Documentation
Simple Physic Library v1.1
Piotr Pietras (peterooo94)

I. General information

App has been created as test ground for development of simplePhysic library. SimplePhysic library allows to simulate behaviour of shapes affected by physics laws. Library is still under construction.

II. Scripts's element

simplePhysic.math

vector - dependency (none) - current ver (1.1)

Vector is a class in simplePhysic object that describes euclidean vector. Vector object is created by passing arrow position(x,y,z).
Class contains static methods:

-**Vector.substract()** - returns vector's substraction
-**Vector.sum()** - returns vector's sum
-**Vector.distance()** - returns distance between two vectors (scalar)
-**Vector.magnitude()** - returns vector's magnitude
-**Vector.unit()** - returns unit vector of given vector
-**Vector.cross()** - returns vector's cross product
-**Vector.dot()** - returns vector's dot product
-**Vector.subdivide()** - returns vector's subdivision to parrarel and normal to given unit vector
-**Vector.rotate()** - returns vector rotated around given point and angle
-**Vector.angle()** - returns angle between given vactors in rad
-**Vector.absolute()** - returns vector by changing given vector's position to absolute

-**Vector.assimilate()** - returns vector by changing vector's sign position to given one

line - dependency (simplePhysic.vector) - current ver (1.1)

Line is a class in *simplePhysic* object that describes line. *Line* is based on vector equetion of line. *Line* object is created by passing two *vector* which points begining and end of the line.
Class contains variables:

-v - it's *vector* which is describes direction of *line*

Class contains static methods:

-**Line.moveBy()** - returns line moved by given vector
-**Line.intersect2D()** - checks whether two given lines interesct each other
-**Line.intersectPoint2D()** - returns vector pointing point of two given line intersection
-**Line.distanceToPoint2D()** - returns vector normal to given line and conteining given point. Vector's arrow points given line.

physic - dependency (simplePhysic.vector) - current ver (1.1)

Physic is a module in simplePhysic object that contains function for impulse collision behaviour of elements. Impulse collision theory is based on wikipedia aricle (https://en.wikipedia.org/wiki/Collision_response).
Module contains functons:

-**collisionImpulse()** - returns elements' **impulse** based on given *collideVector* (point of collision) and *normalUnit* (normal vector to collision surface changed to unit).
Orginal equetion:

$$j_r = \frac{-(1+e)\mathbf{v}_r \cdot \hat{\mathbf{n}}}{m_1{}^{-1} + m_2{}^{-1} + (\mathbf{I}_1{}^{-1}(\mathbf{r}_1 \times \hat{\mathbf{n}}) \times \mathbf{r}_1 + \mathbf{I}_2{}^{-1}(\mathbf{r}_2 \times \hat{\mathbf{n}}) \times \mathbf{r}_2) \cdot \hat{\mathbf{n}}}$$

soruce: link above

-**linearImpulseAffect()** - affects choosen element by previously calculated **impulse**
Orginal equetion:

$$\mathbf{v'}_1 = \mathbf{v}_1 - \frac{j_r}{m_1}\hat{\mathbf{n}}$$

soruce: link above

-**angularImpulseAffect()** - affects choosen element by previously calculated **impulse**
Orginal equetion:

$$\omega'_1 = \omega_1 - j_r \mathbf{I}_1^{-1}(\mathbf{r}_1 \times \hat{\mathbf{n}})$$

soruce: link above

-**relativeVelocity()** - returns sum of angular and linear velocity relatively to given *relativeVector*
-**angularVelocityToLinear()** - converts angular velocity to linear one relatively to givem *relativeVector*

simplePhysic.geometry

geometry - ***dependency (simplePhysic.vector) - current ver (1.1)***

Geometry is a extendable class in *simplePhysic* object that describes geometry of HTML elements. *Geometry* object is created by passing it's width, height, position and color.
Class contains states:

-info.x - position x
-info.y - position y
-info.c - angular position

-info.width
-info.height
-info.color
-info.dragging - is currently being dragged

-physic.v - velocity vector *[px/s]*
-physic.w - angular velocity vector *[deg/s]*
-physic.density - element's density
-physic.mass - element's mass
-physic.inertia - element's moment of inertia
-physic.elastic - element's kinetic absortpion (1 mans no absorption occures)
Class contains methods:
-**addCSS()** - creates html element and append it to *simplePhysic.scene*
-**styleCSS()** - abstract method being overridden by inheriting class
-**removeCSS()** - removes html element
-**setPosition()** - sets current element's position
-**move()** - sets new postion of element based on given *velocityVector*
-**getPositionVector()** - returns base position of html element which is left top point od <div>. Point is static to any element's rotation.
-**getCenterVector()** - returns center of html element
-**setDragging()** - sets state whether element is currently being dragged

circle - *dependency (simplePhysic.geometry, simplePhysic.vector) - current ver (1.1)*

Circle is a extension class to *Geometry* that describes circle geometry of HTML elements. *Circle* object is created by passing it's width*(radius)*, position and color.
Class contains methods:
-**styleCSS()** - creates circle shaped html element

rectangle - *dependency (simplePhysic.geometry, simplePhysic.vector) - current ver (1.1)*

Rectangle is a extension class to *Geometry* that describes rectangle geometry of HTML elements. *Rectangle* object is created by passing it's width, height, position and color.
Class contains methods:
-**styleCSS()** - creates rectangle shaped html element
-**getPointVector()** - returns arrays of vectors which points rectangle edges. Counting edges is counter clockwise

simplePhysic.collision

Every collision calculation is divided into 3 parts:

- detect - which chcecks if collision occured. Every detect function also returns **collide** object that at least contains: *normalUnit* (normal vector to collision surface changed to unit), *collideVector* (point of collision), *distanceVector* (translation vector needed to remove elements' intersaction)
- affect - which applies collision impulse based on **collide** object returned by detect functon
- remove - which removes element's intersaction

detectBallFrameCollision() - *dependency (simplePhysic.vector, simplePhysic.circle) - current ver (1.1)*

DetectBallFrameCollision() is a function that checks whether *simplePhysic.circle* collides with *simplePhysic.scene* 's frame. Base condition is simple, distance between *getCenterVector()* and *simplePhysic.scene* frames has to be smaller then circle radius. Returns array of **collide** object {*normalUnit*, *distanceVector*, *collideVector*}

detectBallBallCollision() - *dependency (simplePhysic.vector, simplePhysic.circle) - current ver (1.1)*

DetectBallFrameCollision() is a function that checks whether simplePhysic.circle collides with simplePhysic.circle. Base condition is simple, distance between getCenterVector() has to be smaller than sum of *circle*'s radius. Returns **collide** object {distanceCenters (distance to *circle*'s centers), normalUnit1, normalUnit2, collideVector}

detectRectangleFrameCollision() - *dependency (simplePhysic.vector, simplePhysic.rectangle) - current ver (1.1)*

DetectRectangleFrameCollision() is a function that checks whether simplePhysic.rectangle collides with simplePhysic.scene 's frame. Base condition is simple, rectangles edges which are described by *getPointsVector()* cannot be beyond *simplePhysic.scene* frames. Returns array of **collide** object {*normalUnit*, *magnitude* (magnitude of *distanceVector*), *collideVector*}

detectRectangleRectangleCollision() - *dependency (simplePhysic.vector, simplePhysic.line, simplePhysic.rectangle) - current ver (1.1)*
(HIGHLY INEFFICIENT, needs to be replaced by faster algorithm)

DetectRectangleRectangleCollision() is a function that checks whether simplePhysic.rectangle collides with simplePhysic.rectangle. Base condition is simple, if one of the sides of rectangle which are described by *simplePhysic.line* intersact each other collision occures. However finding propriate **collide** object of this collision is more complicated.

1. Interates every combination of rectangle's sides. If intersaction is found puts collision information into array. Most importent is *distanceVector* which is *line.distanceToPoint2D()* vector created to be normal to static element and one edge of movable element and *normalUnit* of static element. One intersaction has 4 possible scenerios of collision.
2. Sorts array of collision's scenario by *distanceVector* magnitude. Smaller to higher.
3. Translate moveable element by *correctedDistanceVector* which is *vector.assimilate()* of *distanceVector* and *normalUnit*. If collision still occures takes another scenario into cosideration and reapet translation.

simplePhysic.simulation

Simulation is a module in simplePhysic object that contains one function called *simulate()*.

Module contains constant:

-REFRESH_PERIOD – it is time beetwen next calculation of simulation state

Module contains states:

-simulationInterval – contains object of current *setInterval()*
-checkObjectCollision – allows to check object's collision
-checkFrameCollision – allows to check object's collision with frame

Module contains function:

-**simulate()** - clears then attaches function of simulation process to *setInterval()*. Function of simulation process is a set of phases that leads to collision calculation, affect global effect or move object. Phases:

1. Add global effects (Gravity effect)
2. Check frame collision
3. Check object to object collision
4. Move object