AGH, WEAIiB	**Advanced Pyhon Programming**	Date **18.01.2019**
Systems modelling and Data analysis	Project topic: Multibody collision	
	Authors: **Michał Morawiec, Patryk Jankowski, Pablo Castaño, Alperen Kara**	

1. **Aim:**

The goal of our project was to prepare physical model presenting collision between few objects. Task has been divided into two problems, physical calculations and form of collisions presentation.

2. **Theoretical introduction:**

2.1 Elastic collision:

An elastic collision is a collision in which kinetic energy is conserved. That means no energy is lost as heat or sound during the collision. In the real world, there are no perfectly elastic collisions on an everyday scale of size. In an elastic collision, both kinetic energy and momentum are conserved (the total before and after the collision remains the same).

Conservation of momentum: $m_1 v_1 + m_2 v_2 = m_1 v'_2 + m_2 v'_2$

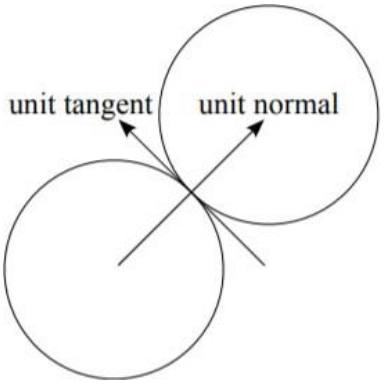
Conservation of kinetic energy: $\frac{1}{2} m_1 v_1^2 + \frac{1}{2} m_2 v_2^2 = \frac{1}{2} m_1 v'^2_1 + \frac{1}{2} m_2 v'^2_2$

During collision implementation we have facing with two kinds of possible collisions: collisions between two circles and collision between circle and border. Each methodology will be briefly descripted below:

2.2 Collision between two circles:

To simplify model, we have decided to treat out phenomena as simple elastic collision in two dimensions. We assumed that in our model we will have only one specific types of body – circle. Each body have velocity, mass, radius and coordinates. Used algorithm has been divided into 6 stages.

In stage one our goal was to find normal and unit tangent vector. The unit normal vector is a vector which has a magnitude of 1 and a direction that is normal to the surfaces of the objects at the point of collision Unit tangent vector is vector with a magnitude of 1 which is tangent to the circles' surfaces at the point of collision



First, we must find unit normal vector which is calculated with following formula:

$$\overrightarrow{un} = \frac{\vec{n}}{\sqrt{n_{x+}^2 n_y^2}}, \text{ where } \vec{n} = \langle x_2 - x_1, y_2 - y_1 \rangle$$

When we have obtained normal vector, we can easily calculate unit tangent vector:

$$\overrightarrow{ut} = \langle -un_y, un_x \rangle$$

Stage two was based on resolving of velocity vectors for each body into both components – tangential and normal, to perform these computations we must calculate dot product, it is calculated according to following formulas:

$$v_{1n} = \overrightarrow{un} \circ \overrightarrow{v_1} \ , \ \ v_{1t} = \overrightarrow{ut} \circ \overrightarrow{v_1} \ , v_{2n} = \overrightarrow{un} \circ \overrightarrow{v_2} \ , \ v_{2t} = \overrightarrow{ut} \circ \overrightarrow{v_2}$$

In stage three we have computed new tangential velocities. The tangential part of velocity has not changed during collision because there is no force between the circles in the tangential direction. So, we will simply assign values before collision to values after collision.

$$v'_{1t} = v_{1t} \ , \ v'_{2t} = v_{2t}$$

Stage four is most complicated part, in this step we have to combine conservation of kinetic energy and conservation of momentum in order to obtain new normal velocities. After performing few algebraic operations, we have obtained proper formulas (Important thing to note is fact that in this step we treat collision as typical one – dimensional collision)

$$v'_{1n} = \frac{v_{1n}(m_1 - m_2) + 2(m_2 - v_{2n})}{m_1 + m_2}, \ v'_{2n} = \frac{v_{2n}(m_2 - m_1) + 2(m_1 - v_{1n})}{m_1 + m_2}$$

Now we have to convert scalar normal and tangential velocities into vector. To do that we have to perform following computations:

$$\overrightarrow{v'_{1n}} = v'_{1n} \cdot \overrightarrow{un}, \qquad \overrightarrow{v'_{1t}} = v'_{1t} \cdot \overrightarrow{ut}, \qquad \overrightarrow{v'_{2n}} = v'_{2n} \cdot \overrightarrow{un}, \qquad \overrightarrow{v'_{2t}} = v'_{2t} \cdot \overrightarrow{ut}$$

In the last stage we want to obtain final velocity vectors, there is no complicated computations, in this order it is enough to add normal and tangential vector for each body:

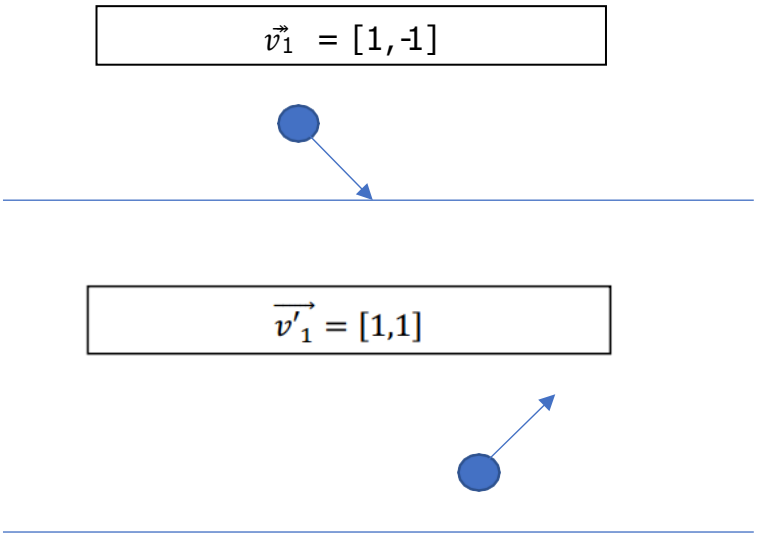
$$\overrightarrow{v'_1} = \overrightarrow{v'_{1n}} + \overrightarrow{v'_{1t}}, \qquad \overrightarrow{v'_2} = \overrightarrow{v'_{2n}} + \overrightarrow{v'_{2t}}$$

2.3 Collision between border and circles:

Second implemented type of collision is collision between border and circle, same as in previous example we have considered only elastic collision. This model is significantly simpler than model above, we don't perform any complicated computations. It is sufficient to multiply proper coordinate by -1. In this collision model we distinct collisions with two kinds of border: vertical and horizontal. Depending on border type our computations are a little bit different:

Collision with horizontal border:

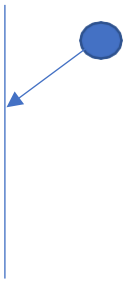
$$\vec{v}' = [v_x, -1 \cdot v_y]$$



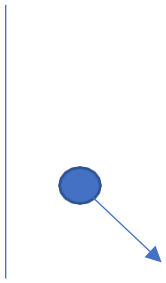
Collision with vertical border:

$$\vec{v}\,' = [-1 \cdot v_x,\ v_y]$$

$$\vec{v_1} = [-1,1]$$



$$\overrightarrow{v'_1} = [1,1]$$



3. Documentation

3.1 Technologies

- Python 3.6
  - NumPy – to simplify calculation on vectors
  - websocket - server
- JavaScript (ECMAScript 2015)
  - webpack - to build and serve user interface
- WebSockets - for frontend-backend communication
- Docker with docker-compose

3.2 Project Assumption

- Fast backend - able to handle at least 60 frames per second
- Flexible physics implementation - easy to switch with more complex model
- User interface - with menu, so the project can be managed easily
- Web browser's window is treated as a world - the boundaries of the browser's window are the borders

3.3 Project Structure

- backend - backend server with world and collision logic
- frontend - user interface with presentation and management menu of multibody collision world

3.4 Backend

main.py

To analyse how the backend server works let's start with main.py. The file is self-explanatory. All it does is initialization of world object, which is then handled by server object. WebSockets server is started and is ready to handle incoming connections from frontend. Finally the infinite while world.running(): loop is started.
The world.running() method always returns True, but it is covered with simple and readable API. Last but not least, the world is processed - checked for collisions which are handled - and objects' positions are send to frontend (to any client connected to the server). time.sleep is used to limit number of frames per second to stable (non-oscillating) number.

main.py uses two classes src.WebSockets.Server and src.World.World.

3.5 Frontend

When user enters the http://localhost:8080 the index.js file is loaded by the browserThe connection to the WebSockets server is opened and maintained until the browser's tab is closed. The initial values are parsed and processed by the backend. As the result, the user can see one object bouncing on the screen. Using the Show/hide menu, user can modify the world:

- change the number of the objects
- select whether the objects should have equal mass or not
- pause or start or init the world again

3.6 Physics

Code responsible for physical computations were encapsulated in src.Physics directory. In this directory we can find two files:

- Collision.py – Include method which allow to execute proper Collision depending passed Tag which describe type of collision
- CollisionsStrategies.py – include implementation of particular type of collision

Next important directory in terms of physical computations is src.Controllers in this file is we encounter class CircleController.py which observe position of circles, and detect collisions.

4. Project Installation

- Install Docker – Follow detailed instruction available at this page: https://docs.docker.com/install/
- Install docker - compose – Follow detailed instruction available at this page: https://docs.docker.com/compose/install/
- Start project locally - To start project locally type docker-compose up -d --build (or use Docker for Windows/Mac). The process of creating and provisioning of the project can take up to few minutes

When the project has started you can visit http://localhost:8080 to see user interface.

5. Literature

- *2-Dimensional Elastic Collisions without Trigonometry*, 2009 Chad Berchek: http://www.vobarian.com/collisions/2dcollisions2.pdf
- *Elastic collision*, Wikipedia: https://en.wikipedia.org/wiki/Elastic_collision