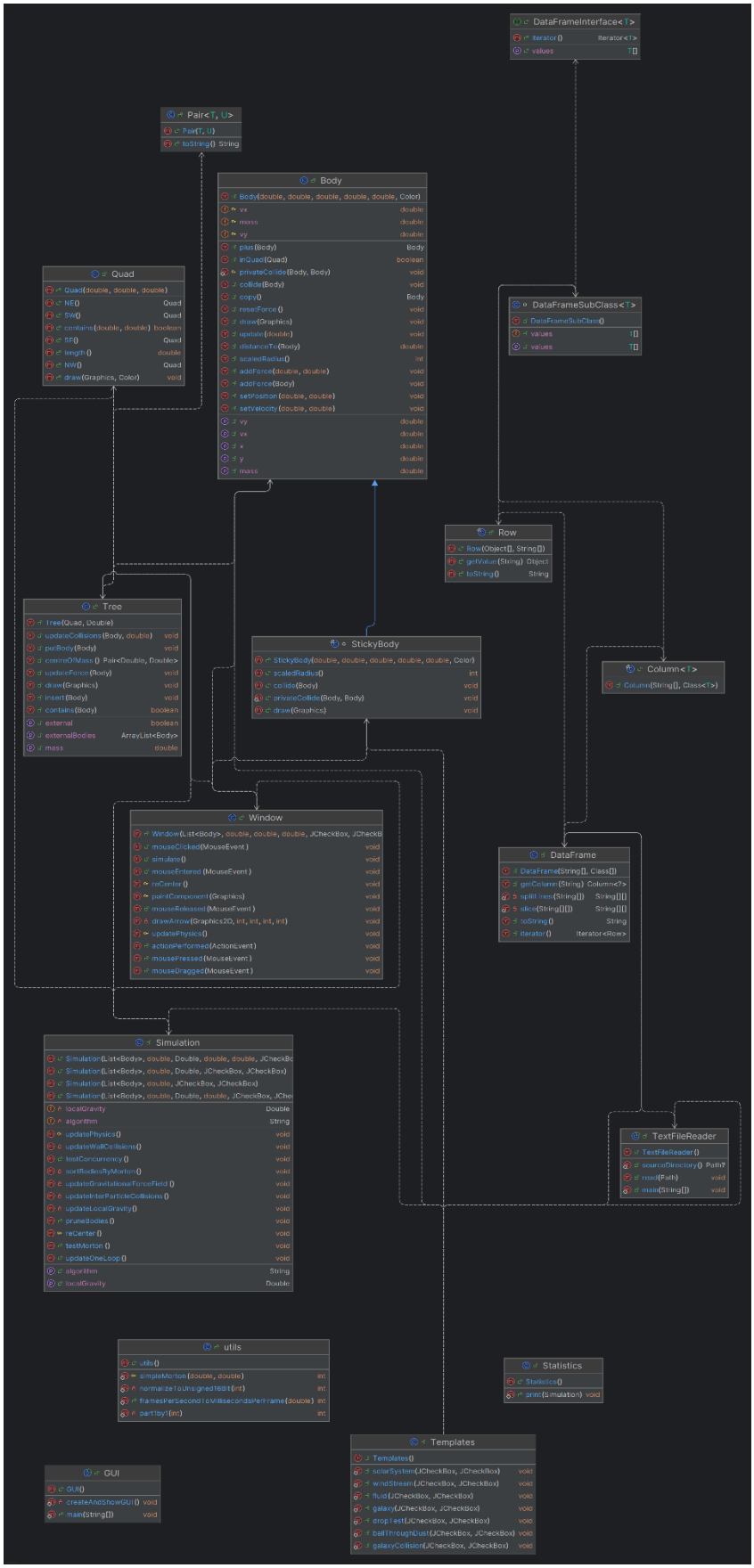
3.2.4. Localization

The GUI has a button where the user can switch from Portuguese to English very quickly. To do this, I created the files above, added a button to the GUI that just switches the default Locale from English to Portuguese or vice-versa.

```
JCheckBox portuguese = new JCheckBox(text:"Portuguese");
portuguese.addActionListener(_ -> Locale.setDefault(portuguese.isSelected() ? new Locale(language:"pt", country:"BR") : Locale.ENGLISH));
panel.add(portuguese);
panel.add(drawQuadTree);
panel.add(StickyCollisions);
```

```
public class GUI {

    Run | Debug
    public static void main(String[] args) {
        Locale.setDefault(Locale.ENGLISH);
        SwingUtilities.invokeLater(GUI::createAndShowGUI);
    }
```

4. UML


DataFrameInterface<T>
Pair<T, U>
Body
Quad
DataFrameSubClass<T>
Row
Tree
StickyBody
Column<T>
Window
DataFrame
Simulation
TextFileReader
utils
Statistics
Templates
GUI

- Body.java <- particle - mass, acceleration, velocity etc
- GUI.java <- preloaded templates
- Pair.java<- insignificant little thing I needed briefly
- Quad.java <- Leaf/Branch of the Quad Tree
- Simulation.java <- main loop/ configuration of the simulation
- Statistics.java <- simple stuff to print statistics out to the console
- StickyBody.java<- sub class of body which is sticky
- Templates.java <- all of my scenarios
- Tree.java <- Barnes-hut tree. Handles updating collisions and gravity very well
- utils.java <- Morton sort etc. Apparently sorting the bodies according to their Morton numbers speeds up the calculation of the tree, however, I didn't notice any gains.
- Window.java <- super class of simulation. Simulation handles physics specific stuff. Window handles interactivity/repainting specific stuff. Helps separate the code into logical portions .It is an abstract class
-

+---IO <- all of these are just to read in the text file

- Column.java
- DataFrame.java
- DataFrameInterface.java
- Row.java
- TextFileReader.java
-

+---resources <- localization

messages_en.properties

messages_pt.properties

messages_pt_BR.propertie

Barnes Hut algorithm

A brute force approach to creating an n-body simulation of n particles is as follows.

For each particle1:

    For each particle2:

        F = Gm1m2/d^2

        Particle1 = F

This, if optimized, requires n(n+1)/2 computations.

The barnes Hut algorithm is as follows:

For each particle:

    For each quadrant:

        Set d = the distance between the particle and the quad

        Let l = the width of the quadrant

        If l/d > theta:

            Then iterate through the quads children and recursively call this function

        Else:

            The force on particle is calculated by approximating the quadrant as a point mass, using it's centre of mass and summing the masses of everything inside.

Where theta is defined as an accuracy parameter. If theta = 0, the algorithm is equivalent to brute force. The higher it is, the less accurate, but more performant. This achieves an computational complexity of O(n log (n))

In plain English, when calculating the force on a particle. If a quadrant is sufficiently far away, there is no point in calculating the force every particle in the quadrant exerts on your reference particle, just use its centre of mass. There is a mathematical proof using Taylor expansion demonstrating that this is a sufficient means of approximation – I believe it's called multi-pole expansion.

Anyways, I hope you like my project as much as I do.