Particle Simulator OOP Assignment

Contents

[1. Introduction 2](#_Toc195974916)

[2. User Stories 9](#_Toc195974917)

[2.1. Locale – English/Portuguese 10](#_Toc195974918)

[2.2. DrawQuadTree 11](#_Toc195974919)

[2.3. Shooting balls 11](#_Toc195974920)

[2.4. StickyCollisions 12](#_Toc195974921)

[2.5. Recentre 12](#_Toc195974922)

[2.6. GalaxyCollision 13](#_Toc195974923)

[2.7. Galaxy 13](#_Toc195974924)

[2.8. ballThroughDust 14](#_Toc195974925)

[2.9. windStream 15](#_Toc195974926)

[2.10. dropTest 16](#_Toc195974927)

[2.11. Our Solar System 18](#_Toc195974928)

[3. Evaluation 25](#_Toc195974929)

[3.1. Fundamentals 25](#_Toc195974930)

[3.1.1. Lambdas 25](#_Toc195974931)

[3.1.2. Streams 25](#_Toc195974932)

[3.1.3. Switch expressions and pattern matching 25](#_Toc195974933)

[3.1.4. Sealed classes and interfaces 25](#_Toc195974934)

[3.1.5. Date/Time API 26](#_Toc195974935)

[3.1.6. Records 26](#_Toc195974936)

[3.2. Advanced 26](#_Toc195974937)

[3.2.1. Collections/generics 26](#_Toc195974938)

[3.2.2. Concurrency 27](#_Toc195974939)

[3.2.3. NIO2 29](#_Toc195974940)

[3.2.4. Localization 30](#_Toc195974941)

[4. UML 31](#_Toc195974942)

# 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](https://github.com/Andrewnolan13/ParticleSimulator).

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 black and white image of a whale

AI-generated content may be incorrect.

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

A screenshot of a computer

AI-generated content may be incorrect.

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

A circular black and yellow circle with dots

AI-generated content may be incorrect.

Initially, and

A yellow and white circle with dots in the center

AI-generated content may be incorrect.

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.

A circular pattern of lines and dots

AI-generated content may be incorrect.

Red squares have are leaves whereas white squares are branches.

#### Two Galaxies colliding

A screenshot of a computer

AI-generated content may be incorrect.

Below with tree visualized

A screenshot of a computer

AI-generated content may be incorrect.

# 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.

A screenshot of a computer

AI-generated content may be incorrect.

## 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.

A screenshot of a computer

AI-generated content may be incorrect.

## DrawQuadTree

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

A screenshot of a computer

AI-generated content may be incorrect.

## Shooting balls

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

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

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

## 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.

## 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

A screen shot of a computer

AI-generated content may be incorrect.

After

A screen shot of a computer

AI-generated content may be incorrect.

## GalaxyCollision

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

A screenshot of a computer

AI-generated content may be incorrect.

## Galaxy

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

## 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.

A black and white screen with a white circle

AI-generated content may be incorrect. A black and white image of a wave

AI-generated content may be incorrect.

A black and white image of a cloud of particles

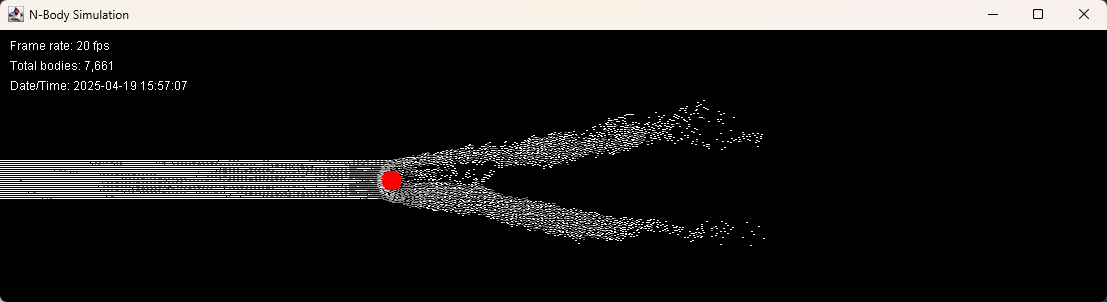
AI-generated content may be incorrect.

## 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.

A screen shot of a computer screen

AI-generated content may be incorrect.



A white particles in the sky

AI-generated content may be incorrect.

A close-up of a spray

AI-generated content may be incorrect.

## 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.

A screenshot of a computer with Marfa lights in the background

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

## 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

A black circle with white dots

AI-generated content may be incorrect.

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

A black circle with white dots and a yellow circle in the middle

AI-generated content may be incorrect.

A black background with many small white dots

AI-generated content may be incorrect.

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

A yellow circle in space

AI-generated content may be incorrect.

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

## Evaluation

## Fundamentals

## Lambdas

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

A black screen with white text

AI-generated content may be incorrect.

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

## Streams

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

## 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.

A screen shot of a computer code

AI-generated content may be incorrect.

## 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.





A screen shot of a computer program

AI-generated content may be incorrect.

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

## Date/Time API

A screenshot of a computer

AI-generated content may be incorrect.

GUI displays the current datetime.

## Records

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

## Advanced

## Collections/generics

You will find these in the statistics.java file

A screen shot of a computer program

AI-generated content may be incorrect.

## Concurrency

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



    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.

## 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);

    }

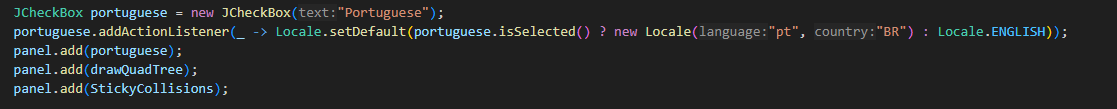
}

## Localization

A black rectangular object with a grey stripe

AI-generated content may be incorrect.

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.



A screen shot of a computer code

AI-generated content may be incorrect.

## UML

A screenshot of a computer screen

AI-generated content may be incorrect.

ª Body.java <- particle - mass, acceleration, velocity etc

ª GUI.java <- preloaded templates

ª Pair.java<- insignificant little thing I needed briefly

ª Quad.java <- Leaf/Branch of the Quad Tree

ª Simulation.java <- main loop/ configuration of the simulation

ª Statistics.java <- simple stuff to print statistics out to the console

ª StickyBody.java<- sub class of body which is sticky

ª Templates.java <- all of my scenarios

ª Tree.java <- Barnes-hut tree. Handles updating collisions and gravity very well

ª 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.

ª 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

ª

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

ª Column.java

ª DataFrame.java

ª DataFrameInterface.java

ª Row.java

ª TextFileReader.java

ª

+---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.