Faculty of Computing, Engineering and Science

Module Title: Object Oriented Programming with Data Structures and Algorithms

Module Code: CS4D768

Lecturer: Dr Janusz Kulon

Assessment Type: Practical Written Work 1

Assessment Title: Collision detection and response algorithm

Weighting: 50%

Submission Date: **17h Nov 2023**

Return Date: 15th Dec 2023

Collision detection and response algorithm

Introduction

Collision detection algorithms determine when and where two objects came into contact. Collision resolution algorithms determine how to resolve the collision between the objects. According to Ericson (2005) there are numerous fields that utilize collision detection and resolution including computer games, physical simulation, and robotics.

Collision detection and resolution algorithms are vastly used in the field of computer games. Without collision detection and resolution, the games will lose their realistic and immersive gam-ing experience. Collision detection changes the game’s digital world experience into a real-world experience. It prevents objects in games from intersecting or going through each other, Ultimately making the game stable for players to enjoy.

Collision detection and response algorithms are also used in physical simulations, such as those utilized in engineering and scientific research. These algorithms enable scientists and engineers to analyze the behavior of objects under diverse conditions, including during impact events or exposure to extreme forces. For example, collision detection and resolution algorithms might be used in car crash simulation. It can simulate how a car deforms or bends after it collides with the wall.

Collision detection and response algorithms are indispensable for robotics. These algorithms empower robots to navigate their surroundings safely, avoiding obstacles and dangers. They are also crucial for controlling the movement of robotic arms.

These fields have different requirements. It’s either real time performance or real time efficiency Ericson (2005). In computer games, collisions must be resolved in real time to maintain a smooth and responsive gameplay experience. In physical simulations and engineering analysis, accuracy is primary. These applications require precise collision detection and response to accurately model the behavior of objects and systems. In robotics, both performance and precision are important. Firstly, robots need to be able to detect and respond to collisions in real time to avoid obstacles and dangers. Secondly, they also need to be able to do so accurately to avoid damaging themselves or other objects.

Collision detection and resolution algorithms depend on many things. Such as the shape of the object like sphere, triangle, square and other polygonal object. It can also be complex objects with different shapes embedded into them. It also depends how many objects are in the simulation and their sizes as well. Performance is also a major factor here. The collision must be detected and resolve between a certain time Ericson (2005).

1

There are many ways to detect collisions between objects. For only Circle shape as it doesnt have any vertices only the center it can be done by using brute force. It will come to a stop when there will be many Circles. It can be Optimized using either spatial partitioning or quad tree. Quad tree is the most optimized one. But it’s hard to detect polygonal objects because of their random edges and vertices. There is a method named separate axis theorm made by Gottschalk et al. (1996). The separating axis theorem is a method for efficiently detecting collisions between convex polygons. It works by building a hierarchy of oriented bounding boxes (OBBs) to cover polygons and groups of polygons. An OBB is a box that fits and contains a convex polygon. The edges of the bounding box are calculated based on the furthest extremes of the polygon it contains. An OBB differs from an axis-aligned bounding box in that it can be rotated so that it better fits its contents. This gives it more flexibility.

Once a hierarchy of OBBs has been established, the separating axis theorem can be applied. This theorem states that if it is possible to draw a line between two objects then those objects are not in contact. Gottschalk et al. (1996) apply the separating axis theorem to test whether OBBs have collided. They project the two OBBs in question onto an arbitrary axis and calculate the half-widths of the two OBBs relative to this axis. If the first half-width plus the second half-width is less than or equal to the distance between the OBB centre positions along this axis, then they may be colliding, and it is necessary to run more tests along different axes to verify if the OBBs are in contact. Otherwise, it is guaranteed that the OBBs are not in contact.

There is also another method for collision detection is GJK algorithm, developed by Gilbert, Johnson and Keerthi (1988). This method uses the Minkowski distance. Minkowski distance is calculated by subtracting all the vertices of the first shape to all the vertices to the second shape. These vertices generate a new shape if the shape encloses the origin that means the two different shapes are colliding (Wikipedia, 2020). But repeatedly recalculating the Minkowski difference and checking whether the resulting shape contains the origin is a costly operation. To reduce this cost Gilbert, Johnson and Keerthi (1988) added a new way. It begins with finding the support points. Support points are those points that are the farthest in each direction. Using the support points, it creates a triangle using Minkowski distance. This is done in three different directions. Once the simplex (triangle) has been calculated, it is determined whether it contains the origin. This is done by checking if the point added last was past the origin in that direction. If it is, then the Minkowski difference contains the origin and therefore there is a collision.

For GJK algorithm you dont need to know the shape of the object only with the vertices it can work perfectly also doesn’t need extra calculation.

2

UML Class Diagram

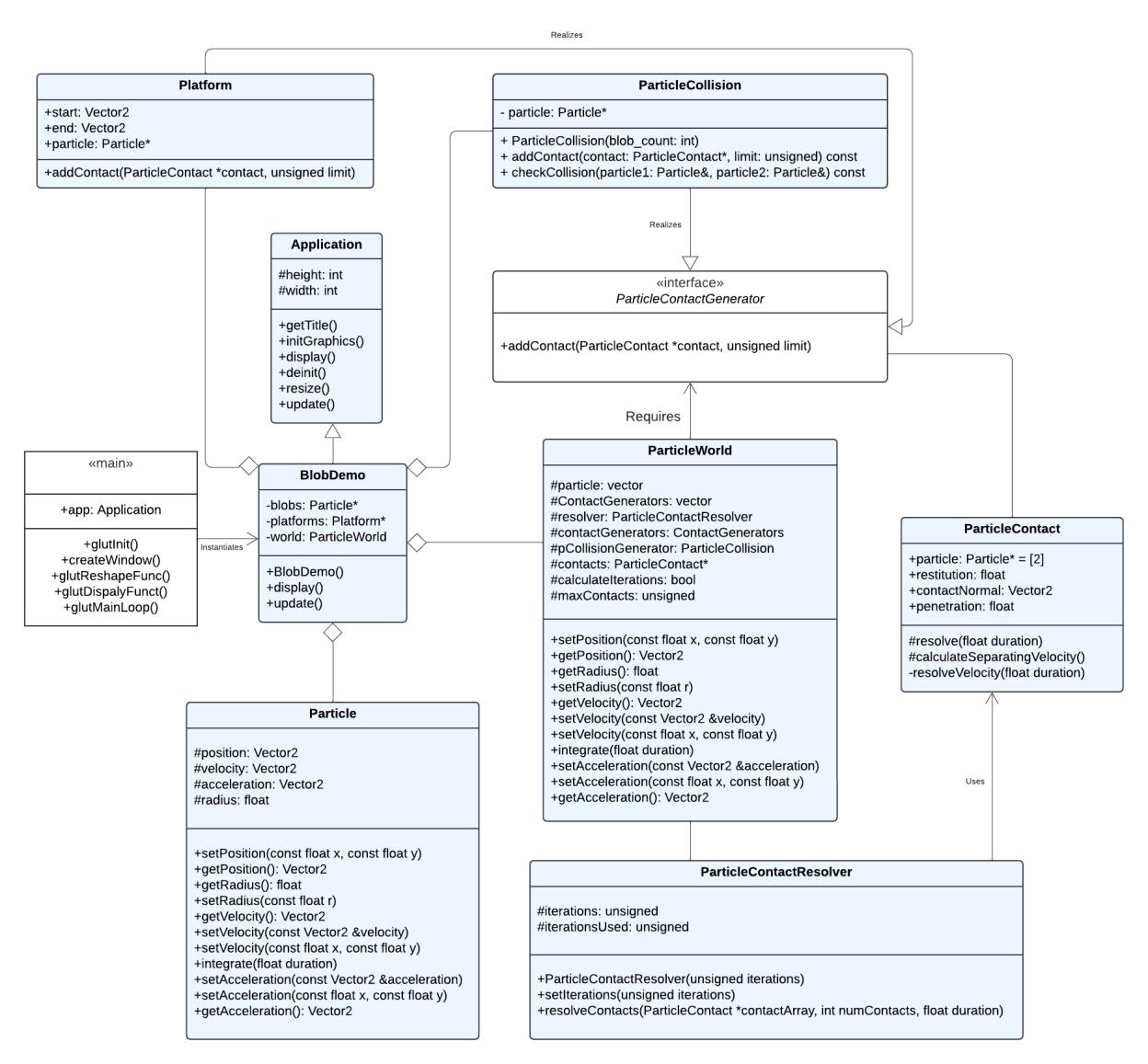
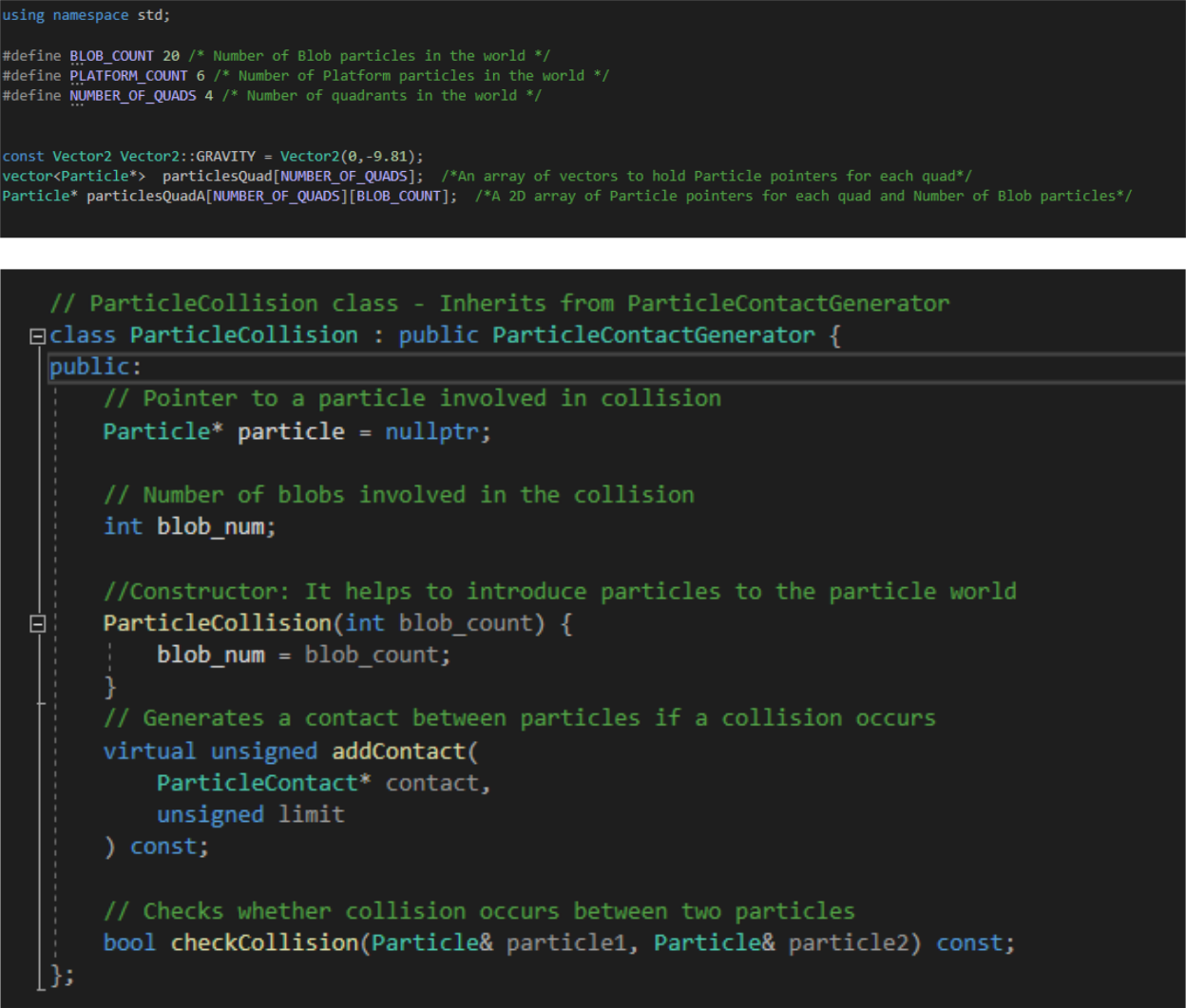


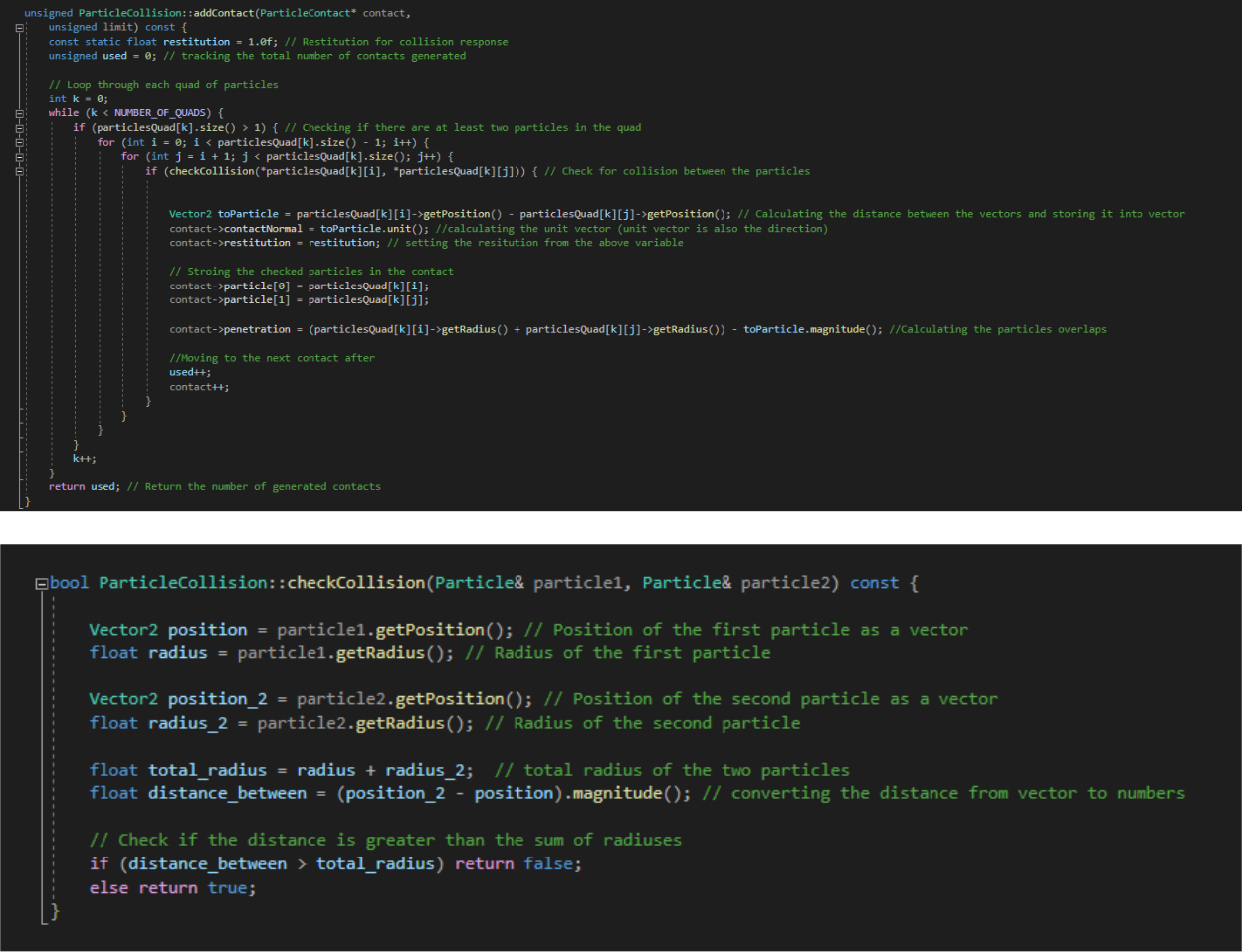
Figure 1: UML Diagram

3

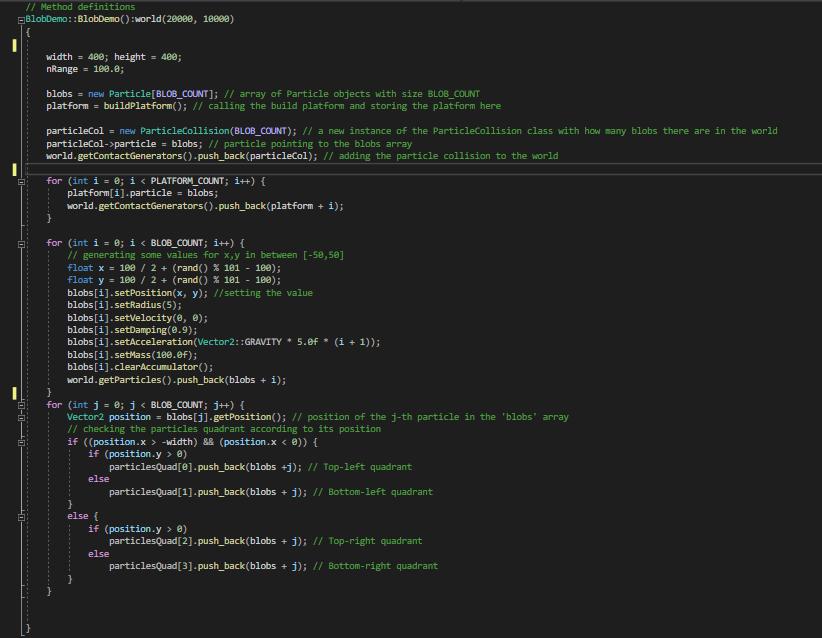
Screenshots of the code



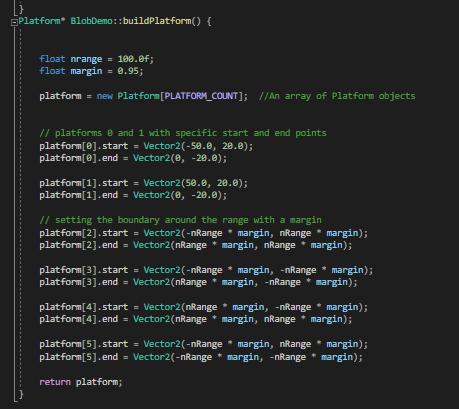
4



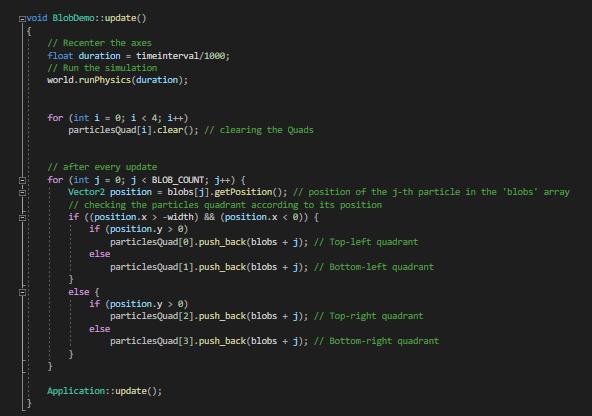
5



6

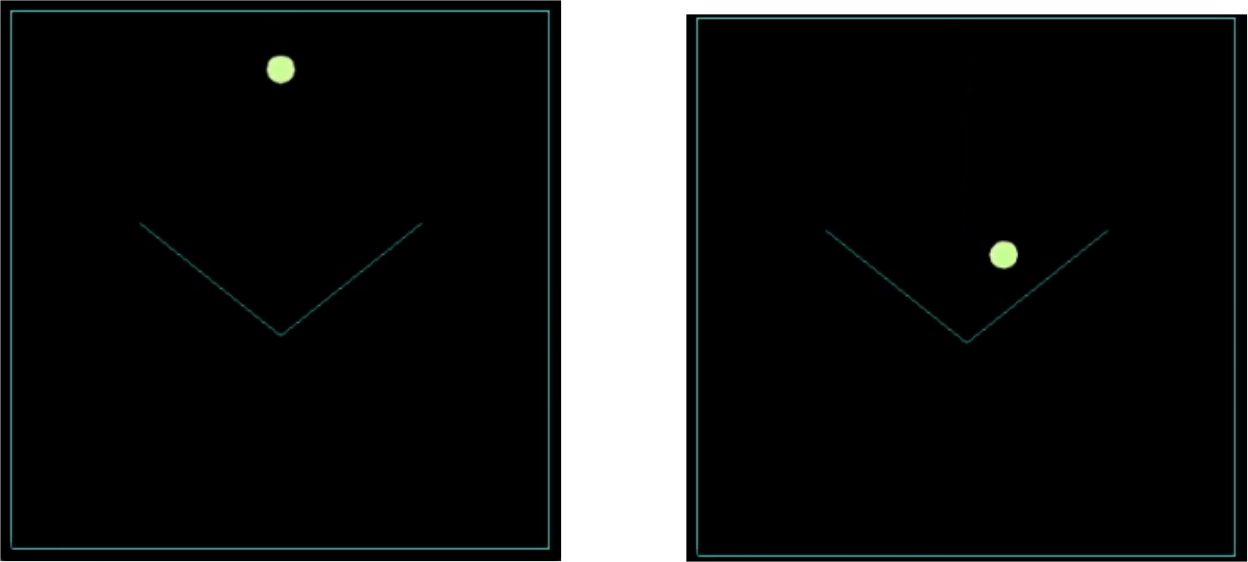


7



Testing and Evaluation

1) Testing 1: Collision between platform and circle



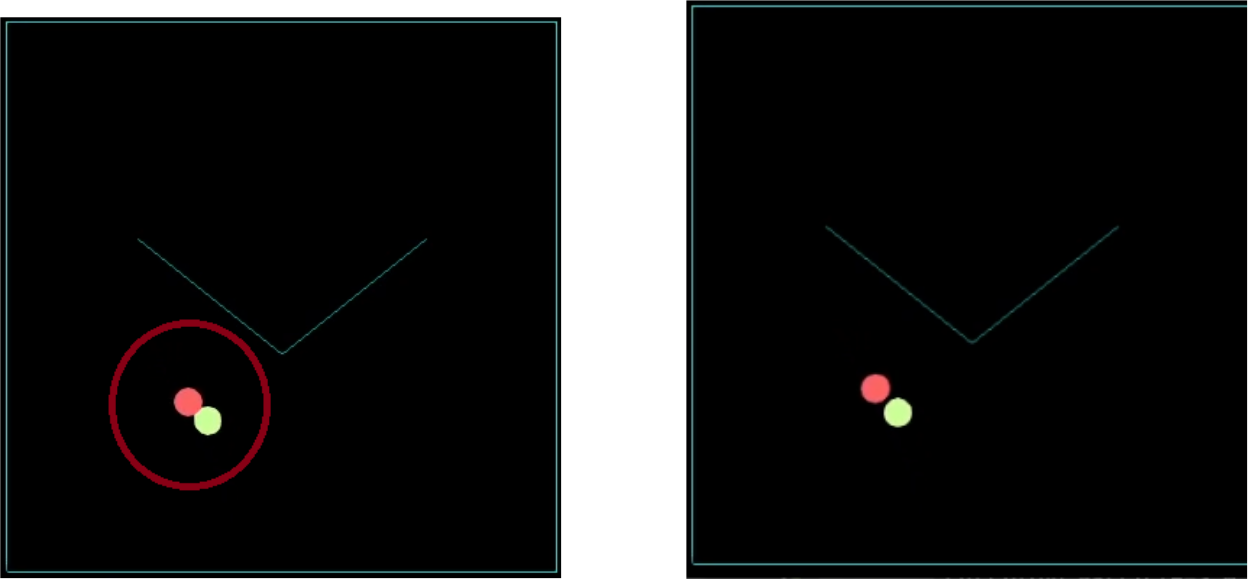
(a) Before (b) after

Figure 2: Collision between platform and circle

This test runs without any errors. The circle collides with the platform.

9

2) Testing 2: Collision between circle and circle



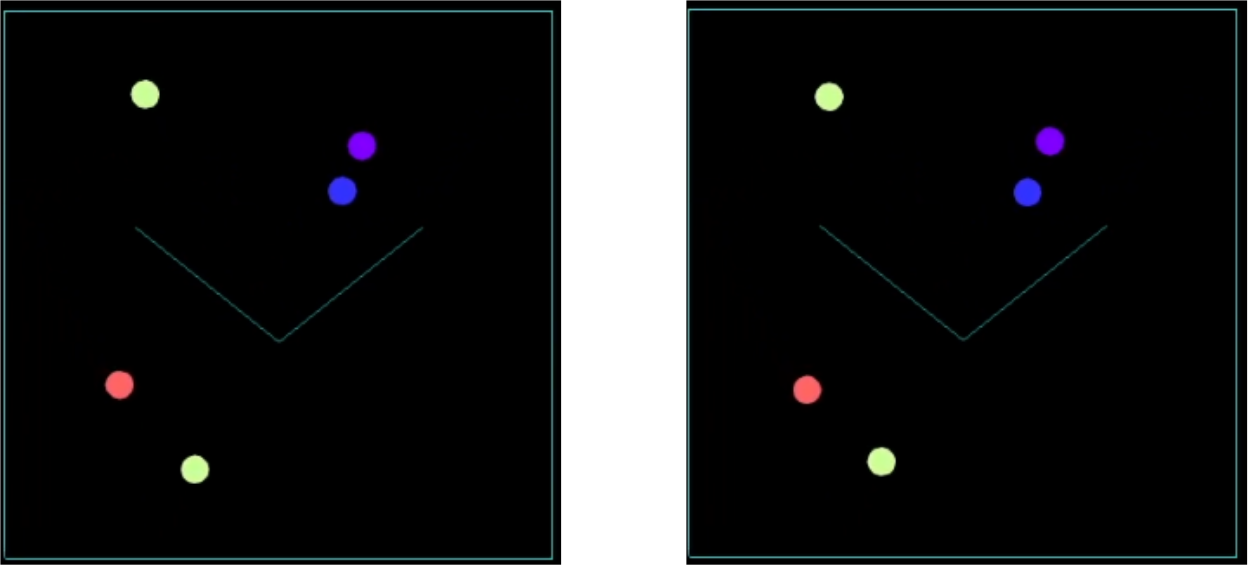
(a) Before (b) after

Figure 3: Collision between circle and circle

This test runs without any errors. The circle collides with each other, and the resolution also works.

10

3) Testing 3: Collision between 5 circles



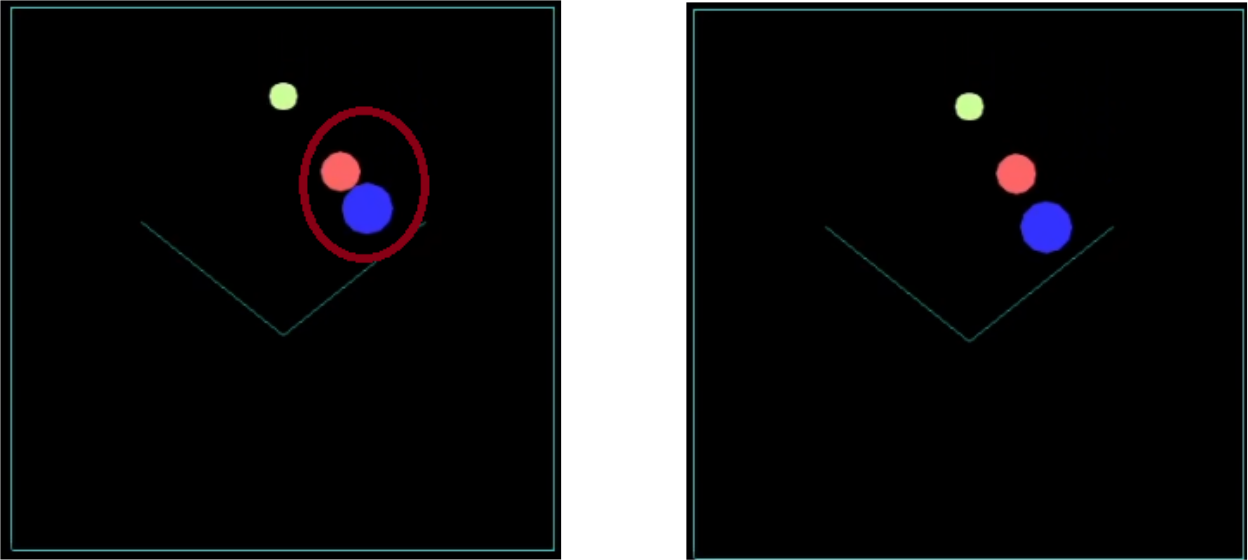
(a) Before (b) after

Figure 4: Collision between 5 circles

This test runs without any errors. The circles collide with each other and the resolution also works.

11

4) Testing 4: Collision between 3 circle with different radius



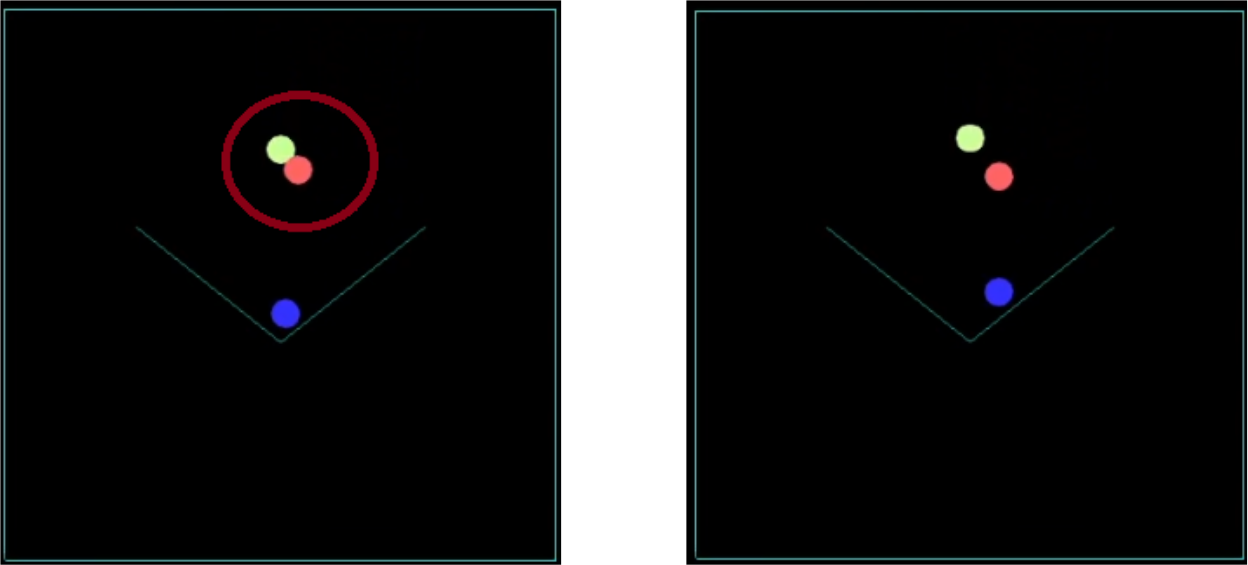
(a) Before (b) after

Figure 5: Collision between 3 circles with different radius

This test runs without any errors. Above Screenshot the blue circle radius is higher than the orange ones because blue has more radius than the orange one blue will have less change to the velocity than the orange one.

12

5) Testing 5: Collision between 3 circles with different mass



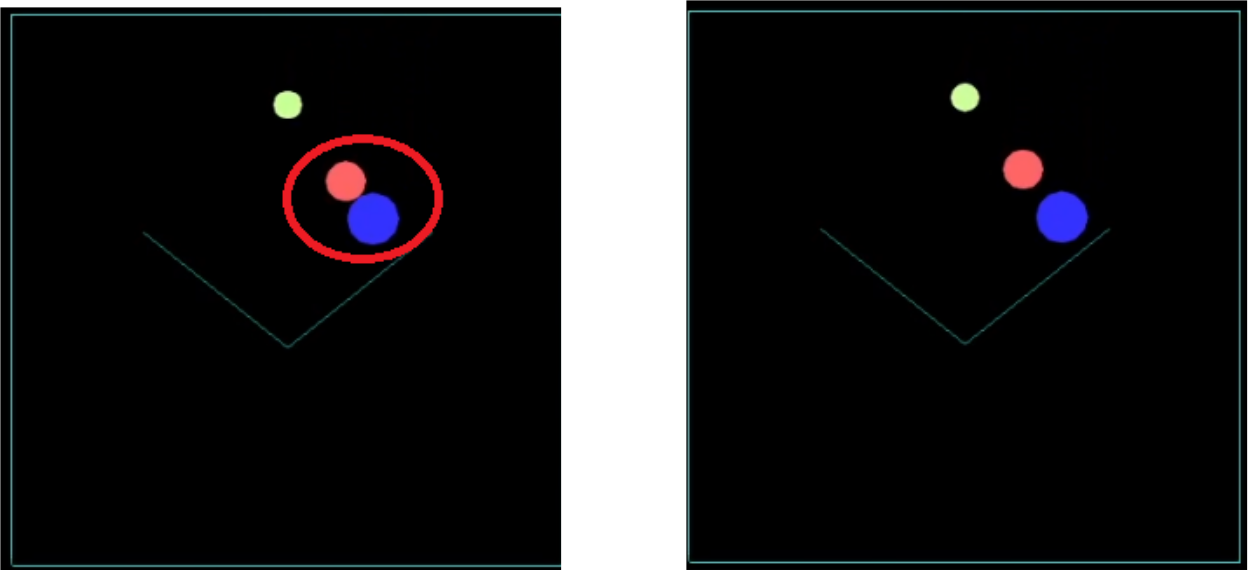
(a) Before (b) after

Figure 6: Collision between 3 circles with different mass

This test runs without any errors. Above Screenshot the orange has larger mass than the green one after the collsiion orange’s velocity was less than the green one due to mass. Orange moved slightly but the green travel far more distance than orange.

13

6) Testing 6: Collision between 3 circles with different mass and raidus



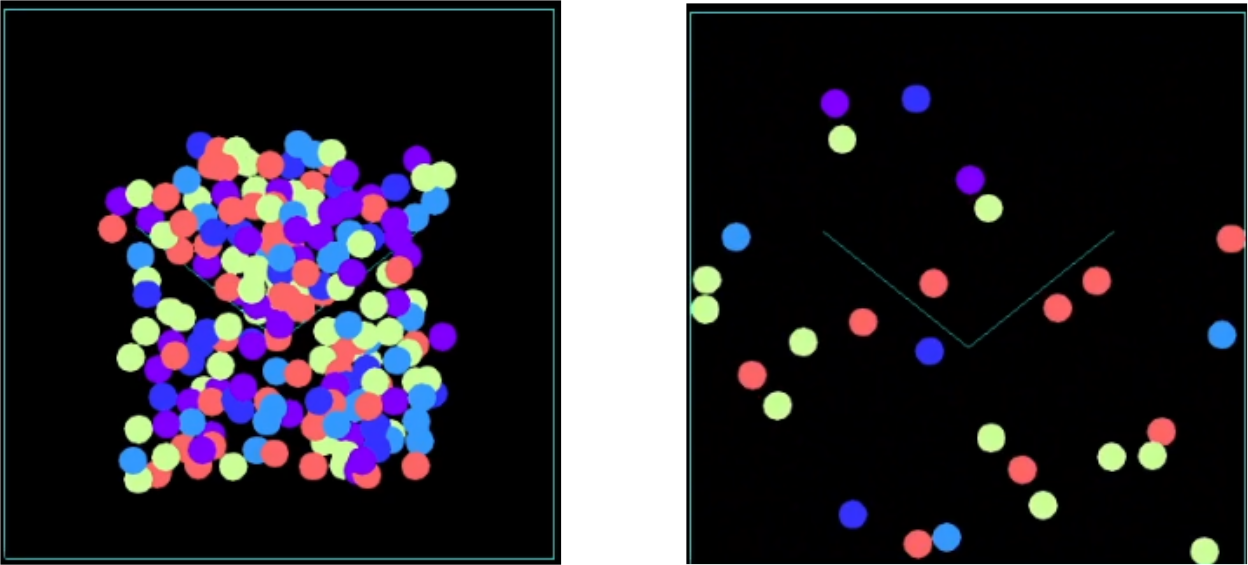
(a) Before (b) after

Figure 7: Collision between 3 circles with different mass and raidus

The orange circle has less radius and mass than the blue circle. Upon collision, the orange circle has been affected more by the collision than the blue circle. It has moved more distance than the blue circle.

14

7) Testing 7: Increasing the circle number to observe the difference



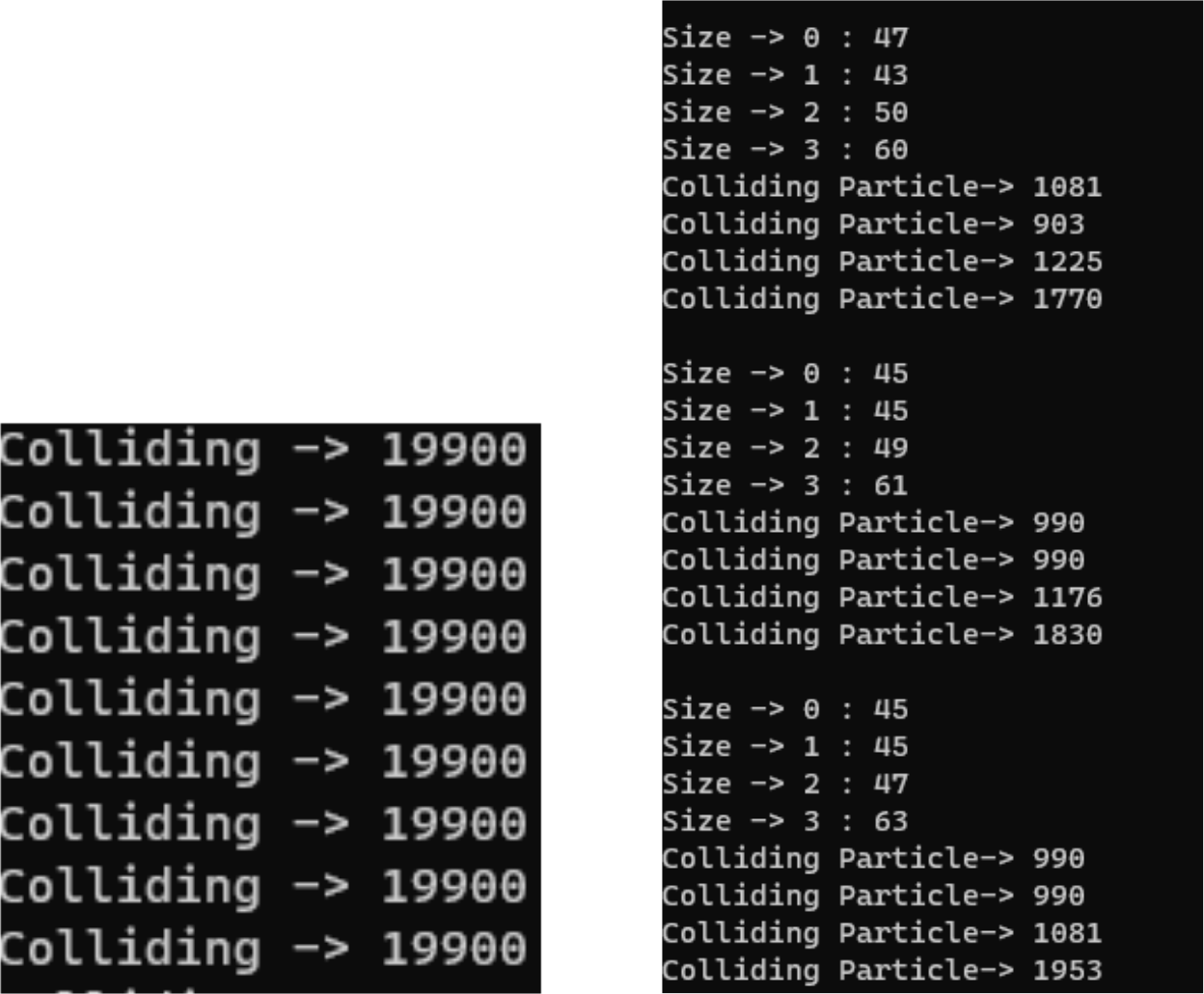
(a) Before (b) after

Figure 8: Increasing the circle number to observe the difference.

This test runs without any errors. After spawning all of the circles start to get out of the box leaving only some of the circles inside. This is due to restitution which helps the circles have more velocity. Due to high velocity the computer cannot compute fast enough to bounce from the platform. There is also some excessive interpenetration as well in certain circles. The cause of it might lie in some minor error on the penetration calculation. Some of the circles overlap fully after the spawning. This might be due to some memory leak, or the pointer has some garbage values as their initial address.

15

8) Efficiency testing: between Brute force vs Spatial partitioning



(a) Brute Force (b) Spatial partitioning

Figure 9: Efficiency Test Between Spatial partitioning vs Brute Force

In the above screenshot we have brute force total number of collision check between al the circles and on the right side we have spatial partitioning. if we combine the spatial partitioning collision check

1081 + 903 + 1225 + 1770 = 4979

990 + 990 + 1176 + 1830 = 4986

990 + 990 + 1081 + 1953 = 5014

Average number of checks for spatial partitioning is around 5000 where as in brute force 19900 is for all the cases. Which clearly states that brute force isn’t efficient enough for particle collision for higher numbers. Because it can’t correctly calculate the total number of collision in time.

16

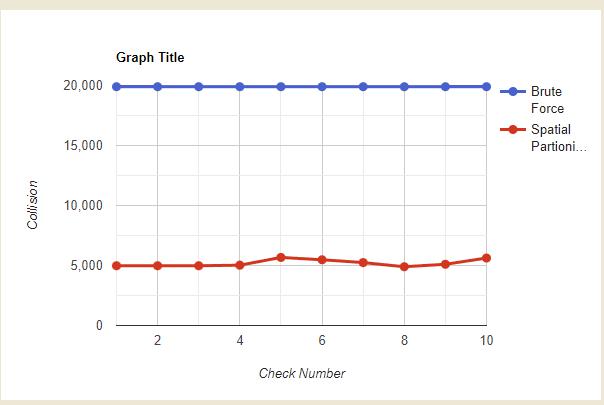


Figure 10: Collision check for 200 circles for brute force and spatial partitioning

Graph’s x-axis shows check No and y axis shows total number of collision check for each checkNo. Red Line is shows the data Spatial partitioning and the blue shows the data of brute force. The graph clearly shows that spatial partitioning checks less collision than brute force. Ultimately decreasing the computing time.

17

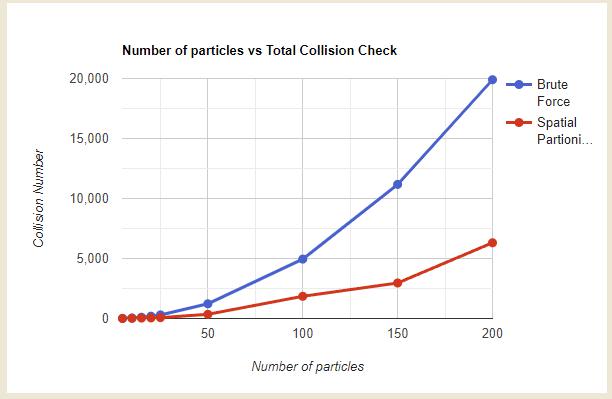


Figure 11: Number of particles vs Total Collision Check

From the above graph we can see that if we increase the number of particles the total number of collision checks increase significantly for brute force. This graph also proves that brute force is not efficient for larger number of particles because collision detection and resolution cannot be computed within that time.

Conclusion

During our exploration of collision detection and resolution, I found these methods are important in gaming, engineering, and robotics. I tested different collisions between objects of all sorts of mass, sizes, and speed. The methods I tried worked well in sorting out these collisions. I compared two ways of detecting collisions: one called brute force, and another called spatial partitioning. Spatial partitioning was way faster, especially when I had lots of objects. Even though spatial partitioning was good in many cases, I saw it needs to be more flexible. So, I think there’s room to make it better for all kinds of situations. In the end, I learned that collision detection and resolution are useful in many situations. But to make it even better, I need to work on improving spatial partitioning to handle all sorts of shapes and sizes. This will help us do collision detection work even better in lots of different areas.

18

References

Ericson, C. (2005) Real-Time Collision Detection. San Francisco : Morgan Kaufmann

Gilbert, E., Johnson, D. and Keerthi, S. (1988). ‘A fast procedure for computing the distance between complex objects in three-dimensional space’, IEEE Journal on Robotics and Automation, 4(2), pp. 193-203.

Gottschalk, S., Lin, M. and Manocha D. (1996) ‘OBBTree: A Hierarchical Structure for Rapid

Interference Detection’, SIGGRAPH ’96: 23rd Annual Conference on Computer Graphics and

Interactive Techniques, New Orleans, Louisiana, 4-9 August. New York: ACM [Online]. Available

at: http://gamma.cs.unc.edu/SSV/obb.pdf (Accessed: 12 January 2018).

Wikipedia. (2023). *Minkowski distance*. [online] Available at: https://en.wikipedia.org/wiki/Minkowski\_distance#:~:text=The%20Minkowski%20distance%20or%20Minkowski [Accessed 16 Nov. 2023].

19