Physics Engine

Technical Document

by Oleg Loshkin

Table of Contents: 1

Test_01: Vectors:

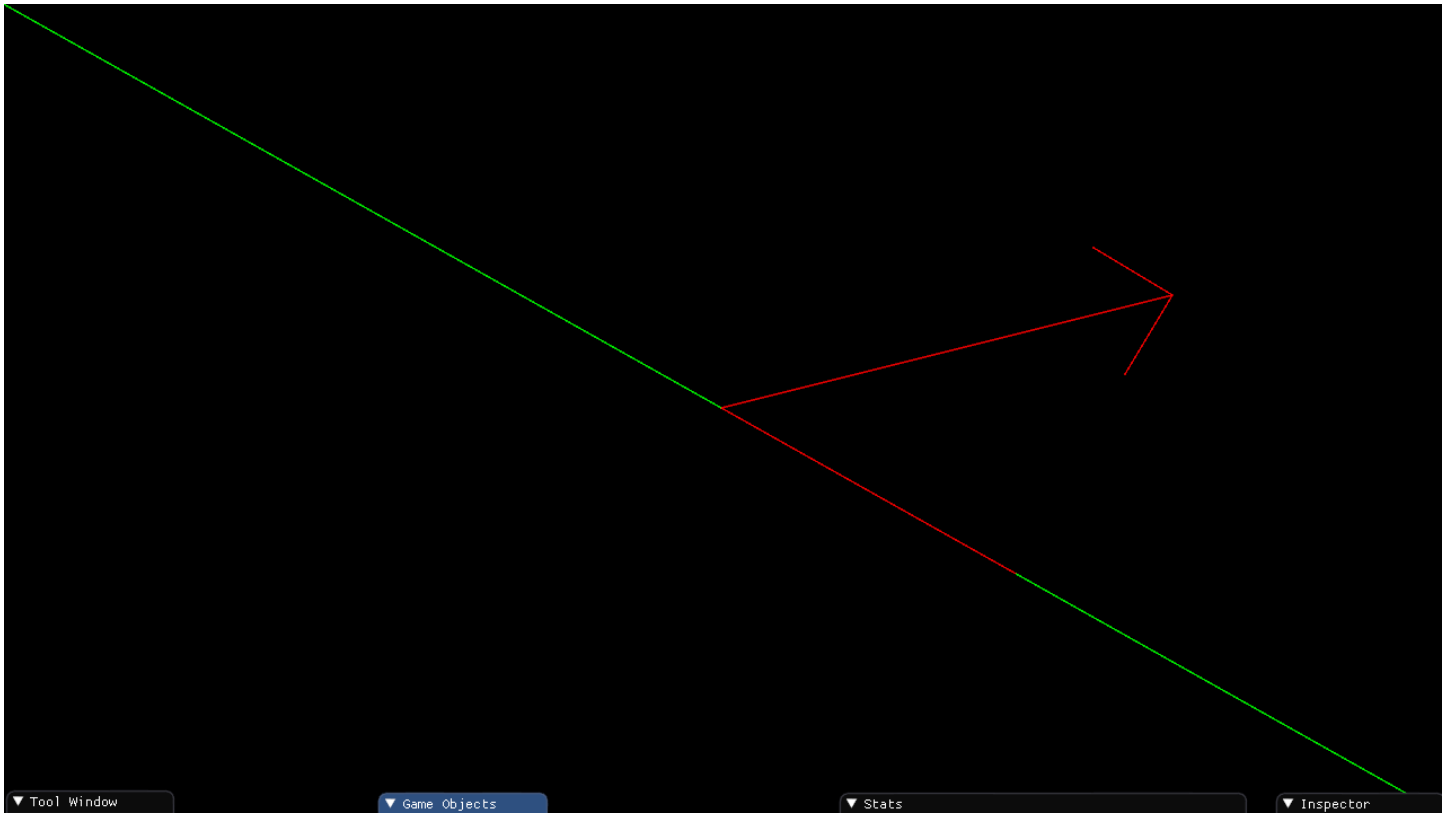
This test demonstrates addition, subtraction, multiplication, division and projection of p2Vec2's.

A red vector is located in the middle of the screen to which unitary X and Y vectors can be added or subtracted from. It can also be multiplied and divided by 2.

A green line traverses the screen, upon which the projection of the red vector is marked in red.

Controls:

Left arrow:	subtract	(1,0) from	red vector.
Right arrow:	add	(1,0) to	red vector.
Up arrow:	subtract	(0,1) from	red vector.
Down arrow:	add	(0,1) to	red vector.
Left shift:	multiply the red vector by 2.		
Left ctrl:	divide the red vector by 2.		



Test_02: Bodies vs Forces:

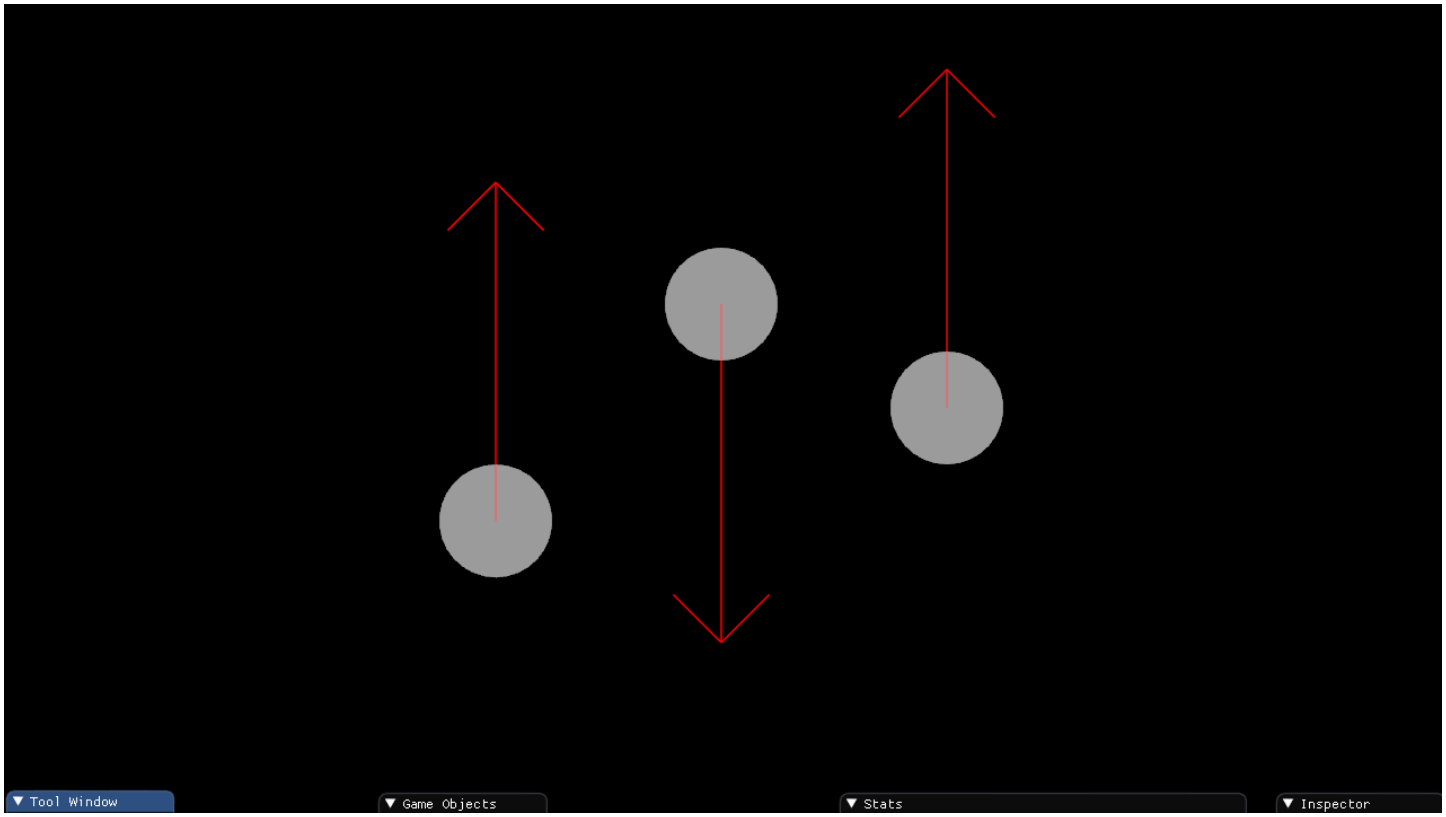
This test demonstrates bodies of different types with forces.

A dynamic body is located on the left side of the screen. It is subject to gravity and an upward force is applied to it if it reaches the bottom third of the screen.

A kinematic body is located in the middle of the screen. It is not subject to gravity and an initial upward velocity is given to it. If it reaches either the top or bottom third of the screen, it's Y velocity is reversed.

A static body is located on the right side of the screen. It is not subject to gravity. An upward force is applied to it every second.

When a force is applied to any body, a red vector representing the force flashes.



Test_03: AABBs:

This test demonstrates the ability to detect a potential overlap using AABBs.

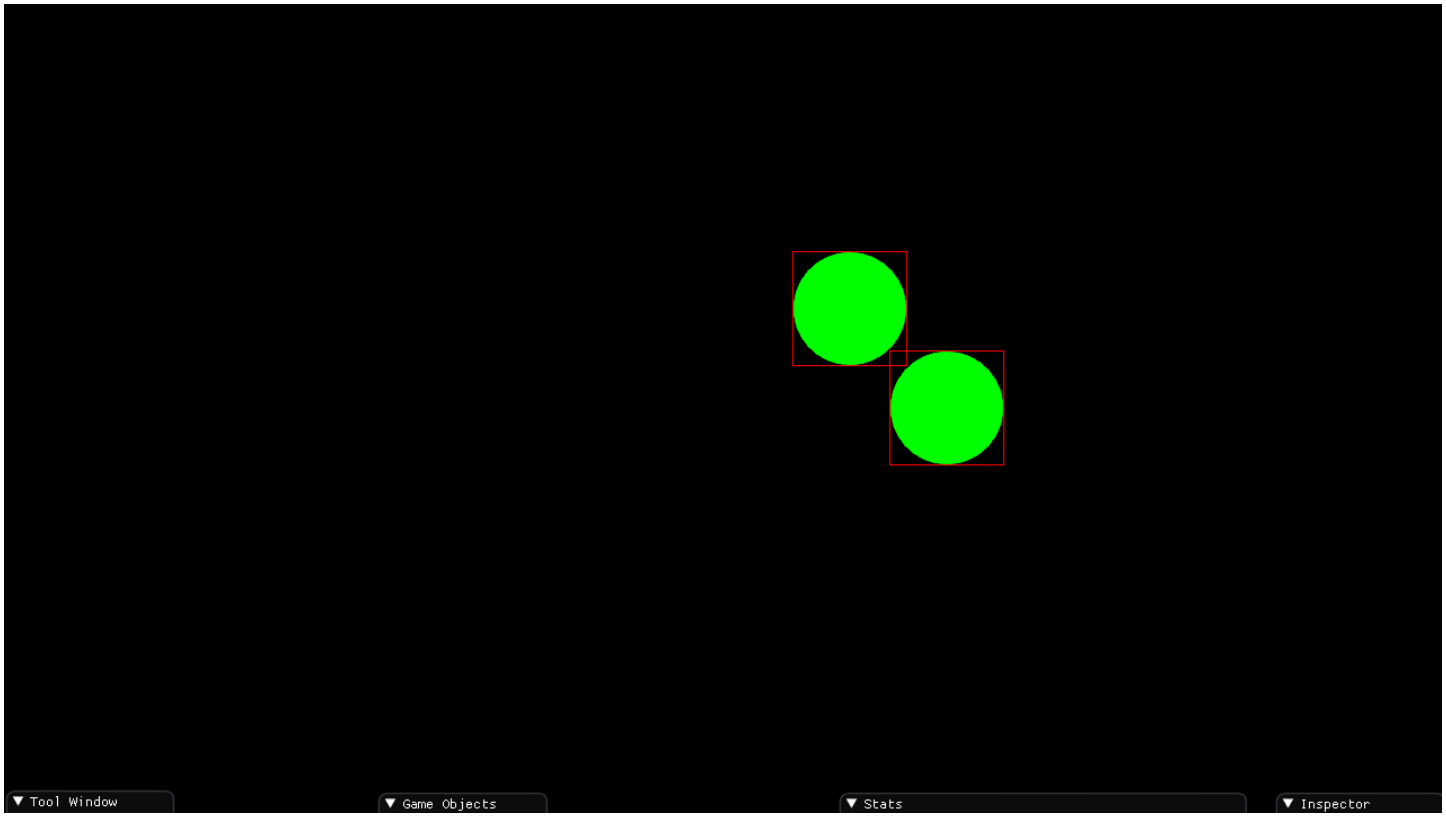
Two circles are placed on screen. Their respective AABBs are drawn in red around them.

If the AABBs overlap, the circles turn green.

Since circles are used for this test, it is possible to show false positives by aligning the corners of the AABBs without allowing the circles to touch, resulting in a change of color despite the absence of contact between circles.

Controls:

Arrow keys: Move one of the circles around.



Test_04: Bodies vs Collisions:

This test demonstrates collision interactions between bodies of different types.

A wide static rectangle is located at the bottom of the screen. A static square is located in the middle of the screen.

A dynamic square is launched at the static square. It impacts with it and falls down on the wide static rectangle.

A kinematic rectangle moves across the screen from the left and traverses both the static and dynamic squares.

The rectangles change color to green when in contact with one another.

Controls:

R: Reset the scene.

Test_05: QuadTree:

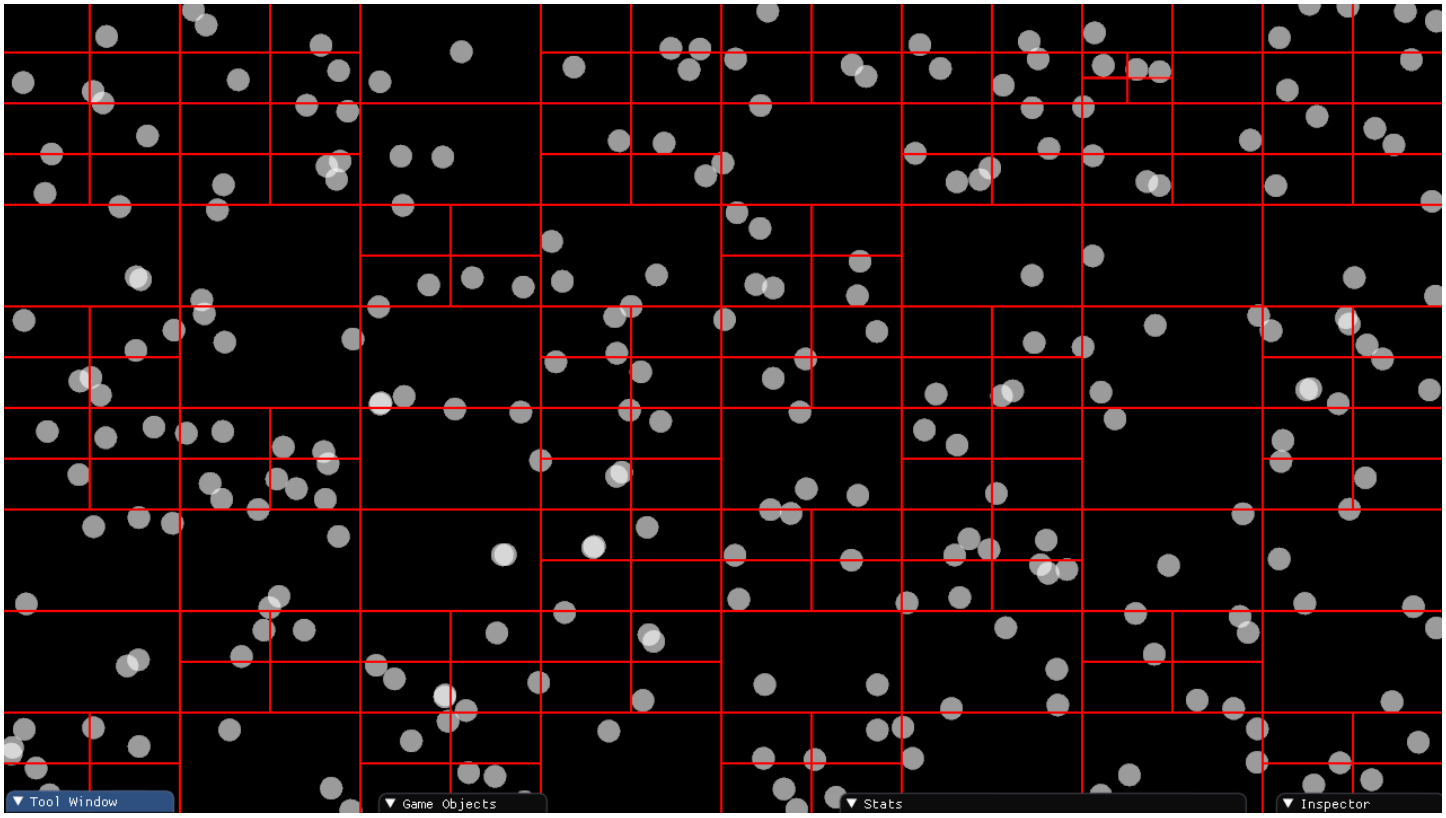
This test demonstrates the QuadTree space partitioning system.

256 kinematic circles are placed randomly on the screen and given a velocity ranging between (-1;-1) and (1;1).

The QuadTree updates every fixed update.

Each Quad's bounds are displayed on screen in red. It splits if it can when more than two bodies are contained within it's bounds.

If a body reached the edge of the screen, it's velocity is reversed.



Test_06: Pendulum:

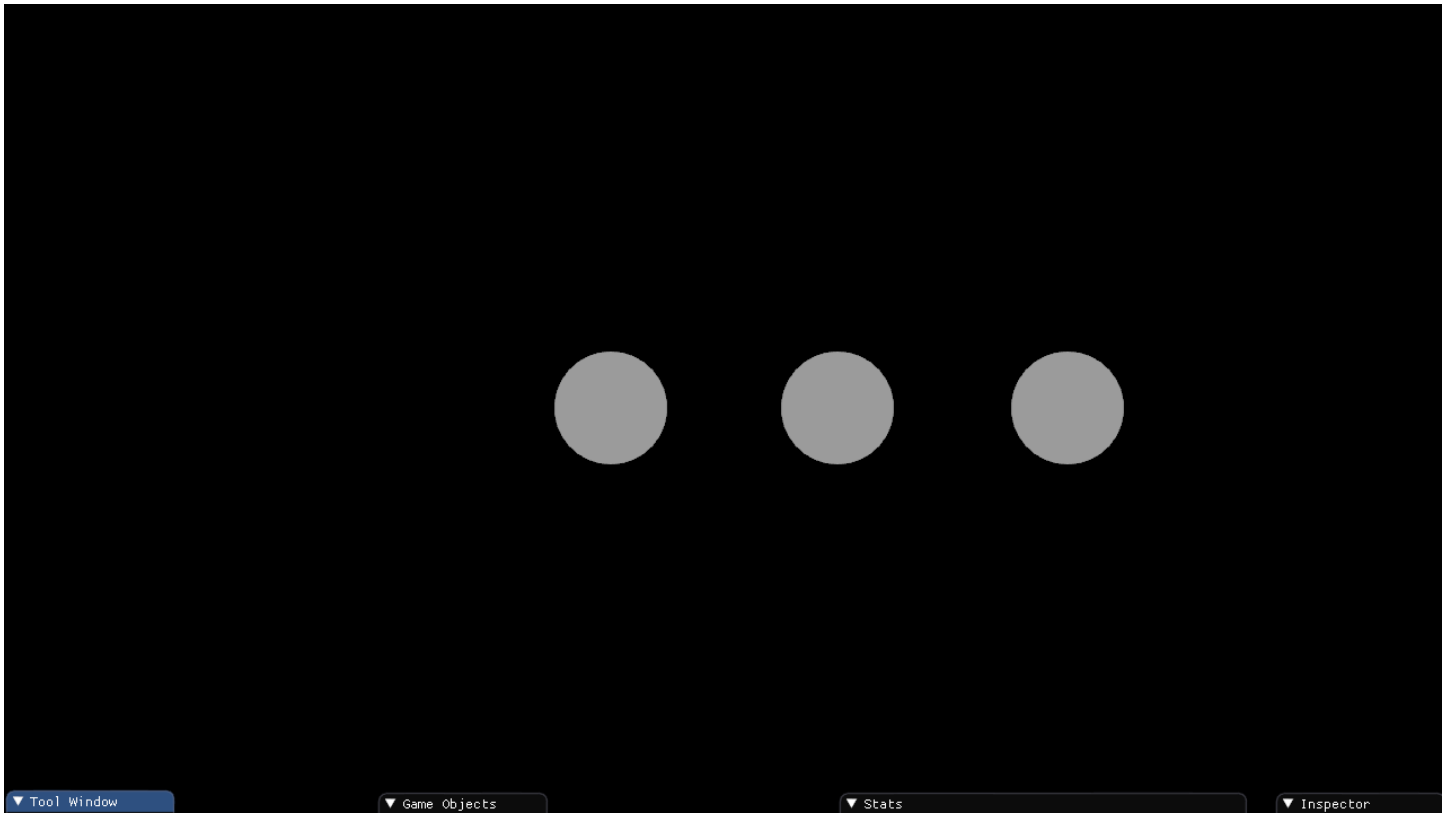
This test demonstrates the conservation of kinematic energy upon collision.

Three dynamic circles are placed in the middle of the screen spaced apart horizontally. Each circle has a mass of 1 and a restitution of 1.

The leftmost circle has an initial velocity and collides with the middle circle. Upon collision the first circle loses all it’s velocity by transferring it all to the second circle. The pattern repeats between the second and third circles.

If a body reaches the screen’s border, it’s velocity is reversed.

The circles keep colliding indefinitely since there is no energy loss.



Test_07: Restitution and Mass:

This test demonstrates the influence of mass and restitution on collision resolution.

Eight circles are placed on screen as to form four rows and two columns.
There are big and small circles. The big ones have a mass of 1, the small ones have a mass of 0.5.

If a body reaches the screen's border, it's velocity is reversed.

Uneven rows have a restitution of 1. Even rows have a restitution of 0.3.

All circles on the left have an initial velocity forcing them to collide with their neighbor to the right. Different sets of circle behave differently upon collision.

The first row (masses 1 and 1, restitutions 1 and 1) behave identically to the pendulum from the preceding test.

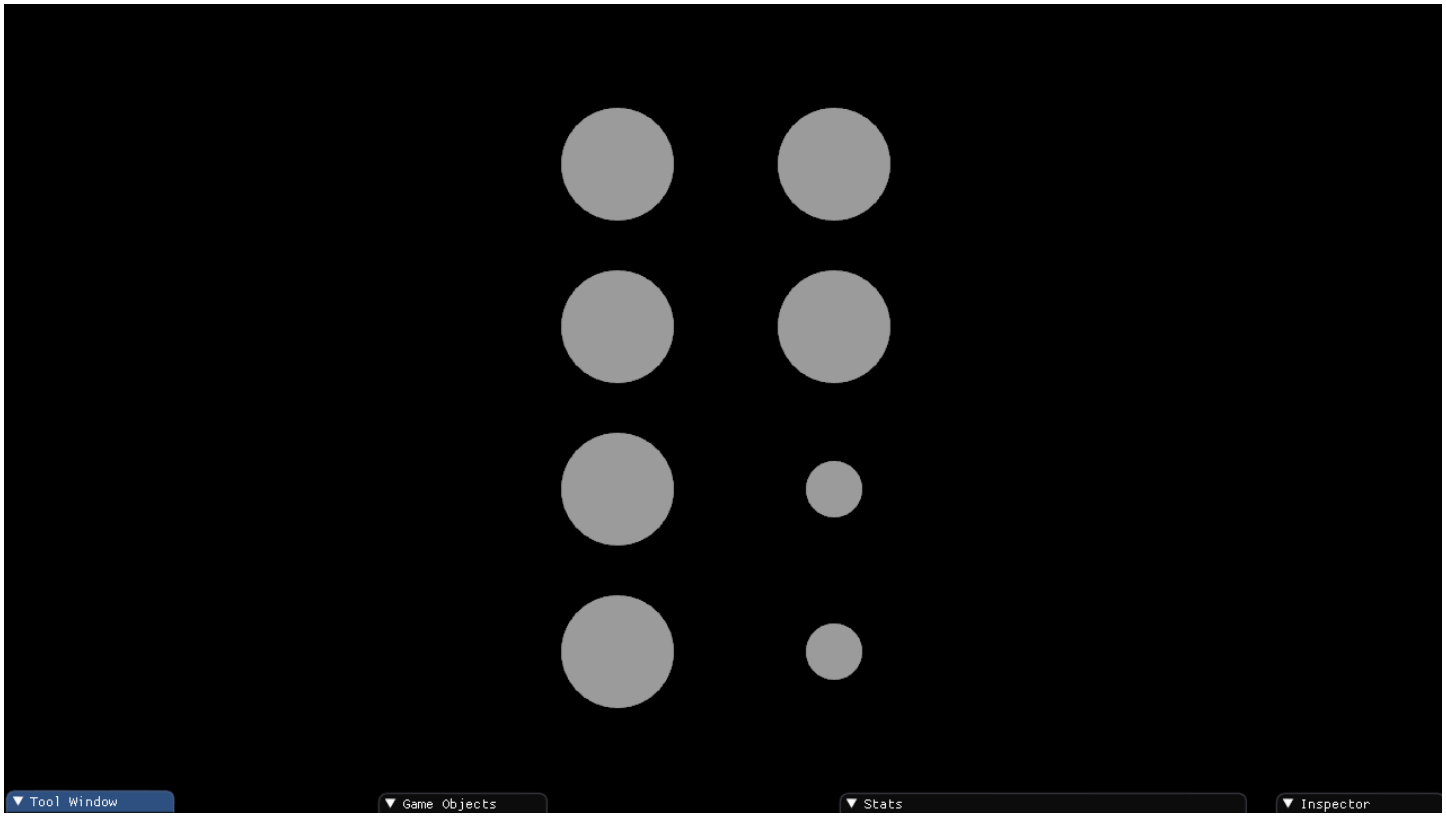
The second row (masses of 1 and 1, restitutions of 0.3 and 0.3) loses energy with every collision and slows down. Upon collision, only the 2/3 of the velocity is transferred to the other circle which means that the left circle keeps some velocity after colliding with the right circle and vice versa.

The third row (masses of 1 and 0.5, restitutions of 1 and 1) don't lose energy and collide with each other indefinitely. Upon collision, the bigger circle keeps some velocity due to it's mass in accordance to the inelastic collisions model.

The fourth row (masses of 1 and 0.5, restitutions of 0.3 and 0.3) loses energy with every collisions and slows down. It demonstrates the same behavior and the second and third rows combined.

Controls:

R: Reset scene.



Test_08: Explosion:

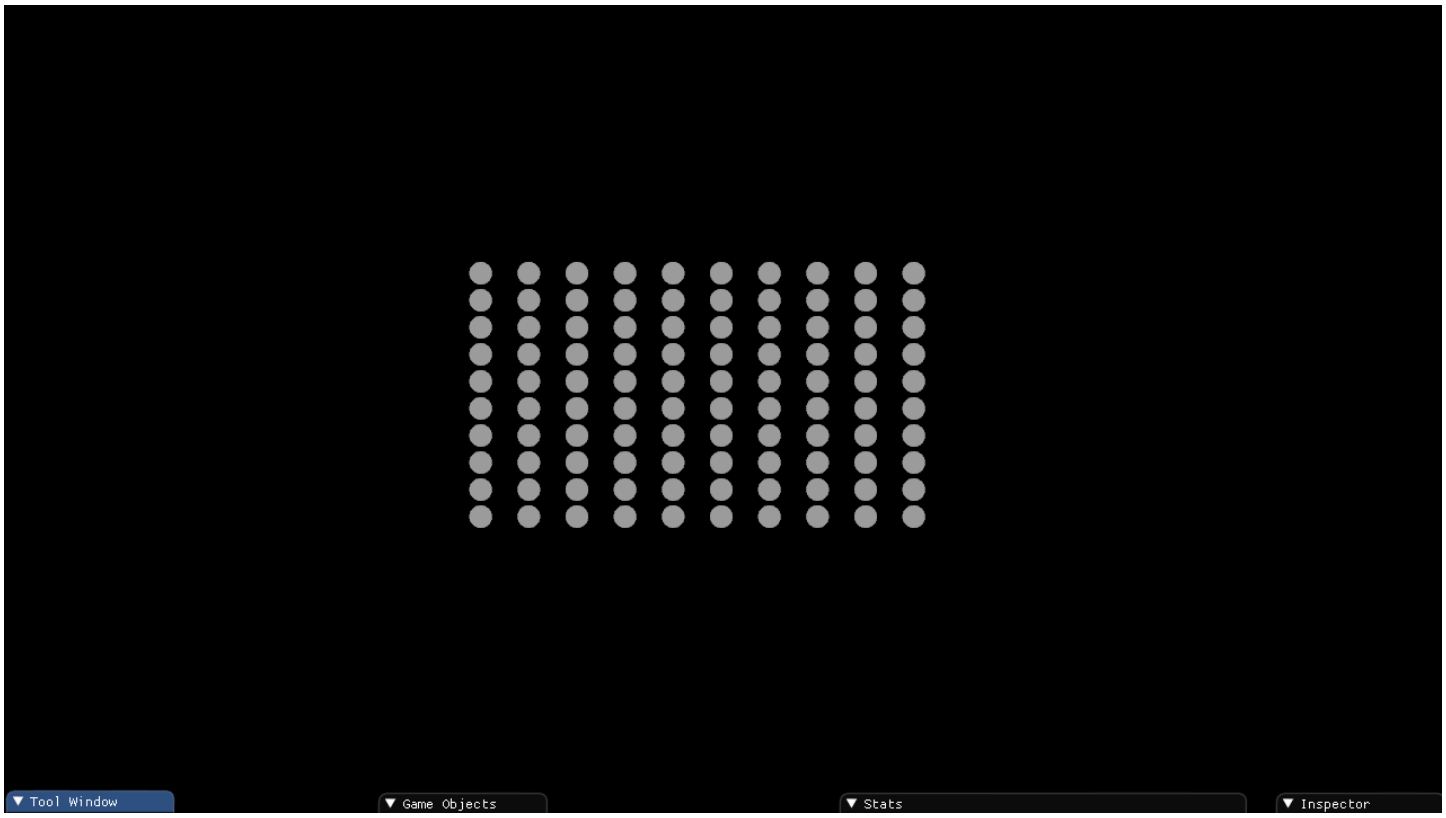
This test is here mostly for fun.

100 dynamic circles are arranged in the middle of the screen. And explosion can be triggered in the center of the screen, sending the bodies flying in all directions.

If a body reaches the screen’s border, it’s velocity is reversed.

Controls:

R: Reset scene.
E: Trigger an explosion.



Test_09: Many Circles:

This test demonstrates the resolution of collisions between a large number of circular shapes.

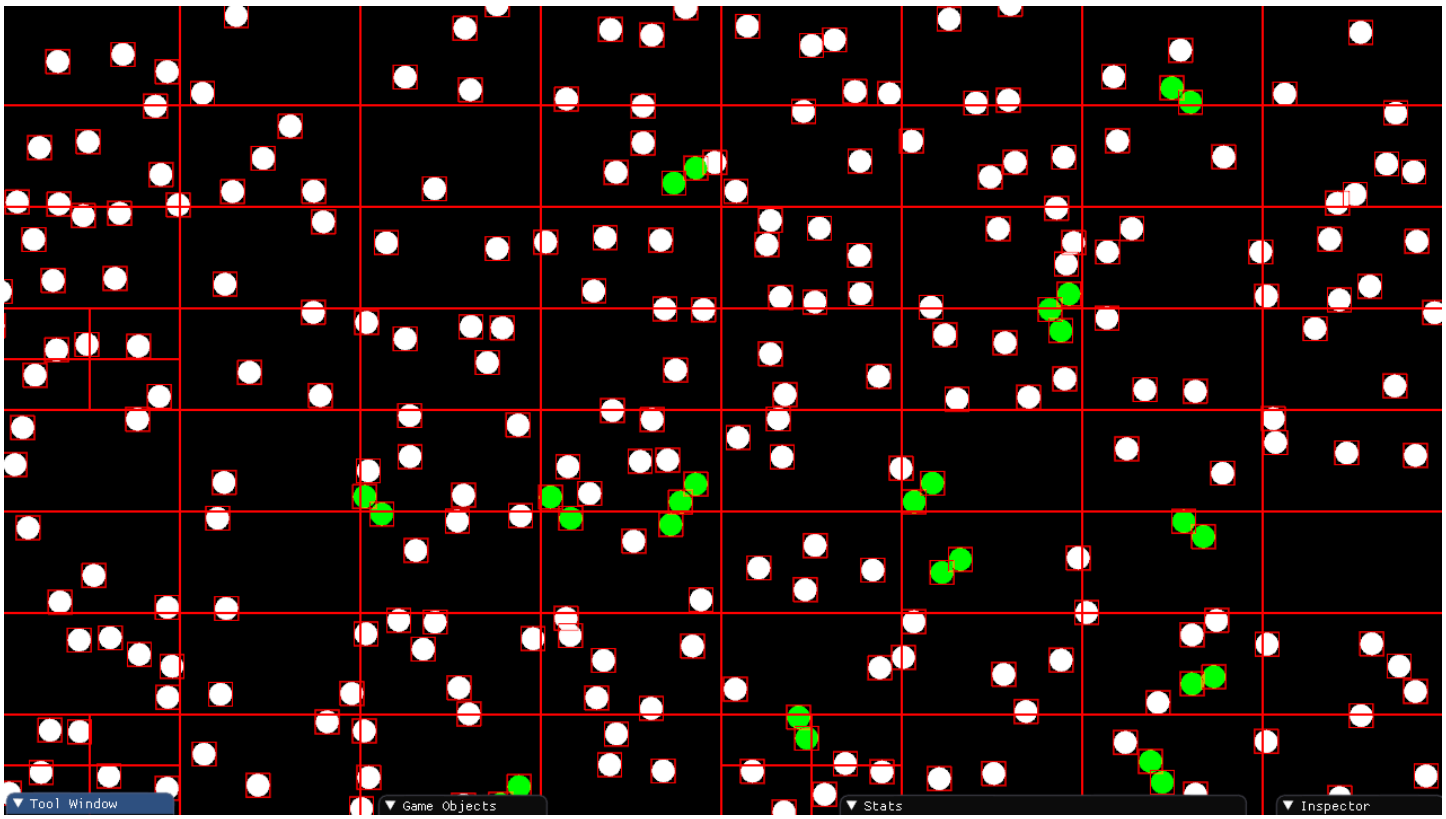
256 dynamic circles are placed randomly on screen and given a random velocity between (-1;-1) and (1;1).

If a body reaches the screen's border, it's velocity is reversed.

Each Quad's bounds are displayed on screen in red.

Each body's AABB is displayed in red around the circle. If two AABBs overlap, the shape changes color to green.

The collisions are resolved as a circle vs circle collisions.



Technical choices:

General considerations:

This project's focus has been to implement the **bare bones** of a physics engine, therefore only the most basic functionalities have been implemented.

All movement, forces and collision calculations assume a **rotation of 0°**.

The only shapes integrated into the engine are **circles and rectangles**.

Friction is not integrated into the engine.

Bounciness is not integrated into the engine.

Physical bodies representation:

A physical body is represented by a **p2Body**. It has a position, a linear velocity, a mass and **a restitution**. It also has **one p2Collider**.

A **p2Collider has one p2AABB** and it can be a "sensor" meaning it will not trigger collisions but will send collision messages.

A **p2Collider has one p2Shape** which is either a p2CircleShape or a p2RectShape.

Collision resolution:

p2World has a p2ContactManager. All **collision resolution is handled by p2ContactManager**.

p2ContactManager has a pointer to a contact listener and it is the **p2ContactManager** that **sends collision messages**.

Position correction data is generated by **p2Collider**. **Forces and velocities** resulting from a collision is generated and applied by **p2Body**.

For **position correction** of a collision **between two Rects**, the **penetration data** returned by p2Collider **is the smallest penetration delta** out of all penetration deltas, each delta corresponding to one axis.

Spatial partitioning:

A **QuadTree** system has been used for spatial partitioning. A **Quad's children** are stored on **heap** memory to handle a Quad's recursive nature.

The QuadTree's **Retrieve()** function has been split between **Retrieve()** which only handles the **root Quad** and **RetrieveRecursively()** which handles all **other Quads**.

A **body overlapping multiple Quads** is stored in the **parent** Quad of the **Quads it overlaps**.

Posession Class Diagram:

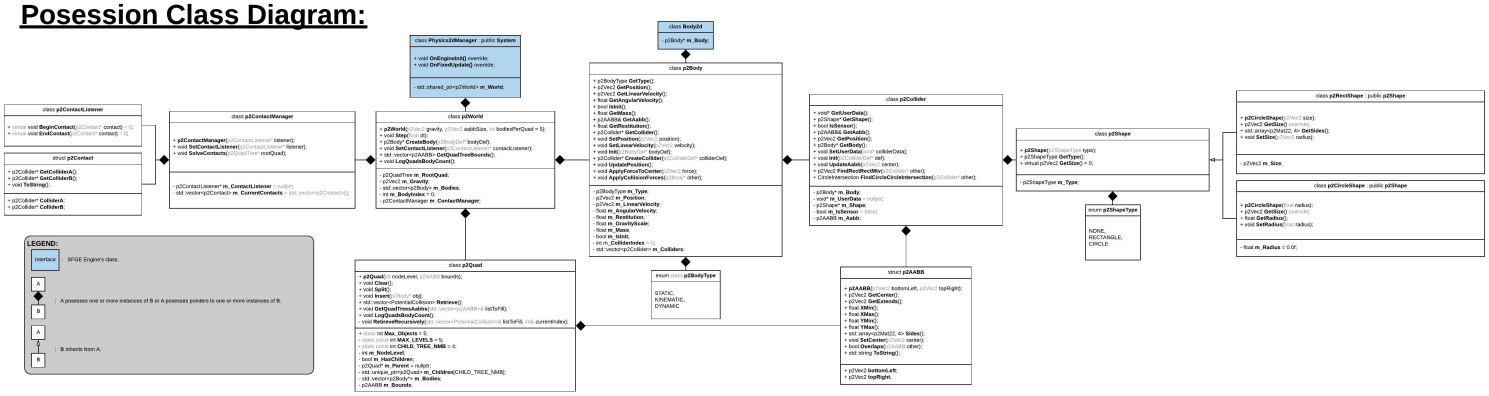


Fig.1: Overlapping detection in 1D of two segments.

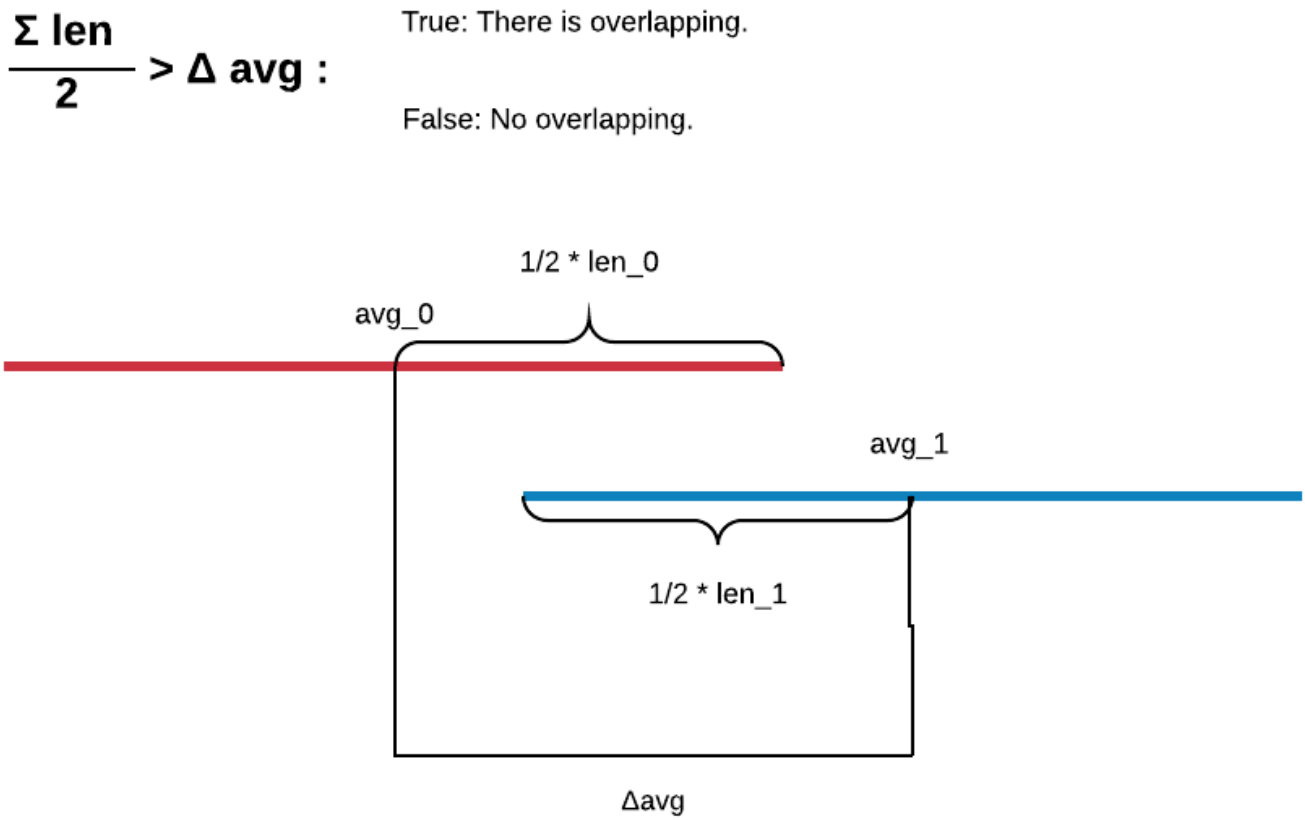
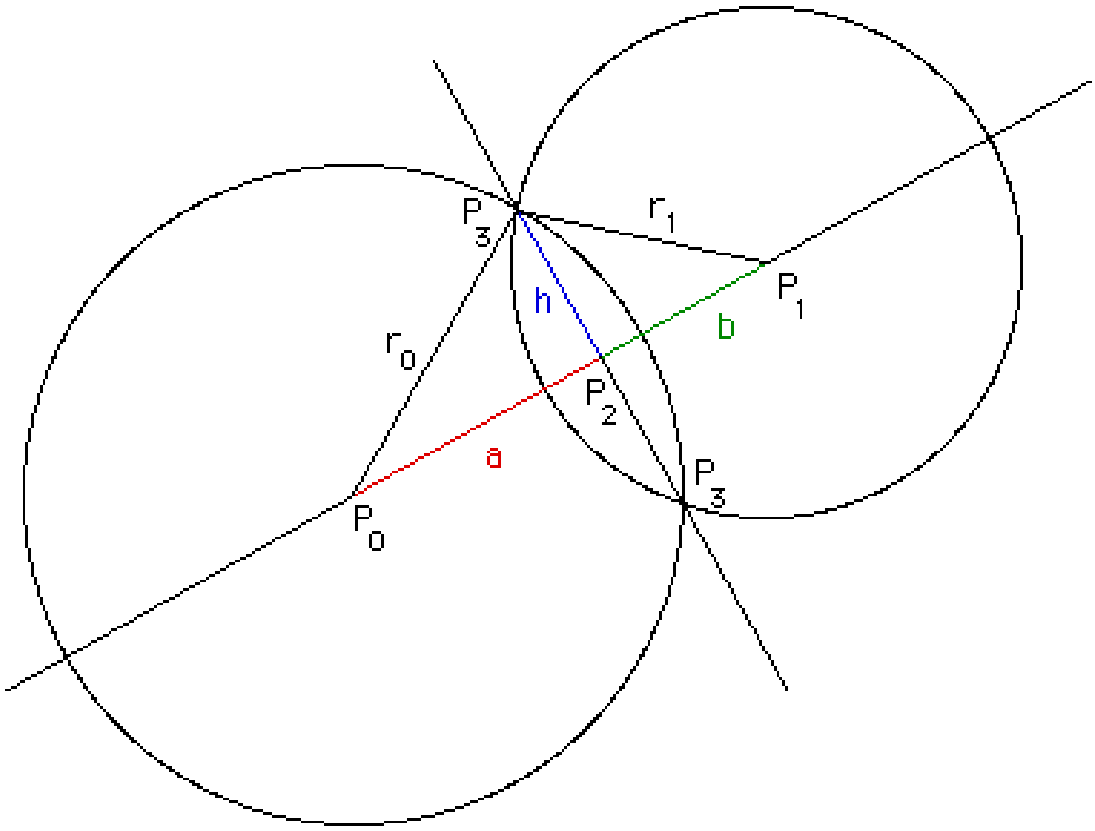


Fig.2: Finding intersections of two colliding circles.



Taken from: http://paulbourke.net/geometry/circlesphere/

Fig.3: Vector projection.

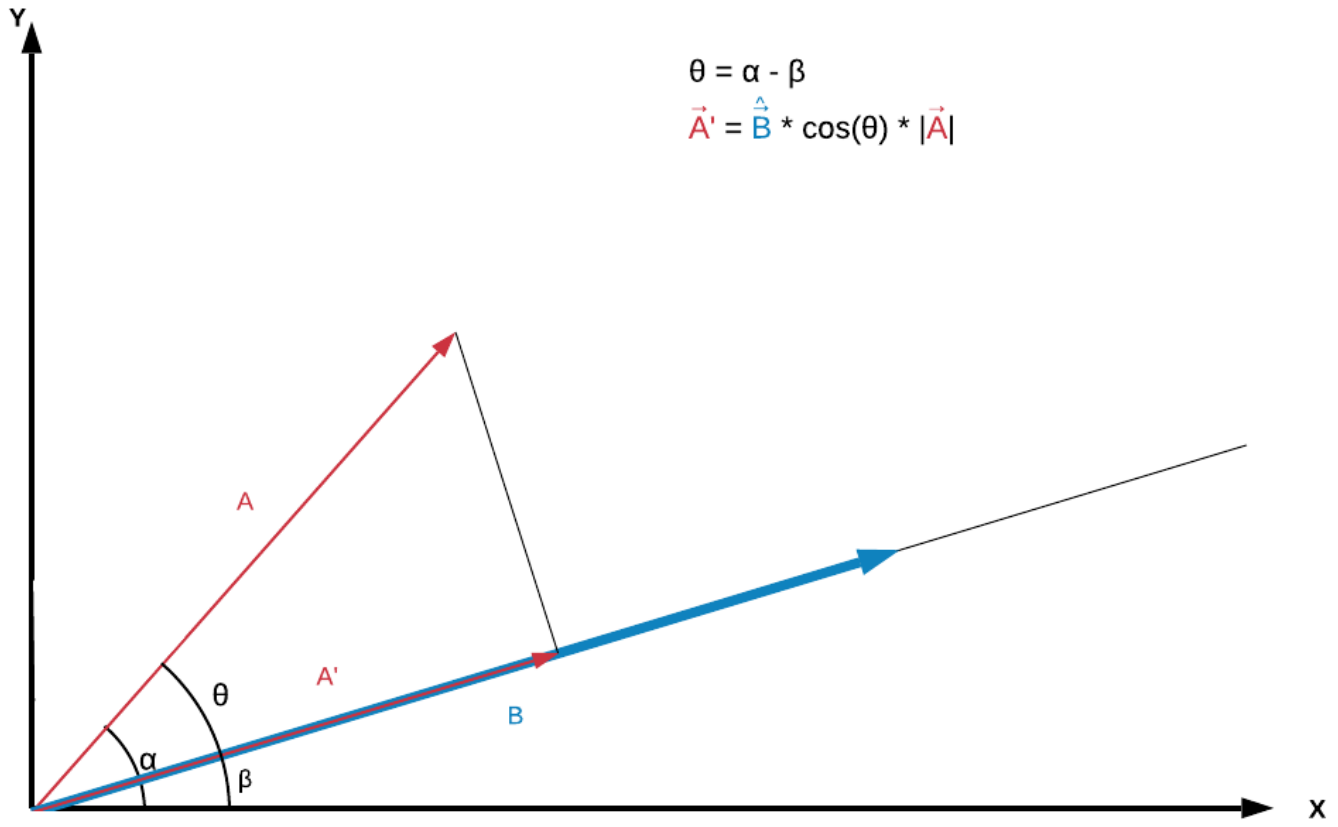


Fig.4: Segment projection.

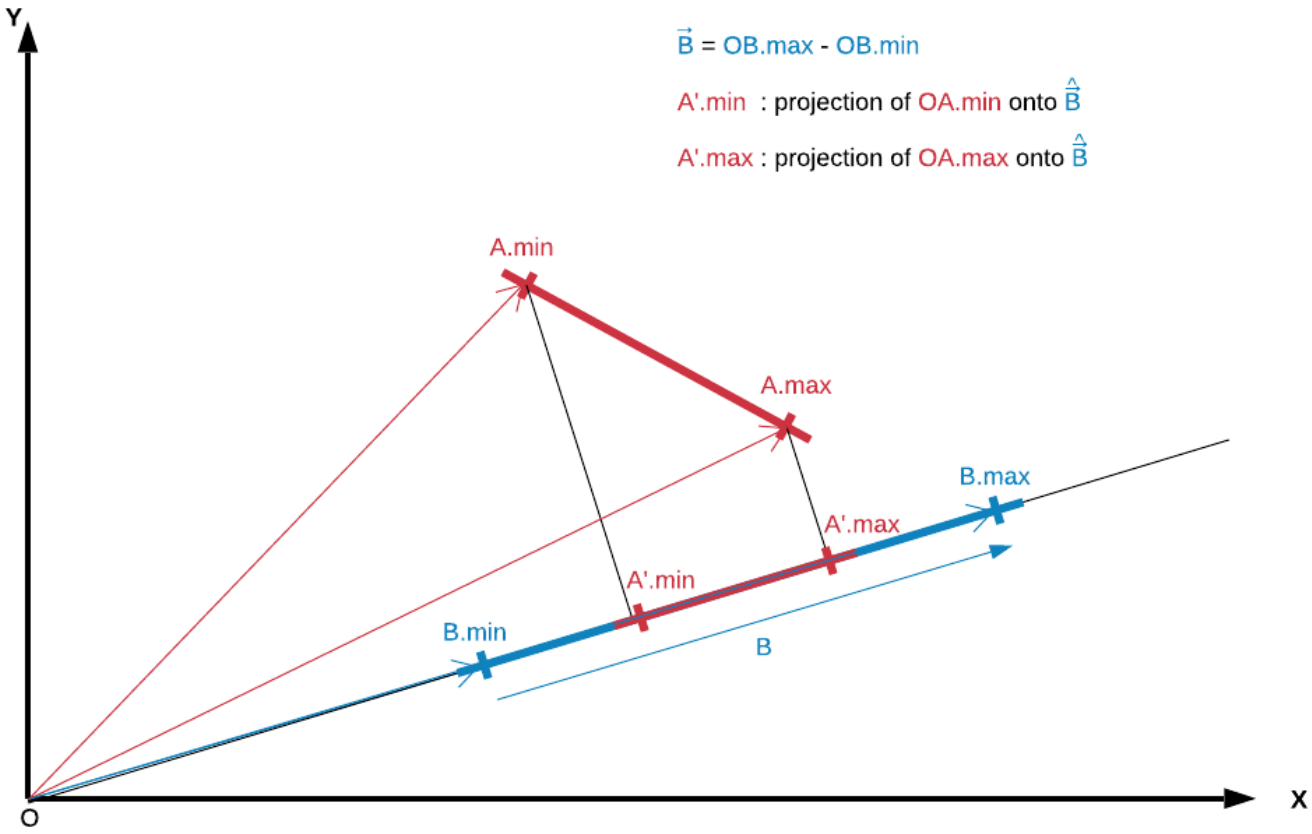


Fig.5: Finding the MTV with SAT.

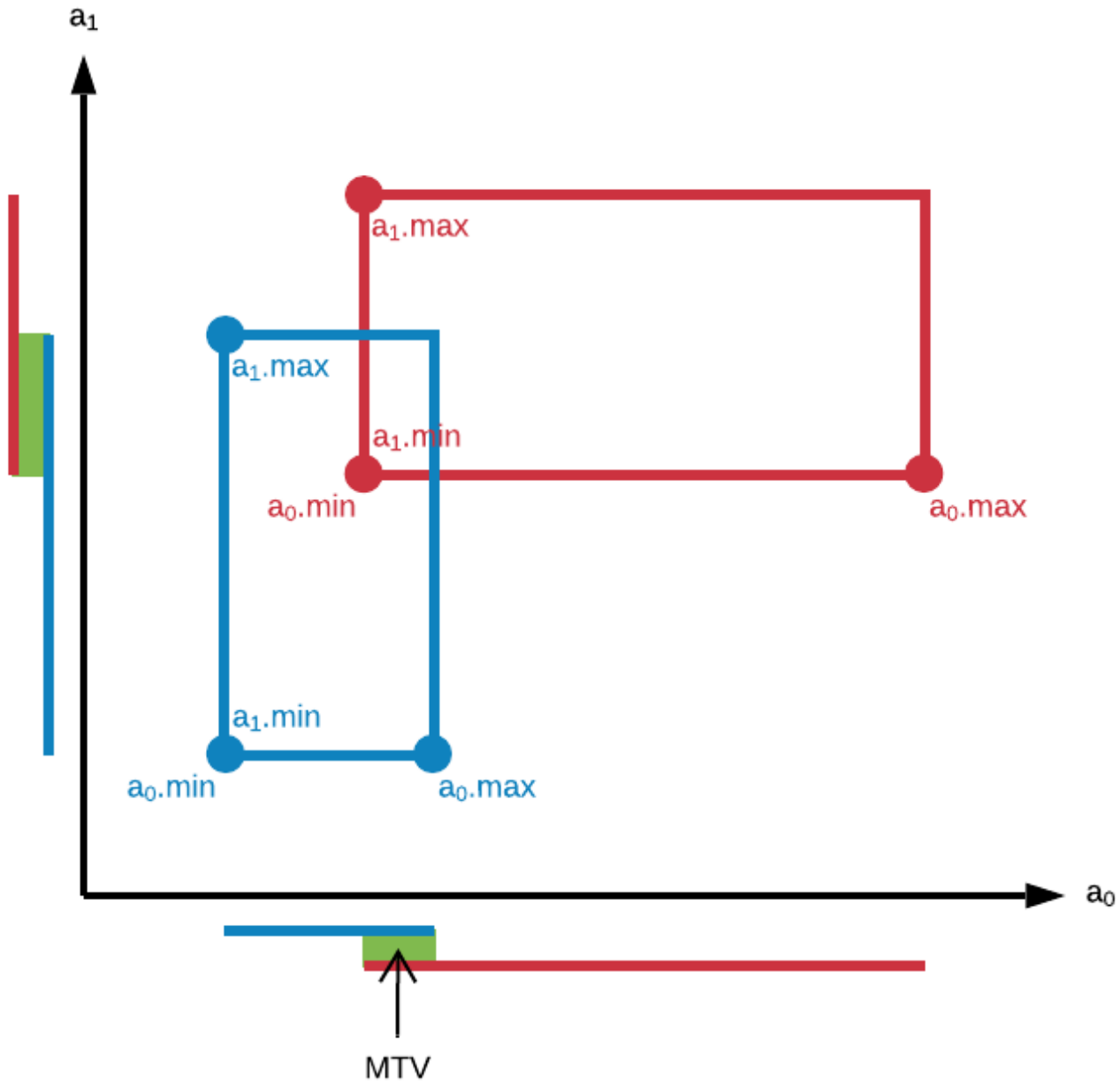


Fig.6: Calculating the reflection of a vector.

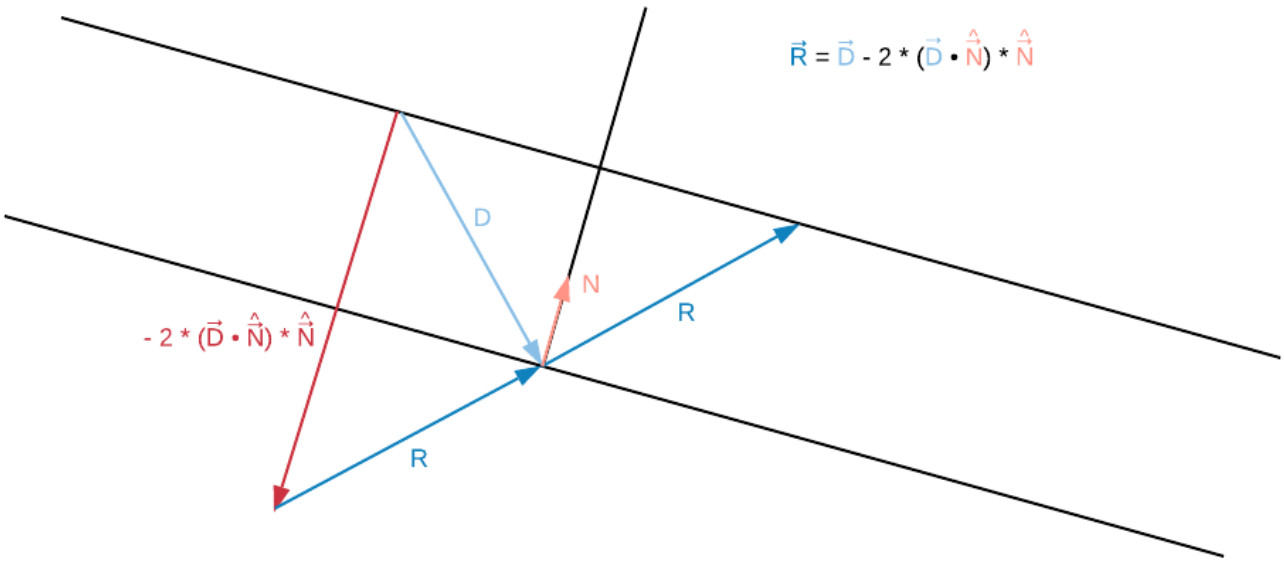
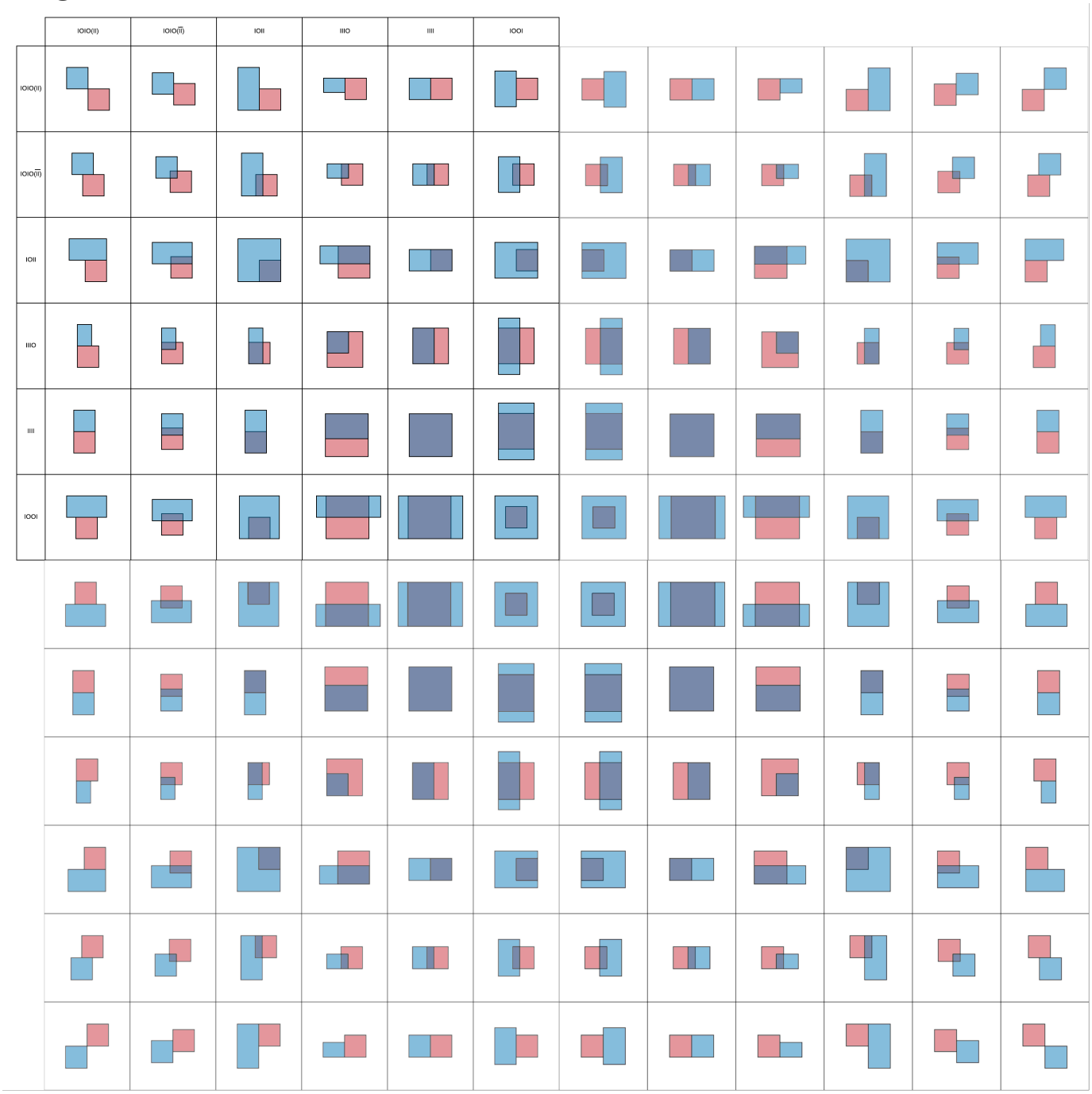


Fig.6: 144 reasons to use SAT. *(aka. all possible ways two rectangles can overlap)*



IOIO(II) = This.Min < Other.Min && This.Max < Other.Max && This.Max == Other.Min

IOIO(Ī) = This.Min < Other.Min && This.Max < Other.Max && This.Max != Other.Min

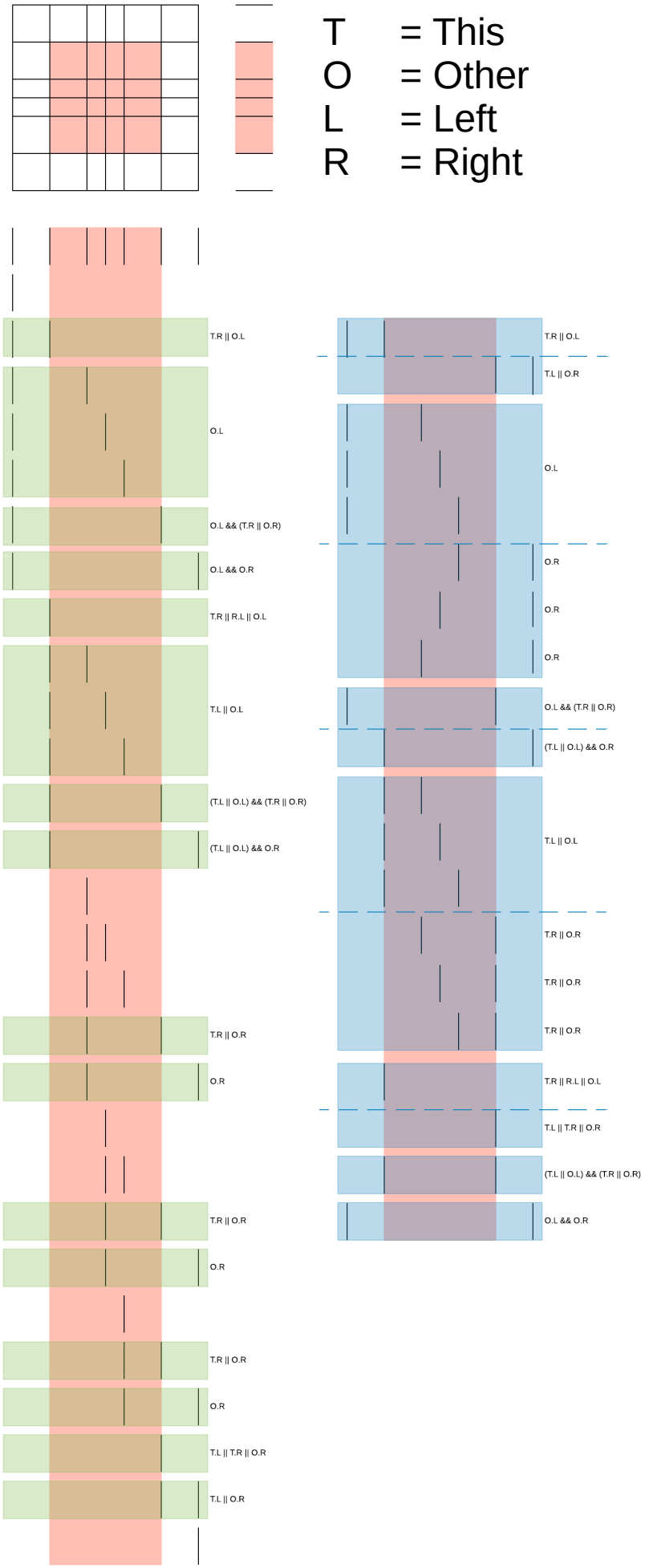
IOII = This.Min < Other.Min && This.Max == Other.Max

IIIO = This.Min == Other.Min && This.Max < Other.Max

IIII = This.Min == Other.Min && This.Max == Other.Max

IOOI = This.Min < Other.Min && This.Max > Other.Max

Fig.7: 28 reasons to use principle described in Fig. 1. *(aka. all possible combinations of min and max coordinates to determine intersection coordinates)*



Bibliography and credits:

- Thanks to Nicolas Schneider for his feedbacks and advices on this project.

- Thanks to Jacob from stackoverflow.com for his circle vs circle intersection finding script: https://stackoverflow.com/questions/3349125/circle-circle-intersection-points

- Thanks to dyn4j.org for providing a concise explanation of SAT: http://www.dyn4j.org/2010/01/sat/#sat-inter

- Thanks to lucidchart.com for providing a great schema creation tool.