Solution to anti-colliding drone bees in a near future

Agustin Restrepo Cadavid
Universidad EAFIT
Colombia
arestrepoc@gmail.com

Sebastian Gonzalez Arango
University EAFIT
Colombia
sgonzalez1@eafit.edu.co

Mauricio Toro
Universidad Eafit
Colombia
mtorobe@eafit.edu.co

ABSTRACT

Bees dying overtime is a matter of much importance because as the population of bees decrease over time, the flowers are going to have difficulty with pollinisation. If it is impossible to stop the bee population from going extinct, one of the possible solutions, is to develop a robot that can replace the job of the bees. If technology is advanced enough to create drones that are small and capable, then collisions between the bees must be addressed. If two bees are in the same area they might collide, so it is necessary and important to create an algorithm that detects when two bees are 100 meters or closer from each other. A quadtree data structure is an algorithm capable of eradicating this problem in a near future due to its ingenious way of dealing with many objects in a small space. Due to the massive amount of data that would be out there, an efficient algorithm would need to take care of it and this is why quadtree's are so important. After being able to demonstrate how this data structure surpasses a normal algorithm, there's hope towards what might sadly happen in a near future.

To write the abstract, you should answer the following questions in a paragraph: What is the problem? Why is the problem important? Which are the related problems?

Keywords
AABB tree
Quatree
Spatial Hashing
Dynamic AABB tree
Node

__**ACM CLASSIFICATION Keywords**__

1. INTRODUCTION

The decrease of bees in the environment is a growing problem for agriculture. This is because bees are needed to pollinate many of the plant species. If there are no bees, the plants do not make seeds and can not reproduce. Bees are dying because of pesticides that are put in the plants they pollinate, climate change, and other human made problems. A solution to this problem is to repopulate the bees, but at the rate they are dying, it is important to look for new solutions in case humans can not stop the bees from going extinct. Another solution is to create robots to substitute bees and carry out the process of pollination. The robots have to be small enough and smart enough to carry this out, but a database is needed to keep them organized.

2. PROBLEM

When working with these robots it is clear that a main problem, is to keep them organized. They must not be in the same area pollinating the same plants. An algorithm must be set to solve the problem of bees colliding. It must be efficient and work with a large quantity of bees.

3. RELATED WORK

3.1 AABB Collision for objects in 2D.

When a program has two objects that can be bounded by a box, and can not collide, it is important to create an algorithm that determines if they are colliding. AABB collision, or Axis-Aligned Bounding Box, is an algorithm that prevents two objects, or more specifically, two bounding boxes around an object, from colliding. The algorithm works by checking the edges of boxes, and determining if they are before the opposite edge and after the same edge of another box. For example, if two boxes are on a 2d plane, and the left edge of a box is before the right edge of another and after the left edge, and the same happens in the top edge and bottom edge, the boxes are colliding. It is important to know that the boxes can not rotate. The algorithm will be given the coordinate of the top right corner and bottom left corner of the boxes and the width and height. With this, it determines if any of the sides are overlapping and returns a true or false.

3.2 Large amount of Objects

When collision detection is needed for a large amount of objects, the time taken by the computer can increase dramatically as the amount of objects can be huge. This becomes a problem when a lot of objects have to be analysed for collision. Quadtree fixes this, by creating small subsections of the 2d plane, in which more than 1 object are located, then it checks these objects for collision with each other and only with each other, instead of checking for collision with all the other objects.

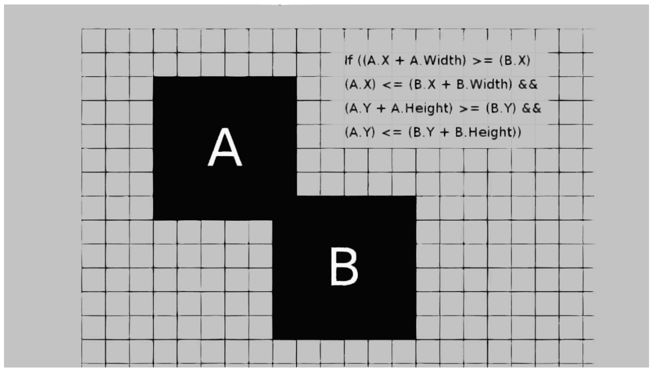
3.3 Time taken to calculate collisions

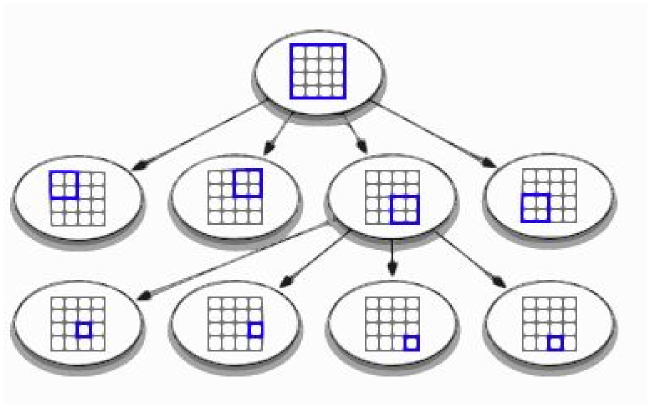
Testing out a high amount of objects during a collision test can be time demanding and it would most likely generate more problems. Spatial hashing fixes all of these problems as it can optimize the objects used in the collision test and the time used. A spatial hash is a dimensional extension of a hash table and what this does is it can store data that's used inside a function to store its value in some sort of buckets, and you use the references of these to find the different information used.

3.4 Memory to calculate and coordinate objects

During the tests done, a lot of memory should be used as this problem should be seen as a possible world change. Whats meant by this; to calculate all of the possible outcomes throughout the world, a lot of memory and computing power would be needed. A way to fix this problem with an easy solution would be to optimize all of the algorithms in order to make the whole process work in harmony. Other than the plausible solutions mentioned before, there really aren't more approaches to make this work out as a lot of study has been done for this research by other people. but dynamic AABB trees could also help out by inserting information, removing it and updating it to get important data about the collisions.

Graphs:


If ((A.X + A.Width) >= (B.X)
(A.X) <= (B.X + B.Width) &&
(A.Y + A.Height) >= (B.Y) &&
(A.Y) <= (B.Y + B.Height))
A
B



4. Title of the first data structure designed

Design a data structure to solve the problem and make a figure explaining it. Do not use figures from the Internet.

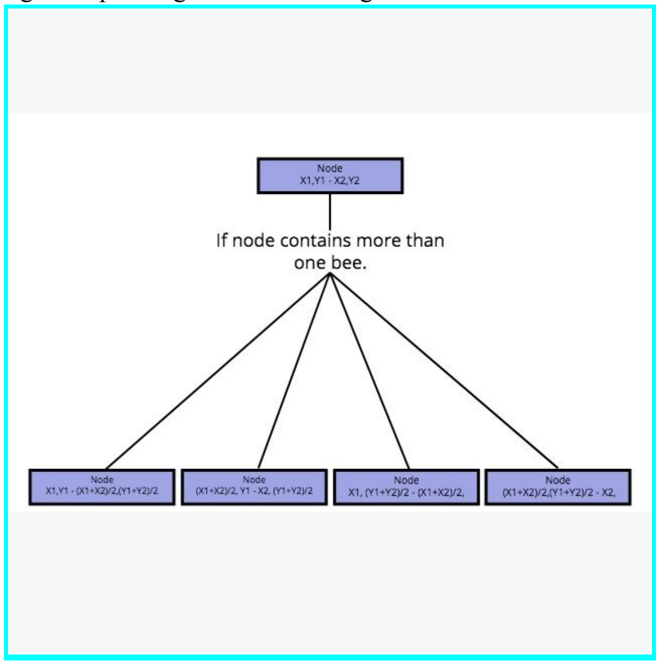

Node
X1,Y1 - X2,Y2
If node contains more than one bee.
Node
X1,Y1 - (X1+X2)/2,(Y1+Y2)/2
Node
(X1+X2)/2, Y1 - X2, (Y1+Y2)/2
Node
X1, (Y1+Y2)/2 - (X1+X2)/2,
Node
(X1+X2)/2,(Y1+Y2)/2 - X2,

Figure 1: Nodes split into four separate blocks when encountered with more than 1 bee in 1 block

4.1 Operations of the data structure

A Quadtree Data structure is appropriate to solve this problem because it will drastically reduce the amount of operations the computer must run.

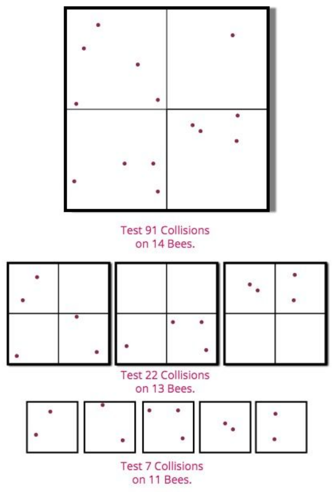


Figure 2: The methods used in this data structure are add, search and delete

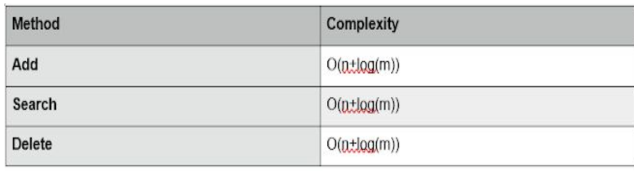
4.2 Design criteria of the data structure

This data structure is very useful for this problem because it is a problem that has to compare many different data points.

The amount of time it takes to analyze all points grow exponentially as points are added because each new point has to be compared to all the other points. The Quad-tree algorithm not only marks some points are un necessary to compare, but allows the computer to only analyze likely collisions. The main reason why one would use this algorithm, is to optimize time.

4.3 Complexity analysis

Derive the complexity of each operation of the data structure for the worst case and best case, As an example, this is a way to report the complexity analysis:



Method	Complexity
Add	O(n+log(m))
Search	O(n+log(m))
Delete	O(n+log(m))

Table 1: Number of data points = n

With of Quad-Tree = m

Even though the complexity does not seem to different from one of a data structure without a quad tree, it is a lot better. This is because the complexity is calculated with the worst case scenario in mind, in which the complexity would be very similar, but in most cases the complexity when using the Quad-Tree is a lot more efficient.

4.4 Execution time

Measure (I) execution time and (II) memory used by the operations of the data structure, for the data set found in the .ZIP file.

Measure the execution time and memory used 100 for each data set and for each operation of the data structure. Report the average values.

Dataset with 4 bees	Dataset with 10 bees	Dataset with 100 bees	Dataset with 1,000 bees	Dataset with 10,000 bees	Dataset with 100,000 bees	Dataset with 1,000,000 bees
The average time taken in 100 trials was: 0 ms	The average time taken in 100 trials was: 0 ms	The average time taken in 100 trials was: 0 ms	The average time taken in 100 trials was: 1 ms	The average time taken in 100 trials was: 61 ms	The average time taken in 100 trials was: 6046 ms	The average time taken in 100 trials was: 43003 ms

Table 2: Execution time of the operations of the data structure for each data set.

4.5 Memory used

Report the memory used for each data set

Dataset with 4 bees	Dataset with 10 bees	Dataset with 100 bees	Dataset with 1,000 bees	Dataset with 10,000 bees	Dataset with 100,000 bees	Dataset with 1,000,000 bees
The average memory taken in 100 trials was: 0 mb	The average memory taken in 100 trials was: 0 mb	The average memory taken in 100 trials was: 0 mb	The average memory taken in 100 trials was: 0.0571 mb	The average memory taken in 100 trials was: 2.871457 mb	The average memory taken in 100 trials was: 322.610221 mb	The average memory taken in 100 trials was: 930.086912 mb

Table 3: Memory used for each operation of the data structure and for each data set data sets.

4.6 Result analysis

Explain the results obtained. As an example, compare different implementation of the data structure and report the comparison in a table or graph.

5. Title of the last data structure designed

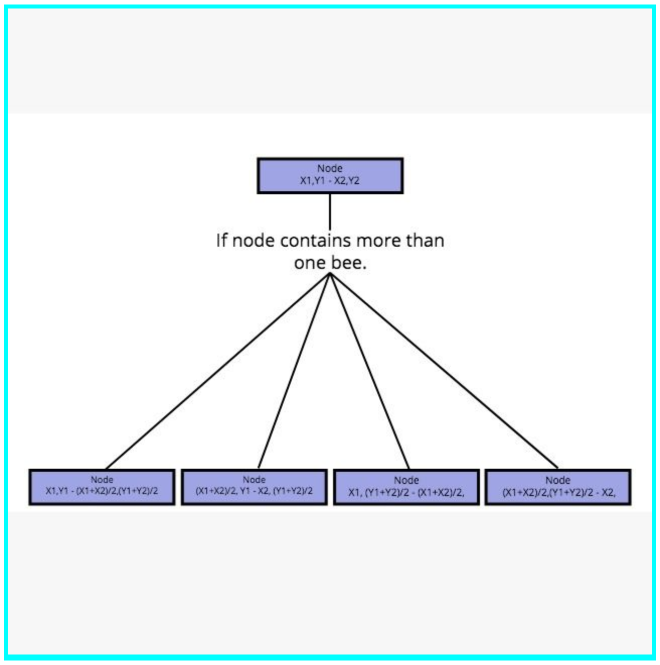

Node
X1,Y1 - X2,Y2
If node contains more than one bee.
Node
X1,Y1 - (X1+X2)/2,(Y1+Y2)/2
Node
(X1+X2)/2, Y1 - X2, (Y1+Y2)/2
Node
X1, (Y1+Y2)/2 - (X1+X2)/2,
Node
(X1+X2)/2,(Y1+Y2)/2 - X2,

Figure 1: Linked List of bees. Bee is a class that contains a position x and y. Nodes split into four separate blocks when encountered with more than 1 bee in 1 block.

5.1 Operations of the data structure

A Quadtree Data structure is appropriate to solve this problem because it will drastically reduce the amount of operations the computer must run.

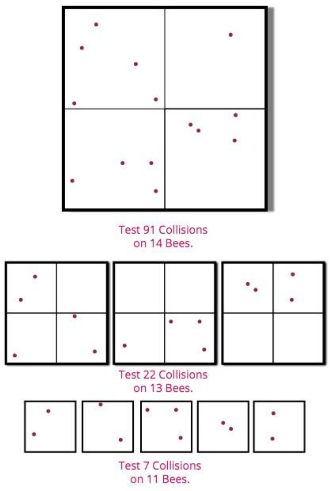

Test 91 Collisions on 14 Bees.
Test 22 Collisions on 13 Bees.
Test 7 Collisions on 11 Bees.

Figure 2: The methods used in this data structure are add, search, delete, size and auxsize

5.2 Design criteria of the data structure

This data structure is very useful for this problem because it is a problem that has to compare many different data points. The amount of time it takes to analyze all points grow exponentially as points are added because each new point has to be compared to all the other points. The Quad-tree algorithm not only marks some points are un necessary to compare, but allows the computer to only analyze likely collisions. The main reason why one would use this algorithm, is to optimize time.

5.3 Complexity analysis

Method	**Complexity**
Add	**O (n + log(m))**
Search	**O (n + log(m))**
Destroy	**O (n + log(m))**
Size	**O (n + log(m))**
AUXSize	**O (n + log(m))**

Table 5: Table to report complexity analysis; the number of data = n and the width of the Quad-Tree = m.

5.4 Execution time

Here are the average measurements for the time the program took to be executed with seven different data-sets that were ran one hundred times each. The time taken to execute the program taken with two different approaches, one using the QuadTree data structure and one without it.

Dataset with 4 bees	Dataset with 10 bees	Dataset with 100 bees	Dataset with 1,000 bees	Dataset with 10,000 bees	Dataset with 100,000 bees	Dataset with 1,000,000 bees
The average time taken in 100 trials was: 0 ms	The average time taken in 100 trials was: 0 ms	The average time taken in 100 trials was: 0 ms	The average time taken in 100 trials was: 1 ms	The average time taken in 100 trials was: 61 ms	The average time taken in 100 trials was: 6046 ms	The average time taken in 100 trials was: 43003 ms

Table 6: Execution time of the operations of the data structure for each data set in milliseconds.

Dataset with 4 bees	Dataset with 10 bees	Dataset with 100 bees	Dataset with 1,000 bees	Dataset with 10,000 bees	Dataset with 100,000 bees	Dataset with 1,000,000 bees
The average time taken in 100 trials was: 0 ms	The average time taken in 100 trials was: 0 ms	The average time taken in 100 trials was: 0 ms	The average time taken in 100 trials was: 3 ms	The average time taken in 100 trials was: 319 ms	The average time taken in 100 trials was: 43797 ms	The average time taken in 100 trials was: 142341 ms

Table 7: Execution time of the operations without the data structure for each data set in milliseconds.

5.5 Memory used

Here are the average measurements for the memory the program used while being executed with seven different data-sets that were ran one hundred times each. The memory usage was taken with two different approaches, one using the QuadTree data structure and one without it.

Dataset with 4 bees	Dataset with 10 bees	Dataset with 100 bees	Dataset with 1,000 bees	Dataset with 10,000 bees	Dataset with 100,000 bees	Dataset with 1,000,000 bees
The	The	The	The	The	The	The average

average memory taken in 100 trials was: 0 mb	average memory taken in 100 trials was: 0 mb	average memory taken in 100 trials was: 0 mb	average memory taken in 100 trials was: 0.0571 mb	average memory taken in 100 trials was: 2.871457 mb	average memory taken in 100 trials was: 322.610221 mb	memory taken in 100 trials was: 930.086912 mb

Table 8: Memory used for each operation of the data structure and for each data set data sets in megabytes.

Dataset with 4 bees	Dataset with 10 bees	Dataset with 100 bees	Dataset with 1,000 bees	Dataset with 10,000 bees	Dataset with 100,000 bees	Dataset with 1,000,000 bees
The average memory taken in 100 trials was: 0 mb	The average memory taken in 100 trials was: 0 mb	The average memory taken in 100 trials was: 0 mb	The average memory taken in 100 trials was: 0.02679 mb	The average memory taken in 100 trials was: 2.599234 mb	The average memory taken in 100 trials was: 282.002401 mb	The average memory taken in 100 trials was:477.104128 mb

Table 9: Memory used for each operation without the data structure and for each data set data sets in megabytes.

5.6 Result analysis

The results gathered after testing out the data sets show a gap (between a well known data structure such as a quadtree and one that is not) in terms of both memory usage and time efficiency. There most likely has to be a balance between these two important factors. instead of showing a table with its comparisons; figure six through nine clearly demonstrate how the quadtree outruns the non quadtree in performance, but at the same time losses in terms of memory usage. The best example to highlight this, is the data set with one million bees; while the quadtree averages an approximate run time that is three times faster than the non quadtree, it uses up to two times more memory than the non quadtree.

6. CONCLUSIONS

During this project, a lot of valuable lessons were learned, most importantly, the knowledge and insight that was sought after in regards to data structures was acquired. Even though there were many obstacles, the first two deliveries made it easier to understand and project how the final product would end up being. The goal of having a comparison between a data structure and a non data structure to measure how different they really were was achieved. The most important issued of the report was pretty much the last one, due to the fact that valuable lessons were learned from previous mistakes, and the satisfaction of having a completely functional piece is worth everything.

The most important results obtained during this project were pretty much the comparisons to a non data structure. Finally cleaning the code and having hand to hand comparisons between the two collision tests show that using a quadtree data structure indeed saves up time and is more efficient than a normal algorithm. Once the results were taken into tables to be compared, a sense of balance between memory usage and time taken to run is taken into account because optimizing both is pretty difficult. In the case of a quadtree data structure, there's a toll in regards to memory in order to fasten up the process the algorithm takes to run. At the end, the result that was being expected showed up, as the idea of using a quadtree to solve this problem came back with the information that was previously found.

The solution we thought of in the beginning is pretty much what we went forward with and it fortunately ended up working out. There were many problems that were successfully taken care of, such as the ways to take coordinates from the file into the bee arraylist, testing out all of the different datasets one hundred times each in an efficient way, understanding and taking into account that sometimes the memory can be negative due to the trash collector, and many other mistakes/difficulties that let us learn a lot.

6.1 Future work

In the future, the only thing that we would like to improve about this data structure is finding some methods to be more efficient memory-wise, and optimizing lines of code to make it a more appealing and maybe even faster code in terms of running speed. The most fun thing would be learning how we could apply this data structure to any other everyday problem in the near future and if the possibility was available, to actually implement it on future robotic bee drones.

ACKNOWLEDGEMENTS

Agustin Restrepo Cadavid (AKA the god of programming) for giving his very best to accomplish a perfect and clean code

REFERENCES

1. AABB. (2018, April 14). Retrieved April 16, 2018, from https://en.wikipedia.org/wiki/AABB

2. Quad Tree. (2018, February 07). Retrieved April 16, 2018, from https://www.geeksforgeeks.org/quad-tree/
3. Spatial hashing implementation for fast 2D collisions. (2009, June 13). Retrieved April 16, 2018, from https://conkerjo.wordpress.com/2009/06/13/spatial-hashing-implementation-for-fast-2d-collisions/
4. Dynamic AABB Tree. (2013, October 21). Retrieved April 16, 2018, from http://www.randygaul.net/2013/08/06/dynamic-aabb-tree/
5. Myopic Rhino. (2009, October 01). Spatial Hashing. Retrieved April 16, 2018, from https://www.gamedev.net/articles/programming/general-and-gameplay-programming/spatial-hashing-r2697/
6. J. (n.d.). Introductory Guide to AABB Tree Collision Detection. Retrieved April 16, 2018, from http://www.azurefromthetrenches.com/introductory-guide-to-aabb-tree-
7. Bhati, A. (2016, June 20). How to calculate memory usage of java program? Retrieved from https://stackoverflow.com/questions/37916136/how-to-calculate-memory-usage-of-java-program?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa