PROJECT REPORT

Simulation of molecular dynamics

MODELING OF PHYSICAL SYSTEMS

Patryk Gałczyński, Konrad Najder

Saturday 16th June, 2018

1 Project Aims and Objectives

Main goal of this project was to simulate, visualize and analyze multi body collisions in micro scale also known as molecular dynamics. For the simplicity sake of this project, simulation is based on classical Newton mechanics principles. That means, no quantum effects are being considered.

2 Selected methods and principles

Since this simulation is within inertial frame of reference, classical Newton mechanics is obvious choice for this type of simulation.These laws can be described as:

1. In an inertial frame of reference, an object either remains at rest or continues to move at a constant velocity, unless acted upon by a force.

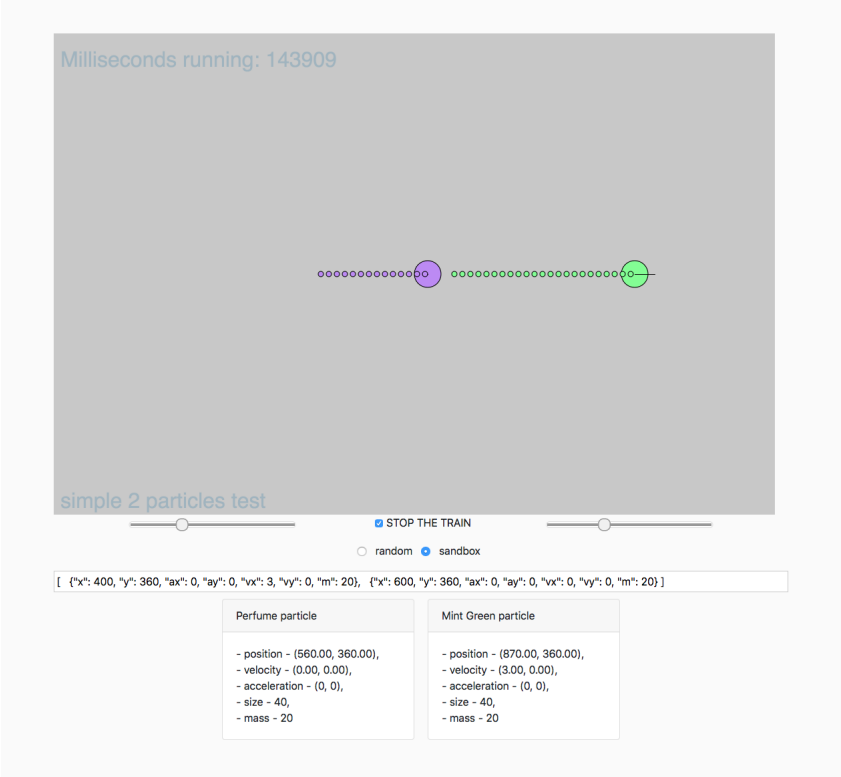


Figure 2.1: Two particles obeying first law of Newtons dynamics, purple one remains at rest (no velocity and no acceleration) green one having constant velocity (velocity constance can be concluded from constant distances between particle trace)

2. In an inertial reference frame, the vector sum of the forces F on an object is equal to the mass m of that object multiplied by the acceleration a of the object: $F = ma$

3. When one body exerts a force on a second body, the second body simultaneously exerts a force equal in magnitude and opposite in direction on the first body.

Another important principle that impacts simulation course, is product of mass and velocity called Momentum. Its crucial property is that the sum of particles momenta in a closed system is conserved.

Last but not least important property of motion that is being used in case of multi body collision is kinetic energy.

2.1 Elastic collisions

Elastic collisions are when total kinetic energy before and after collision is conserved, as opposed to inelastic collisions where kinetic energy is lost (converted into heat). Elastic collisions don't usually happen in macro-scale because of internal friction, only the collisions of atoms can fully be considered as such. Collisions of molecules, such as in gas and air, aren't usually perfectly elastic, but because half of them are inelastic, and half super-elastic (increasing kinetic energy after the collision), across the entire system we can approximate them to be elastic.

2.1.1 One-dimensional collisions

The simplest form of a collision is one occurring in one dimension. The conservation of the total momentum demands that the total momentum before the collision is the same as the total momentum after the collision. The equation describing this:

$$m_1 v_1' + m_2 v_2' = m_1 v_1 + m_2 v_2$$

where:

- v_1' - vector of speed of body 1 after collision
- v_2' - vector of speed of body 2 after collision
- v_1 - vector of speed of body 1
- v_2 - vector of speed of body 2
- m_1 - mass of body 1
- m_2 - mass of body 2

Solving for v_1' and v_2':

$$v_1' = \frac{v_1(m_1 - m_2) + 2m_2 u_2}{m_1 + m_2} \qquad v_2' = \frac{v_2(m_2 - m_1) + 2m_1 u_1}{m_1 + m_2}$$

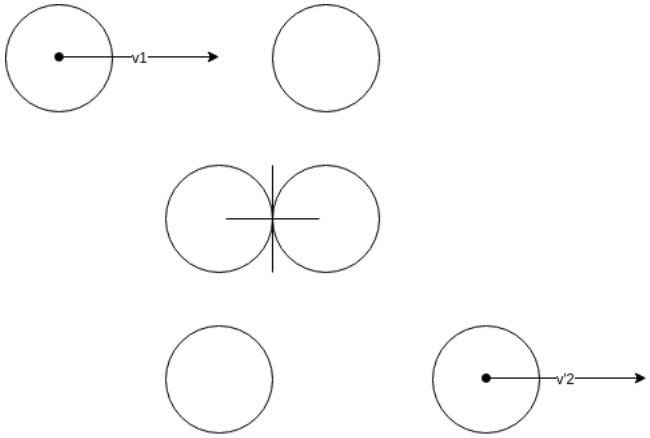


Figure 2.2: Theoretical collision of two particles in 2d, one moving

In this simple case, $v_2' = v_1$, because $v_2 = 0$ and both particles have the same mass.

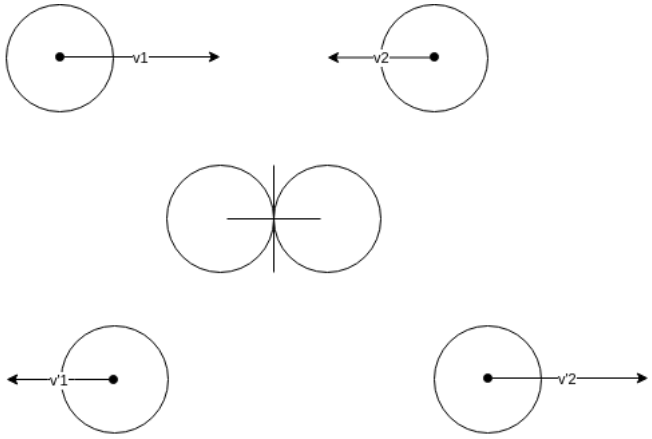


Figure 2.3: Theoretical collision of two particles in 2d, both moving

Here, $v_2' = v_1$ and $v_1' = v_2$

2.1.2 Two-dimensional collisions

The more interesting, and more complicated case, is when collisions occur in two dimensions.
Bodies' behaviors are described using the following equations:

$$v_1' = v_1 - \frac{2m_2}{m_1+m_2} \frac{\langle v_1 - v_2, x_1 - x_2 \rangle}{||x_1 - x_2||^2} (x_1 - x_2)$$

$$v_2' = v_2 - \frac{2m_1}{m_2+m_1} \frac{\langle v_2 - v_1, x_2 - x_1 \rangle}{||x_2 - x_1||^2} (x_2 - x_1)$$

where:

- v_1' - vector of speed of body 1 after collision
- v_2' - vector of speed of body 2 after collision

- v_1 - vector of speed of body 1
- v_2 - vector of speed of body 2
- x_1 - vector of position of body 1
- x_1 - vector of position of body 2
- m_1 - mass of body 1 (scalar)
- m_2 - mass of body 2 (scalar)

Here it is visualized, for one stationary body ($v_2 = 0$):

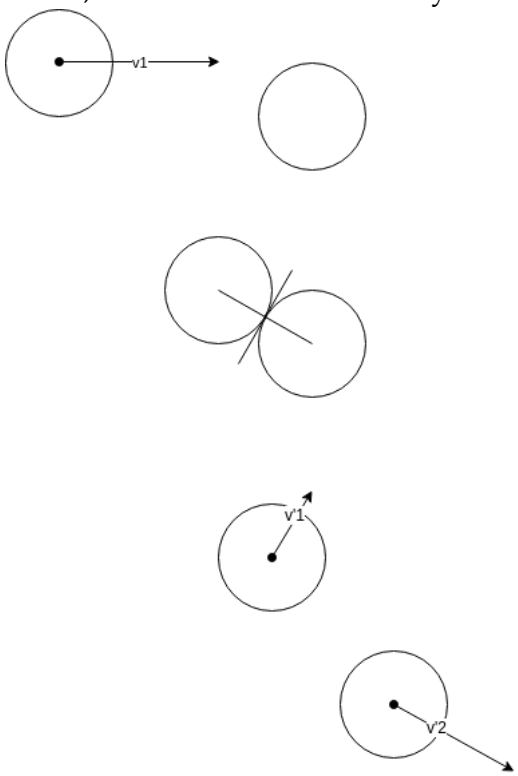


Figure 2.4: Theoretical collision of two particles in 2d

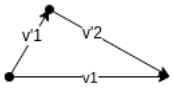


Figure 2.5: Resulting vectors add up to the original vector v_1 (the law of conservation of total momentum)

And also visualized in the simulating engine:

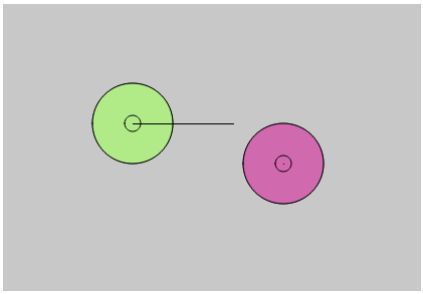


Figure 2.6: Simulated collision, before the hit.

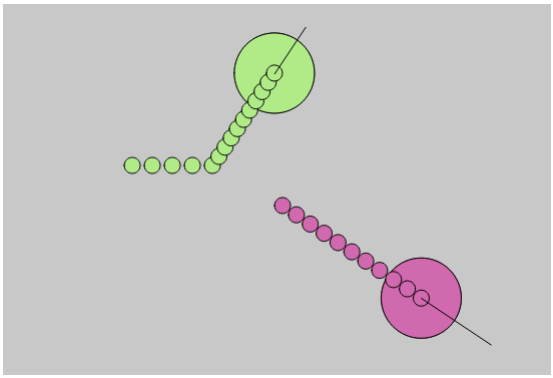


Figure 2.7: Simulated collision, after the hit.

2.2 Simulation description

The simulation consists of populating our simulation space with randomly placed particles. Particles' masses and speeds are randomized, within user-defined limits set in simulation parameters. Particles can't exit the system, they bounce elastically off of the walls (boundaries).
All Particles are assumed to be equally dense, so if they differ in mass, they also differ in size, which makes it easier to distinguish heavy ones from the lighter.

2.2.1 Sandbox mode

For modeling specific situations, for example unmixed gas/liquid, or precise systems like Newton's Cradle, it's possible to populate the simulation with completely user-defined particles. It only determines initial state, the rest works exactly the same as in the normal case.

3 Technical overview

3.1 Technologies

To make such simulation possible, easy to use and distribute - web application approach has been used. Whole simulation engine and presentation layer is written in `JavaScript`, supported by additional libraries like `p5.js` or `bootstrap`.
This approach guarantees ease of end user use (no need to install additional software, besides web browser) and repeatability of results (web browser environment is pretty well standardized and its behavior is mostly the same on every platform).

3.2 Source Code

(a) https://github.com **(b)** https://cloud.google.com/

Figure 3.8: Used hosting providers

Its source code is being kept open source within git version control system repository at GitHub:

`https://github.com/evemorgen/physics_assignment`

along with readme containing usage and developers guide to this project.

Because, most of this project components are web-based, natural consequence would be to host it within public Internet. For that purpose, free trial of Google Cloud Platform has been chosen, and is available under link provided below:

`https://goo.gl/keWpja`

3.3 Libraries support

Main simulation engine runs in web browser in JavaScript runtime, however, plain JavaScript does not provide convenient tools to operate on vectors, etc. This is where `p5.js` becomes handy.
p5.js is JavaScript library under active development equipped with lots of useful features including:

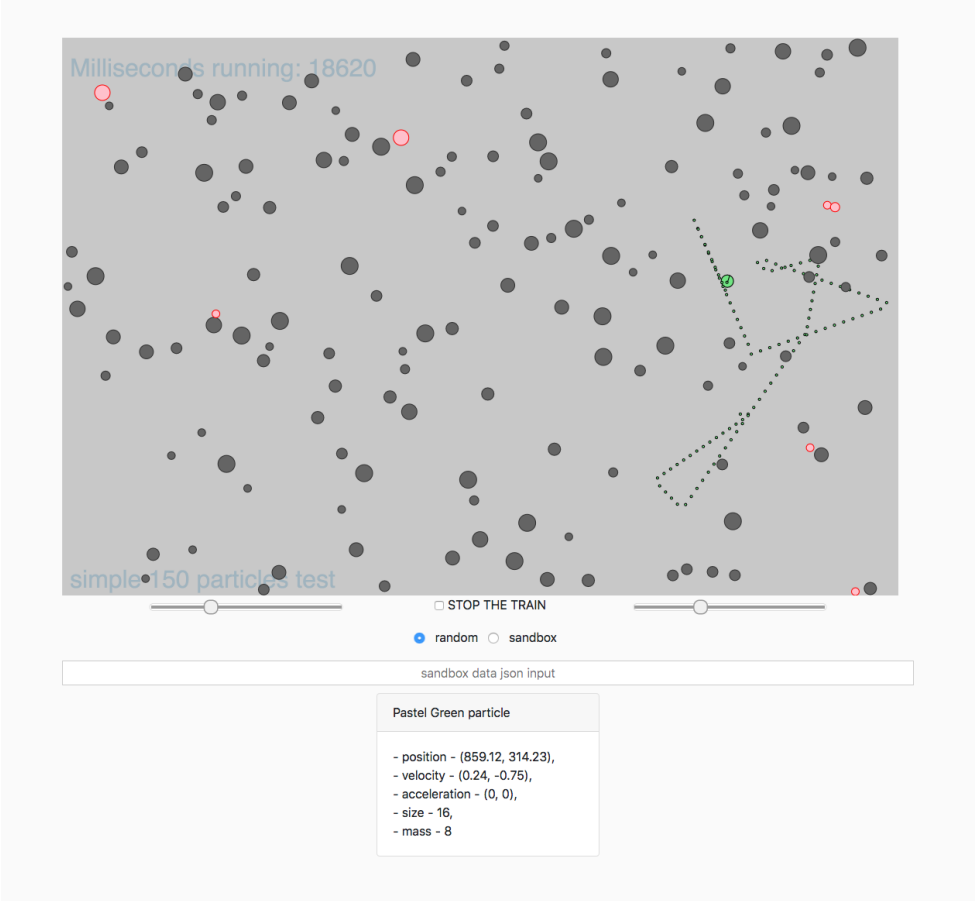
(a) `https://p5js.org/` **(b)** `https://getbootstrap.com`

Figure 3.9: Used supporting libraries

- vectors arithmetic
- browser event handling (like mouse click, etc.)
- easy 2D and 3D shapes drawing
- DOM manipulation helpers

and many many more. But that only covers drawing and calculating simulation state case.
For overall look and feel, more CSS based solution is needed. That is why Bootstrap is being used to give style everything besides simulation canvas, i.e. input form for sandbox, cards with particles data and so on.

3.4 GUI description

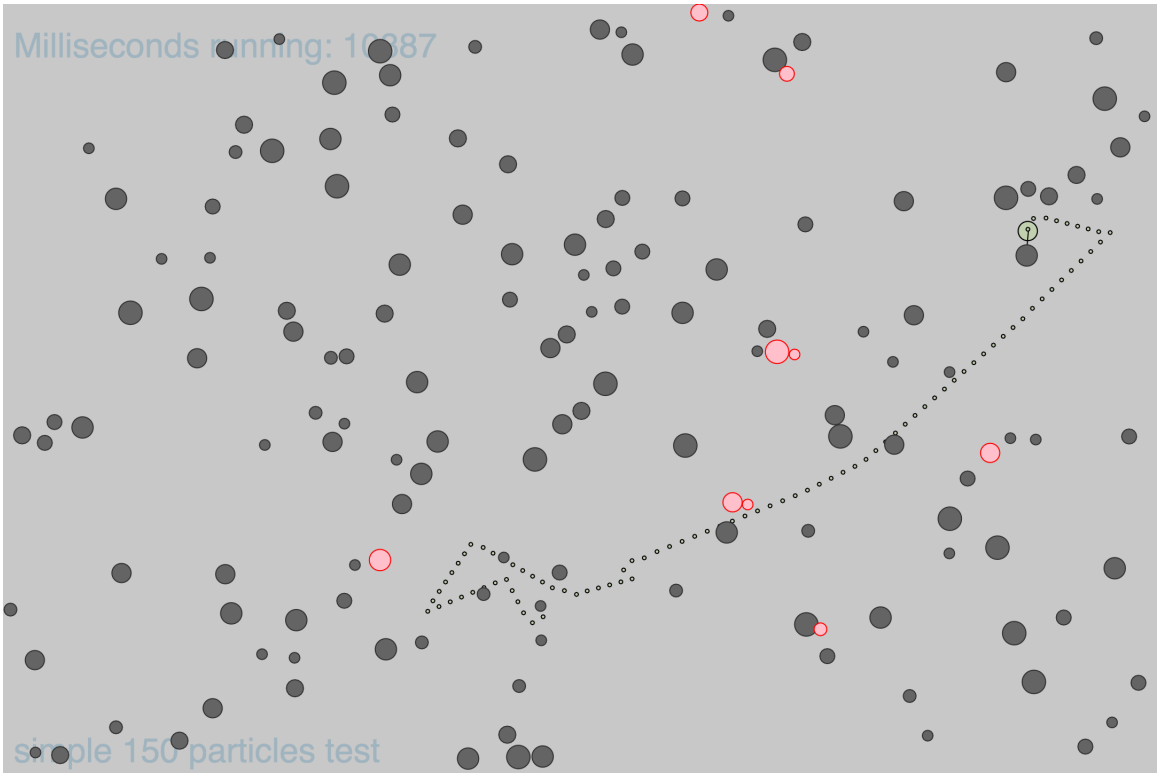


Simulation graphical user interface consist 3 major elements:

1. Canvas with simulation results, particles represented as circles, total number of particles in bottom left corner and simulation time in upper left corner;
2. Control elements right under canvas, allowing user to drive simulation parameters. The controllable elements are:
   - two sliders defining: number of particles participating in simulation (one on the left) and initial speed of particles (one on the right);
   - check box to pause simulation (between sliders);
   - two select buttons indicating weather simulation is in random or sandbox mode;
   - text input for json data which is being used in sandbox mode.
3. So called "particle cards" providing information about current particle state (only the one that were checked) like its position, velocity, acceleration, mass and so on.

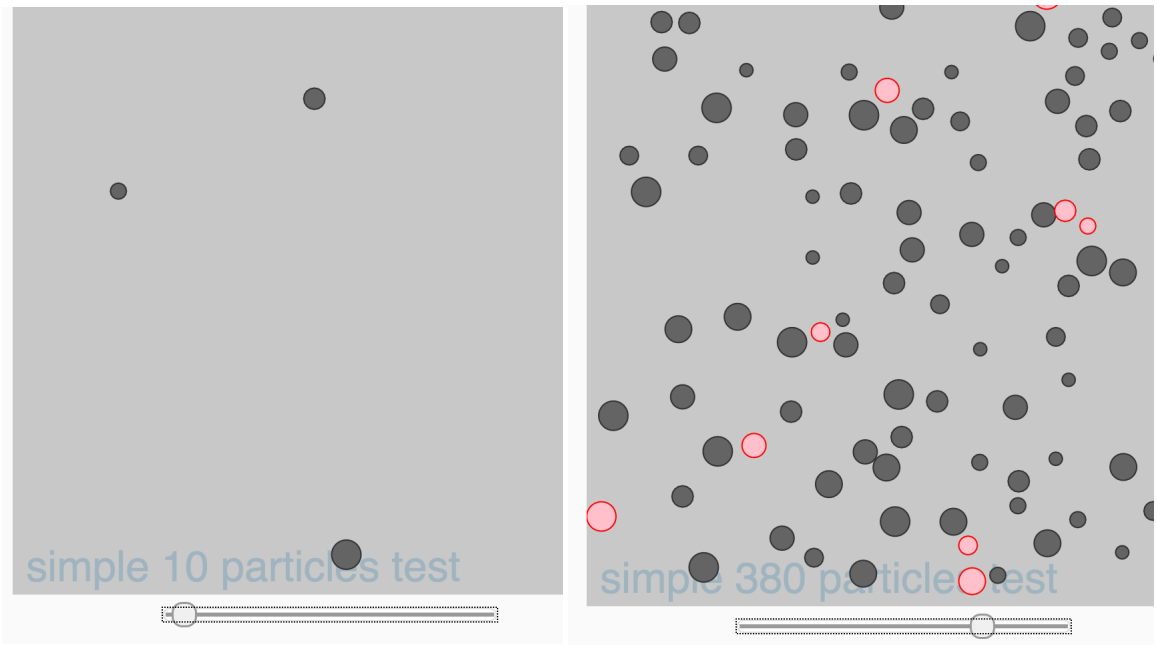
3.4.1 Canvas

As mentioned before, canvas shows current simulation state, each particle is represented by the circle, which size indicates its mass (the bigger the circle, the more heavy it is, constant density is assumed). All of them has place on 2d space, where they can bounce off each other.
When particle interact with one another, they light up in pink-ish colour for 3 animation frames to express such situation.
Last but not least, user can set any particle to tracking state, that means, when one clicks on particle, it alters its colour to different then environment, it leaves trace after its path (smaller circles in the same colour) and shows vectors indicating its velocity and forces applied to it.



3.4.2 Controls

As mentioned before, there are couple of settings, that end user can adjust, like number of particle at simulation start or initial velocity of particles.



(a) Simulation starting with 10 particles

(b) Simulation starting with 380 particles

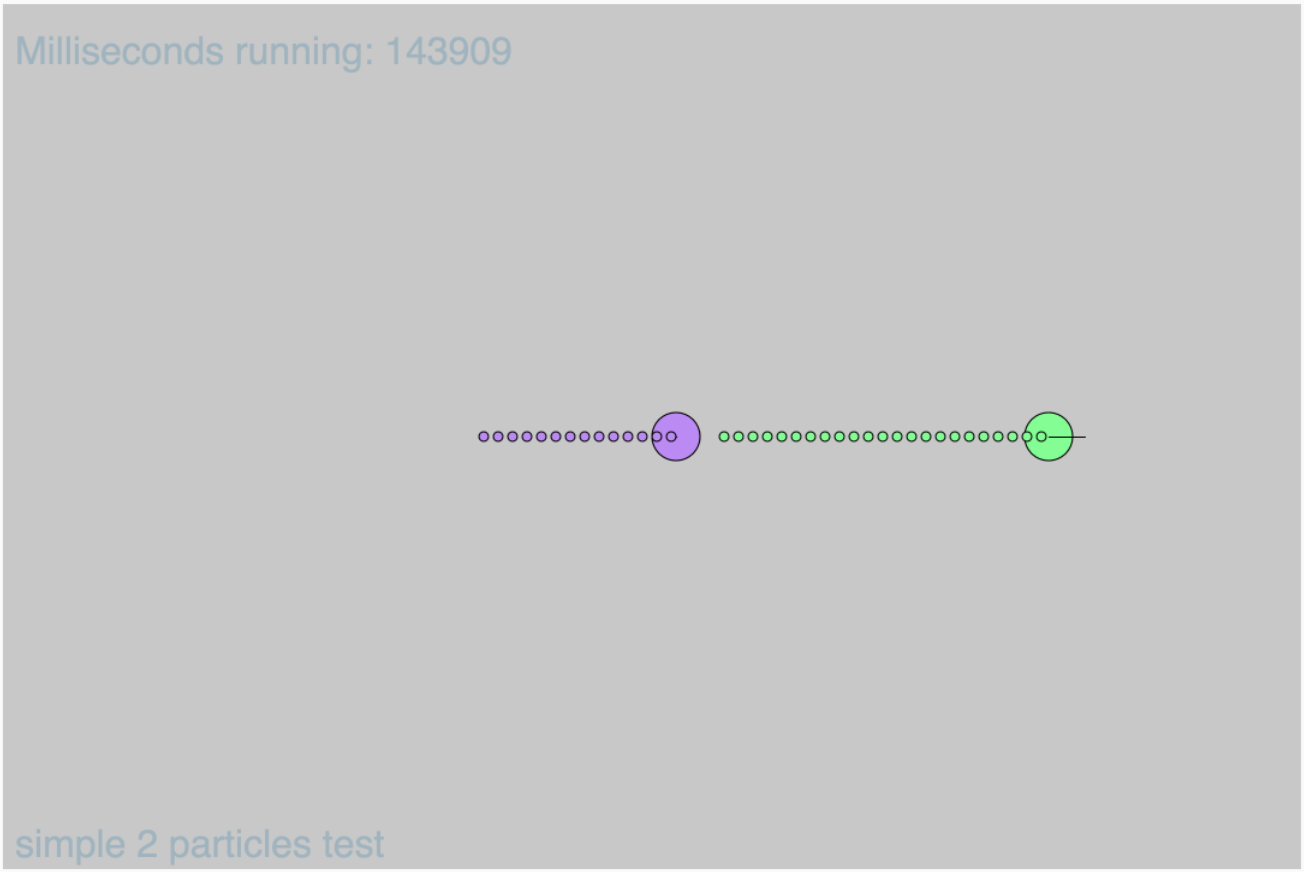
Figure 3.10: Particles slider values comparison

What is more, instead of random situation, one can specify their own with sandbox mode. To achieve that, initial particles state in json format is necessary. For example, two particles, at specified point, with initial velocity and no acceleration but with defined mass can be represented as following json array:

```
[
  {"x": 500, "y": 360, "ax": 0, "ay": 0, "vx": 5, "vy": 0, "m": 20},
  {"x": 700, "y": 360, "ax": 0, "ay": 0, "vx": 0, "vy": 0, "m": 20}
]
```

Listing 1: Two particles system json object example

(For more examples please see `data` directory)
which results in such situation after pasting mentioned json array to text input and switching to sandbox mode:



STOP THE TRAIN

random sandbox

[{"x": 400, "y": 360, "ax": 0, "ay": 0, "vx": 3, "vy": 0, "m": 20}, {"x": 600, "y": 360, "ax": 0, "ay": 0, "vx": 0, "vy": 0, "m": 20}]

Perfume particle

- position - (560.00, 360.00),
- velocity - (0.00, 0.00),
- acceleration - (0, 0),
- size - 40,
- mass - 20

Mint Green particle

- position - (870.00, 360.00),
- velocity - (3.00, 0.00),
- acceleration - (0, 0),
- size - 40,
- mass - 20

3.4.3 Particles Cards

Last thing to point out are so called “Particles Cards”. They enable end user to determine current particle parameters. Each card is named after particle colour name, which helps to identify which card relates to which particle. As mentioned before, such card provide all useful particle properties such as position, velocity, acceleration, its mass and size.

4 Conducted experiments

Since developed simulation engine is very flexible as well as capable, authors of this report decided to provide some examples that shows this project potential. Four main experiments were provided:

1. Newton's cradle
2. Random walk analysis
3. Diffusion presentation
4. Simplified semipermeable membrane simulation

4.1 Newton's cradle

Original Newton's cradle is a physical device showing conservation of momentum and energy with swinging balls (with identical mass and size). One ball at the end is lifted and released, it bumps into stationary ball, and through all stationary balls transmits force up to the last ball, which is being pushed out. When it returns to its initial position, bumping stationary ball, cycle starts again. With provided simulation engine it is fairly easy to reproduce this toy. To achieve that, one could create such initial setup:

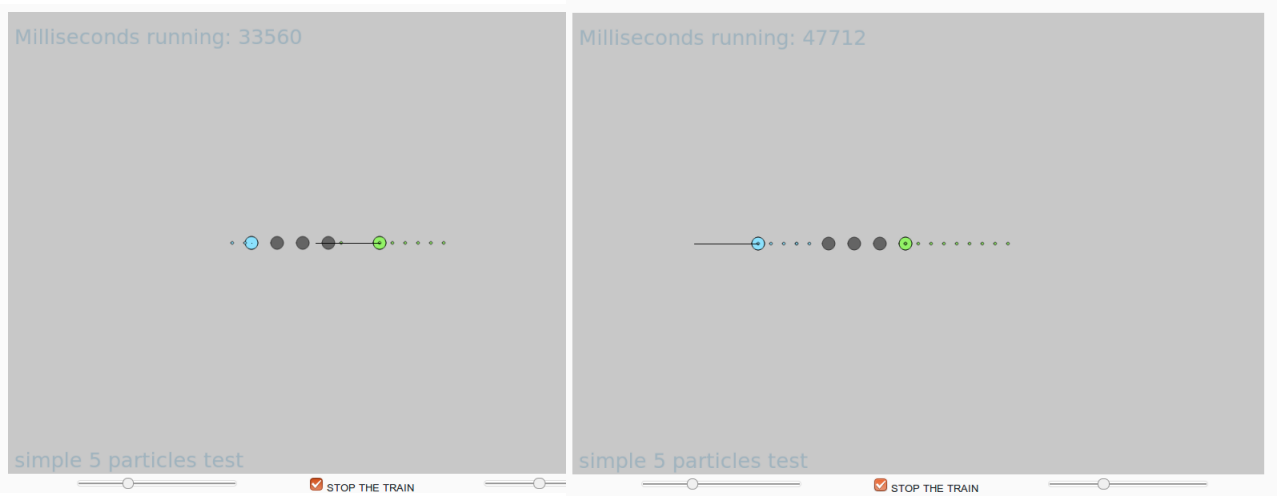
```
[
  {"x": 350, "y": 360, "ax": 0, "ay": 0, "vx": 10, "vy": 0, "m": 10},
  {"x": 400, "y": 360, "ax": 0, "ay": 0, "vx": 0, "vy": 0, "m": 10},
  {"x": 440, "y": 360, "ax": 0, "ay": 0, "vx": 0, "vy": 0, "m": 10},
  {"x": 480, "y": 360, "ax": 0, "ay": 0, "vx": 0, "vy": 0, "m": 10},
  {"x": 520, "y": 360, "ax": 0, "ay": 0, "vx": 0, "vy": 0, "m": 10}
]
```

Listing 2: Newton's cradle json representation

This json creates 5 particles with same mass (m: 10 units), no initial velocity and acceleration, except for one particle (velocity in x axis = 10). Also, they are all perfectly lined up. Since particles can bump into each other, and bounce off the walls, hence this whole setup results in similar effect as described before.
Still image cannot fully describe this phenomena, but is available on figure 4.11, which is why authors encourage to try this yourself with this simulation engine.

4.2 Random walk analysis

The system of multiple particles colliding elastically can be an approximation of particles in a gas, or a liquid. Because of that, by following a single particle, it's possible to model Brownian motion, and see why it looks like it's random.
First of all, to most accurately represent a particle suspended in gas/liquid, there need to be a lot of particles. Also, to make simulation faster, size of the enclosing



(a) Newton's cradle before collision

(b) Newton's cradle after collision

Figure 4.11: Newton's cradle simulation example

box has been shrunk. Now, using a trail, path of a single particle can be checked, to see the type of motion that undergoes in such system. It looks very similar to a particle path that would be generated randomly.

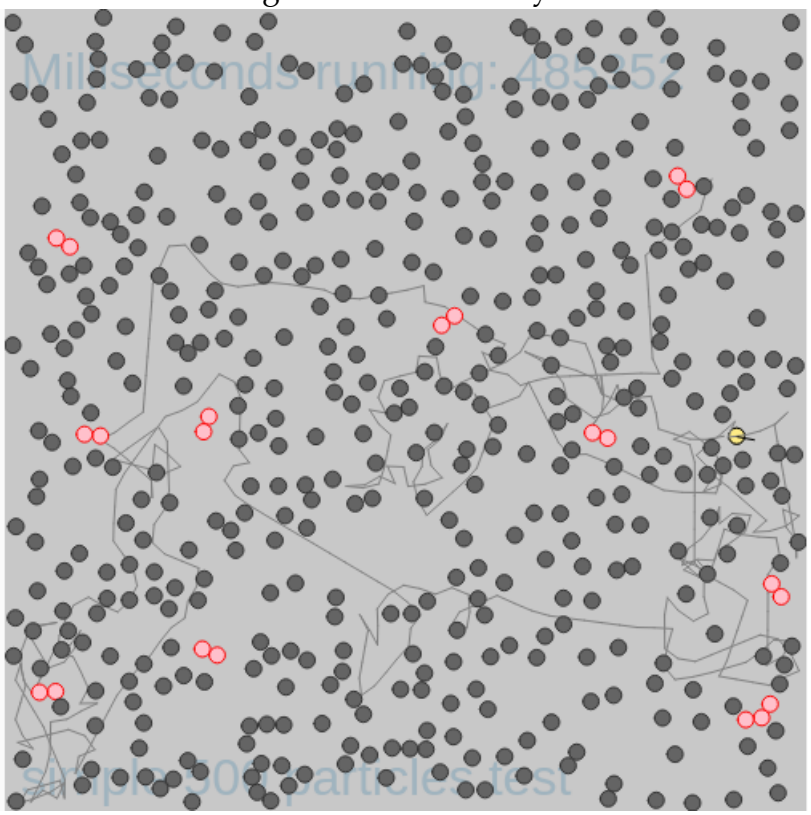


Figure 4.12: Brownian motion of a tracked particle (yellow)

For particles that move randomly, what should be observed is that their mean squared displacement grows linearly in time. Mean squared displacement is defined as: $\frac{1}{N}\sum_{N}^{n=1}(x_n(t) - x_n(0)))^2$
where:

- N - number of particles,
- $x_n(0)$ - reference (initial) position of each particle,
- $x_n(t)$ - particle position in time t,

The simulation engine allows for drawing a graph of mean squared displacement for particles in time, as well as calculating the linear regression for said graph.

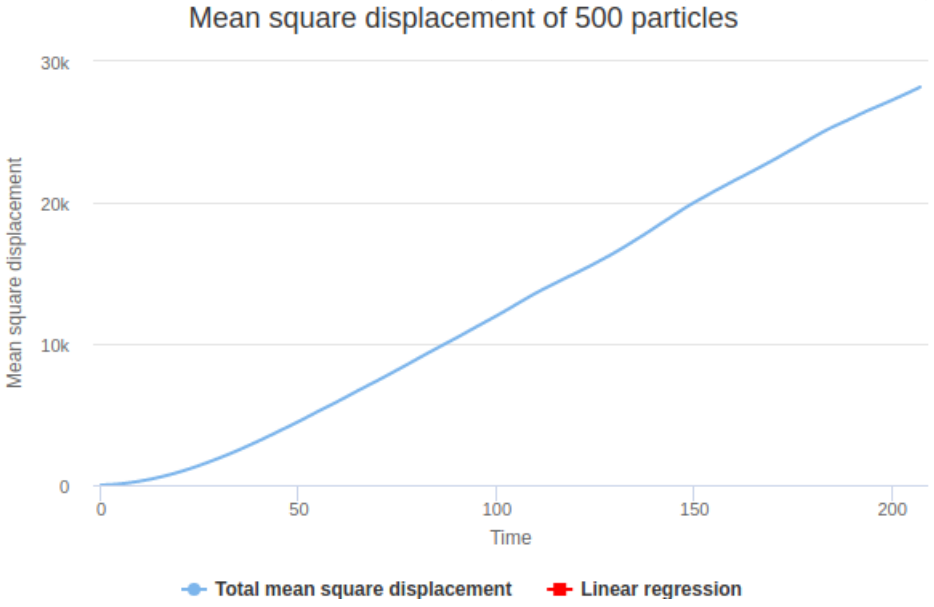


Figure 4.13: Mean squared displacement, 200 frames

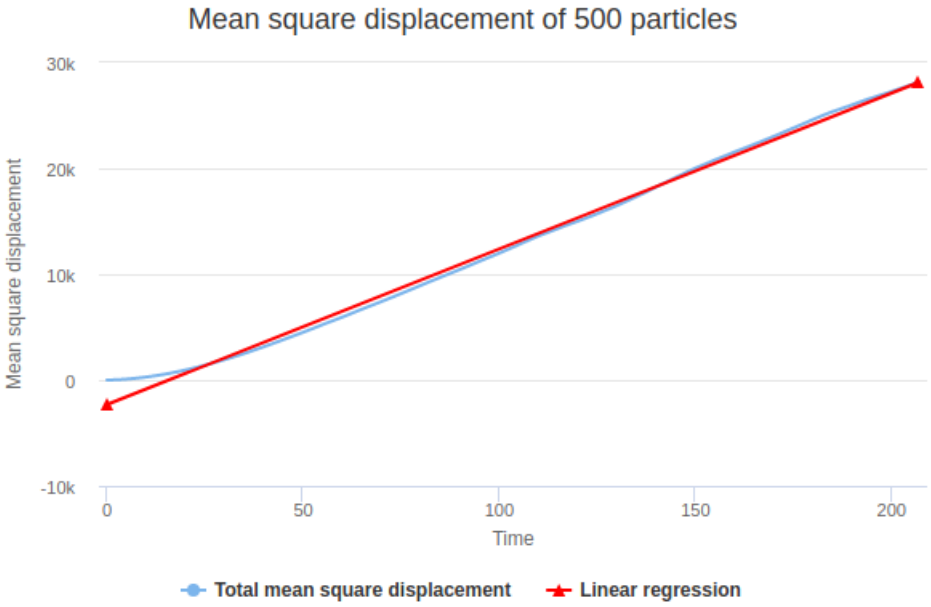


Figure 4.14: Mean squared displacement with linear regression

The graph shows exactly the result that was expected: linear relationship between time and total MSD. The problem comes when the simulation runs for a longer period of time.

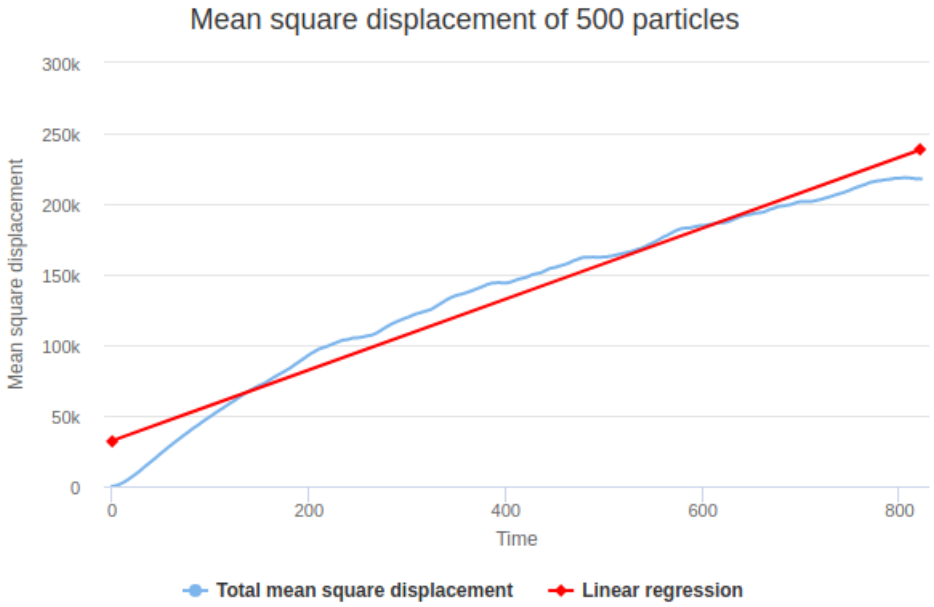


Figure 4.15: Mean squared displacement, 800 frames, non-linear

MSD stops being linear, its growth slows down. The most probable reason for that is that all the particles have to exist in confined space, there's a limit to how far they can go. It's the limitation of this way of modeling Brownian motion.

Although still, for shorter periods of time its a useful simulation for random walk.

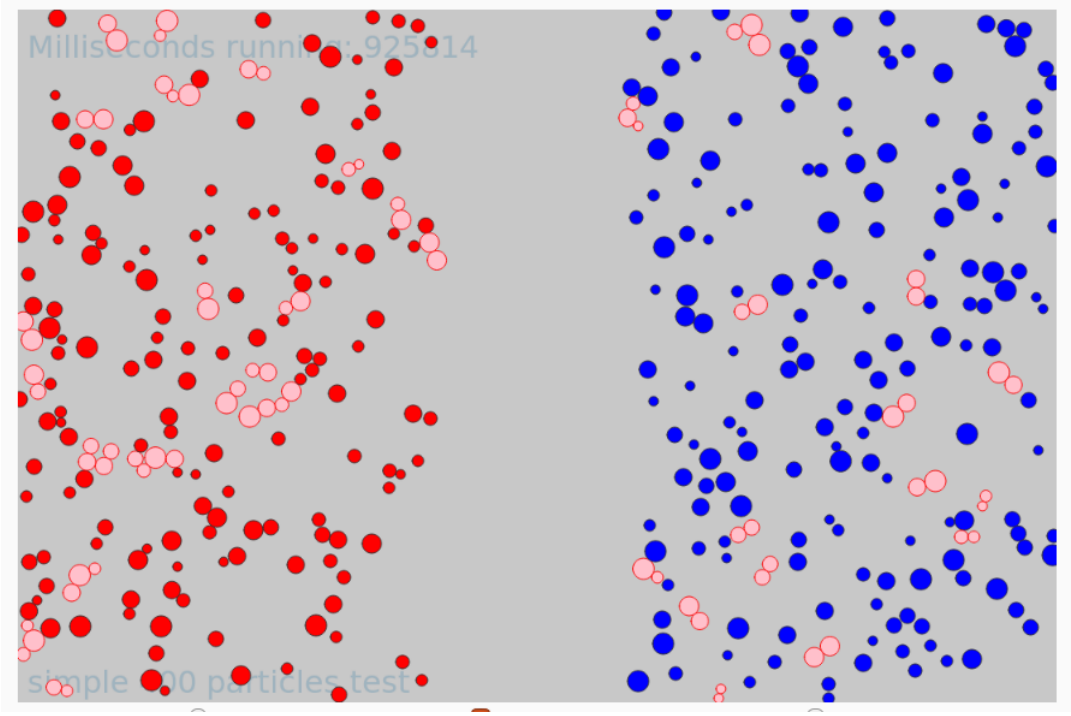
4.3 Diffusion

The phenomena called diffusion can be another good example, because it highly depends on so called ”random walk”, and results in mixing distinguishable particles, resulting in changes of particle concentration over distance - which is called concentration gradient.
To simulate diffusion effect, one would need to generate some particles. The idea is to provide 2 groups of distinguishable with random initial mass, size, velocity. Each group located at the opposite side of the simulation area. This setup could be obtained by usage of the following script

```
import json
import sys
from random import uniform

# generate particles on a left side of simulation area
left_particles = [
     # pick random x position from beginning to more less half
  {"x": int(uniform(0, 1080 / 2 - 100)),
     # pick random y position
     "y": int(uniform(0, 720)),
     # particle will be red
     "c": "#ff0000",
     "dc": False,
     # with 0 acceleration
     "ax": 0,                # generate 200 particles
     "ay": 0} for _ in range(0, 200)
]

right_particles = [{"x": int(uniform(1080 / 2 + 100, 1080)), "y": int(uniform(0,
    720)), "c": "#0000ff", "dc": False, "ax": 0, "ay": 0} for _ in range(0, 200)]

# print result as json
print(json.dumps(left_particles + right_particles))
```

Listing 3: Diffusion data generator

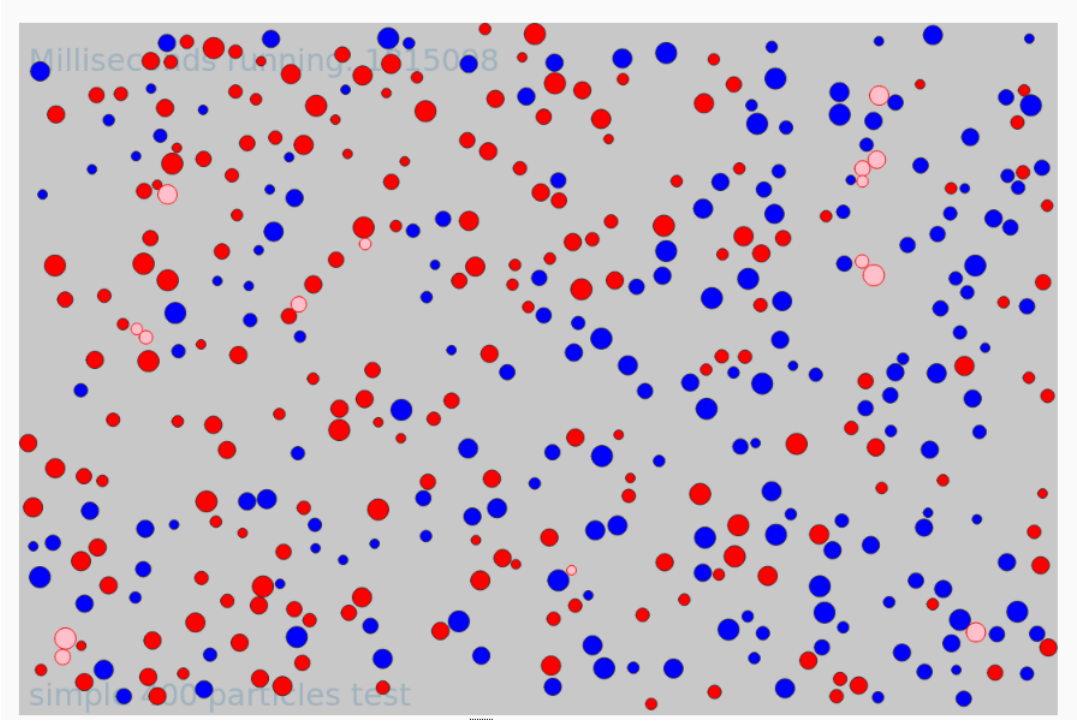
After generating input for this kind of simulation, in its initial state it looks like this:



One of proposed methods to measure concentration gradient could be to calculate mean color for image fragments. Image has been spit into two sides, left and right with split point right in the middle. The following results were obtained:

- left side - rgb(32, 12, 12), hsl(0, 45%, 9%)
- right side - rgb(8, 6, 28), hsl(245, 65%, 7%)

Which seems reasonable, that left side tends more to the red, and right side tends more to the blue.
After couple of minutes of simulation, one could see the following image:



Performing same operation as before, achieved results were as follows:

- left side - rgb(23, 7, 15), hsl(330, 53%, 6%)
- right side - rgb(12, 4, 20), hsl(270, 67%, 5%)

It is clearly visible that mean color of these two sides changed, both sides shifted towards one another initial colors. If one would wait some more time, results would be even more clear.

4.4 Semipermeable membrane

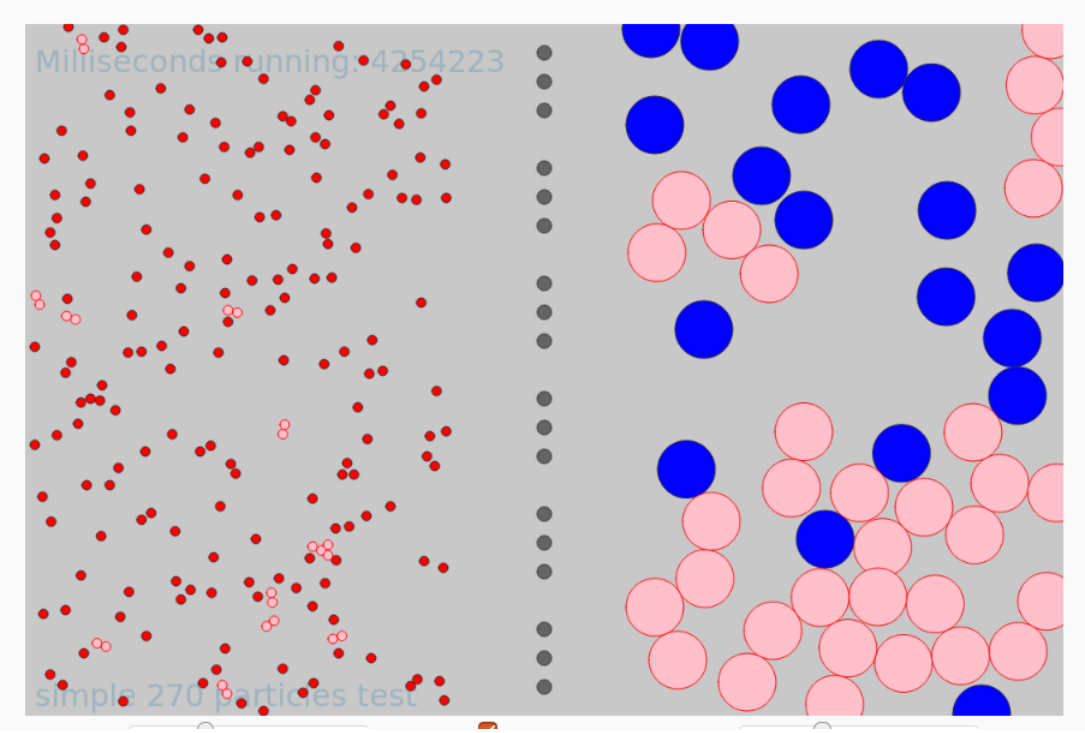
Last chosen experiment was to simulate simplified version of semipermeable membrane. Semipermeable membrane is type of biological membrane that allow certain molecules to pass through it (by size) by diffusion (this experiment assumes passive transport for simplicity sake).
To simulate this particular effect with this engine, the following example json can be used

```
[
  # particles that imitates membrane - high mass should prevent particle from
      moving
  # x, y coordinates,
  # m, s stands for mass and size
  # vx, vy, ax, ay stands for velocity and acceleration in x,y axis
  {'x': 500, 'm': 999999999, 'y': 740, 's': 15, 'vx': 0, 'ay': 0, 'vy': 0, 'ax':
      0},
```

```
  {'x': 500, 'm': 999999999, 'y': 750, 's': 15, 'vx': 0, 'ay': 0, 'vy': 0, 'ax': 0},
  ...
  # small particles on left side of membrane
  # vx and vy are not defined, which result to random value
  {"ay": 0, "c": "#ff0000", "ax": 0, "y": 645, "x": 76, "s": 10, "m": 10, "dc": false},
  {"ay": 0, "c": "#ff0000", "ax": 0, "y": 686, "x": 185, "s": 10, "m": 10, "dc": false},
  ...
  # bigger particles on the right side of membrane
  {"ay": 0, "c": "#0000ff", "ax": 0, "y": 5, "x": 651, "s": 60, "m": 60, "dc": false},
  {"ay": 0, "c": "#0000ff", "ax": 0, "y": 597, "x": 827, "s": 60, "m": 60, "dc": false}
]
```

Listing 4: Diffusion data generator

After applying this model, subsequent result would be gathered:



Review of this image proves that, with very little effort one could obtain interesting outcome. This model can be used for example to calculate minimal time needed to migrate small particles from one side of a membrane to another including bumpbing back from bigger particles from another side.

5 Conclusion

This report provided description of developed simulation engine for multi body collisions. It covered basic concepts behind its development (used technologies and programming approach) as well as basic physical laws and principles utilized. It proves how powerful simplicity of physics laws is and how well it describes so many things around us, from macroscopic to molecular size. In favor of this opinion, 4 different experiments were conducted, each highlighting how easy to use and powerful at the same time this simulation engine is, despite its simple principles. To conclude, authors of this reports are pleased with received results.