

Computer Games Development

Project Report

Year IV

**Mantas Zalnierius**

**C00241626**

**28/04/2022**

# Declaration

# This examination/assessment will be submitted using Turnitin as the online submission tool. By submitting my examination/assessment to Turnitin, I am declaring that this examination/assessment is my own work. I understand that I may be required to orally defend any of my answers, to the lecturer, at a given time after the examination/assessment has been completed, as outlined in the student regulations.

**Contents**

Acknowledgements 3

Project Abstract 4

Project Introduction and/or Research Question 5-6

Literature Review 7-10

Results 11-18

Project Milestones 19-21

Major Technical Achievements 22

Project Review 23-25

Conclusions 26

Future Work 27

References 28

Appendices 29-35

**Acknowledgements**

Major Technical Achievements

The first person I would like to acknowledge is my project supervisor, Jason Barron. Jason helped me on keeping on track on my project and giving me advice on how to develop my documents and demos.

The next group of people I would like to acknowledge is all the other supervisors, Noel O’ Hara, Lei Shi, Martin Harrigan and Oisin Cawley, for keeping me constructive feedback on my project throughout the year that helped me develop the project I have right now.

Another person I would like to thank is Anthony D’Angelo for providing an incredibly detailed paper on Quad-Trees that helped me immensely on my development of the quad-tree I have developed in my project. **[1]**

The last group of people I would like to acknowledge are, Erin J. Hastings, Jaruwan Mesit and Ratan K. Guha who together created a paper that helped me in understanding what spatial hashing is and how to go about developing a spatial hashing grid in my project.**[2]**

**Project Abstract**

This research project is about researching into different spatial partitioning algorithms that will reduce collision checks and increase frame rate for machines in a 2D plane. This research project will also look at the data and evaluate what type of approach works better in different situations.

The goal I have for this research project is to implement 2 different spatial partitioning algorithms. A Quad-Tree and Spatial Hashing algorithm. With these algorithms I want to collect data and see what the impact that has on performance on machines when using them on in scenes that have hundreds of objects. This data will be as follows, the number of collisions occurring per frame, the current frame rate, the number of objects within the scene and the type of grid the user has active. I will also implement a debug view for these different algorithms, so that anybody can see what the distinct types of grids look like when running my project.

At the end of my research project, what I will want to have enough data to display the major improvements spatial partitioning provide overusing no spatial partitioning algorithms.

# Project Introduction and/or Research Question

In a traditional 2D game, there are lot of objects that must interact with each other in a multitude of ways. One of these ways is colliding off each other. Collision response (By this I mean what happens when object is colliding with another object.) does not have the biggest impact on machines. What does have the biggest impact is how to know when to check for collisions. A naive approach would be to just check every object against each other. This is N^2 in logarithmic time, which is exponential. Which leads to problems of performance because the more objects you add in the more time it will take to check all those collisions. With enough objects in the scene this can lead to slow down on user’s machines which turn make the game un-immersive (The user finds it unenjoyable to play because of the slow down). Spatial partitioning algorithms is way of solving this problem. With these algorithms, it could lead to scenarios where the performance impact of these collision would reduce or even completely be removed entirely. Which in turn would lead to better immersive gameplay for the user.

In this research project I will research into two algorithms, one is a Quad-Tree algorithm. In 2016 **Antony D’ Angleo** went into depth in his paper about what the different type of quad-tree structures there are. Quad-Tress is a spatial hierarchy tree. In his paper he talks about “Space-Filling Quad-Trees” and “Region Quad-Tress” For this research project, I will investigate Region Quad-Trees. Region.

The other algorithm I will investigate is the Spatial Hashing algorithm. In the paper written by **Erin J. Hastings, Jaruwan Mesit and Ratan K. Guha.** They go into detail into spatial hashing and why it is necessary in large simulations regarding collision response, AI, and rendering. I will investigate spatial hashing regarding collision response. Spatial Hashing is where you take objects in a 2D or 3D simulation/game which are indexed into a hash map. In theory this will make it extremely fast to get objects because in hash maps retrieval of data is 0(1), which is constant.

**Questions to answer**

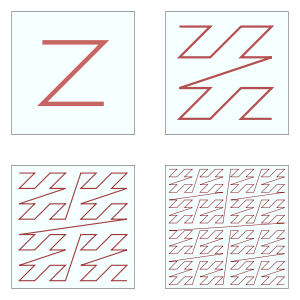
These following research Questions will be answered using my local machine which have the following specifications (My Machine Specs are 32 Gb’s of ram and has a 3.9 Ghz processor)

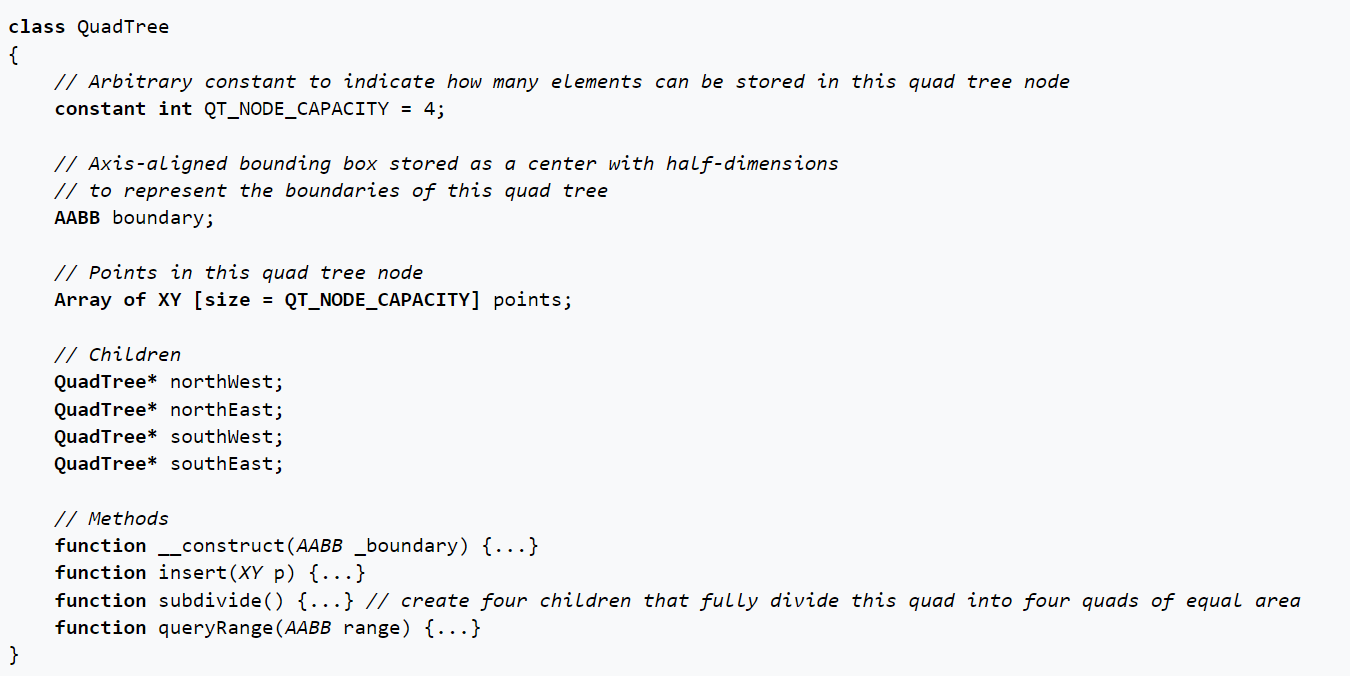
* How much of an impact does 100 Objects checking collisions have on performance on my Machine Without using any Spatial partitioning algorithms?
* How much of an improvement does Quad-Tress have on performance using 100 Objects?
* How much of an improvement does Spatial Hashing have on performance using 100 Objects?

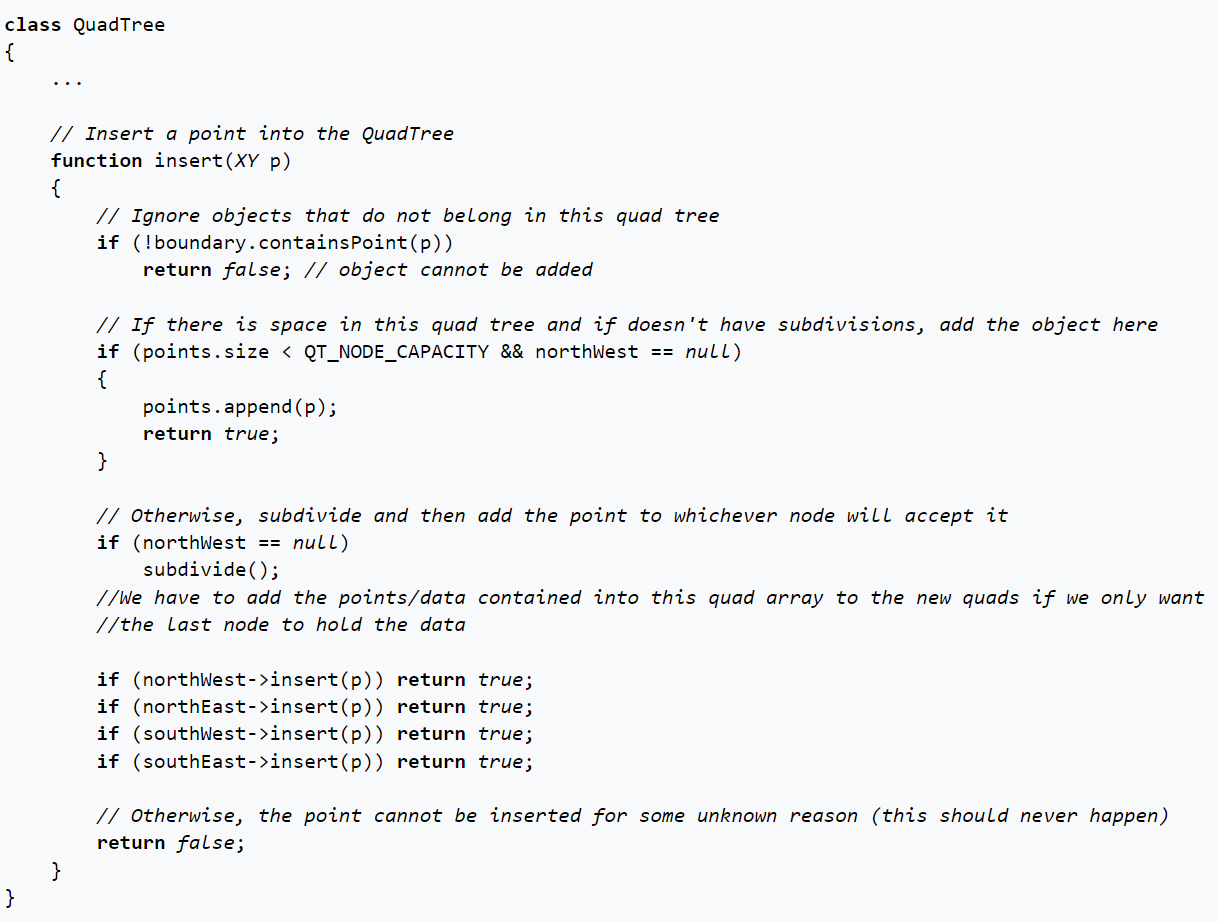
# Literature Review

In this paper [**[1]**](#_[1]._Cheng_Liang)it details the different type of Quad-Tress there are and what the use cases are for these Quad-Tress. In this paper it describes how quad-trees can be used in any dimension and that quad-trees are always a recursive decomposition of space that allows of storage of important data that is relevant to any application. So, for example, in a game. The important data would be the Objects within the game and which Node they are in within the Quad-Tree. This can be extremely useful as you can retrieve Objects that you want to handle collision between other Objects that are in the same Node. The paper also goes into detail about distinct types of quad-trees one of these is the “Space-Filling Curves” quad-tree. The paper talks about how a space-filling curve is a tree where the range is every point in the space you provide (The root node of the tree). In a game, the root node could be the actual window. The paper mentions the z-order curve or the Morton space filling curve. A z-order curve is a way of mapping 2d or 3d data into 1d containers. The paper uses an example with 2 by 2 sqaure and uses cardinal coordinates in this order South-West, North-West, South-East and North-East. You may find an example of this z-order curve on **figure 1.1.** [**[1]**](#_[1]._Cheng_Liang)

The paper also goes into detail about another type of quad-tree. This Quad-Tree is called Region Trees. [**[1]**](#_[1]._Cheng_Liang) Region Trees recursively subdivides the root node and other internal nodes into smaller squares. Also, the data that is stored in leaf nodes in a Region Tree contain data such as the RGBA value of a pixel on an image. Region Trees is most used in image compression. After researching into Space Filling Trees and Region Tress. I have decided that Space Filling Trees be the quad-tree that I will use for my research project. As my research project is about collision handling and Space Filling Trees handle this problem. In terms of logarithmic time, this quad-tree approach is in the time space of O (Log N) This is much better than the naive approach of handling collisions which is 0(N^2). You can find a pseudocode implementation of a Quad-Tree on **Figure 1.2.** [**[1]**](#_[1]._Cheng_Liang)

**Figure 1.1 displaying what a z-order curve is.** [**[1]**](#_[1]._Cheng_Liang)



**Figure 1.2 highlights pseudocode on how to implement a Quad-Tree**

[**Wiki**](https://en.wikipedia.org/wiki/Quadtree) **This wiki page is where I got the pseudocode for the implementation of a quad-tree.**

In this paper [**[2]**](#_[2]._Steve_Swink)it describes how in games/simulations that grow in scale (I.e., Objects are added in enormous amounts), that it becomes clear that there needs to be optimization techniques to handle all that data. The technique that the paper suggest utilizing is the Spatial Hashing technique. Using this technique, it states that objects in games in 2D or 3D, can be indexed into a 1D HashMap. [**[2]**](#_[2]._Steve_Swink) The reason to store this type of data into a HashMap is because of how retrieval of data works in HashMap's. HashMap’s use hash keys to access data directly from the container and this retrieval of data is in a O (1) logarithmic time which means it is constant, which makes it extremely fast to rederive data immediately with no delay. In the paper it says there is three aspects of a Spatial Hashing grid. One is the grid itself. This grid for games could be the window size of the game or the actual camera the player is looking through. The other aspect is the hashing function, this hashing function is a way of getting a hash key to be used with the 1D hash map. [**[2]**](#_[2]._Steve_Swink) This hashing function in a 2D game, can be calculated using the local position of an object and the grid size in the X axis. The last aspect is the actual HashMap, this HashMap will use the hash keys produced to store objects that are in the game. **Figure 1.3** shows my implementation of the hash key function in my application.



**Figure 1.3 highlights the hash Key function I use in my implementation.**

# Results

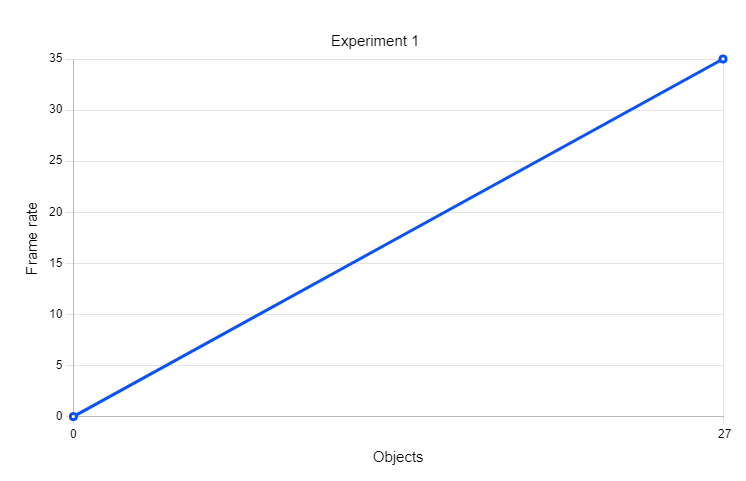
There were 3 experiments that I performed with the implementation I have put forward to display how much of an improvement spatial partitioning algorithms have over the brute-force approach.

The first experiment I did was to see at what object count does the scene start to have impact from all the collision checks. When I was running my experiment, I got to 27 objects, where there was a considerable impact on performance. The frame rate dropped to an average of 30-40 and the number of collisions were 18954 (27 \* 26 \* 27) and this was only for one frame out of 60. For a full second there would be 1137240 collisions checks. This highlights the problems of using brute-force for handling collision checks. Even my machine which is a powerful machine struggled on only 27 objects. When the object count would reach 200, you could see the application just freeze going over all the possible collisions there could be. This experiment was a sucess as it highlights why there is a need for a better solution.

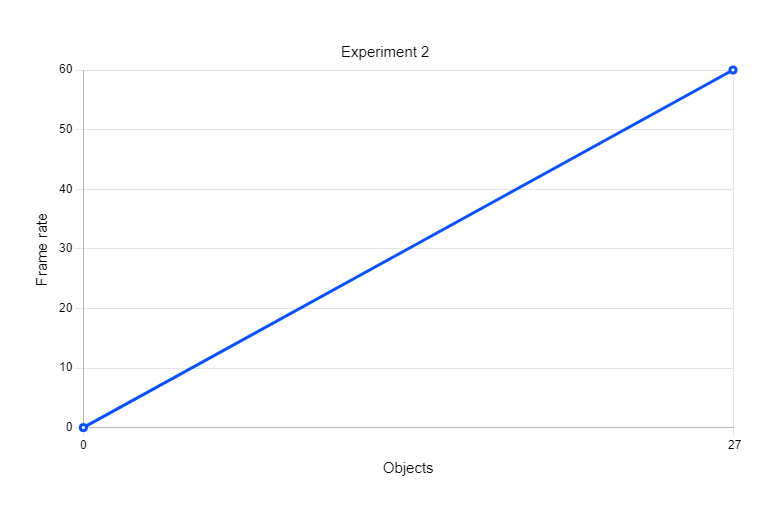
My second experiment I ran with my project was to check if there was any improvement using the same number of objects as the previous experiment but using a Quad-Tree to handle collision checking. With this approach, I saw significant improvement on the frame rate in the game, it stayed at a solid 60 frames per second. Also, the number of collisions that were happening per frame reduced also dramatically to 20-200 per frame. This is an enormous improvement over the brute-force approach.

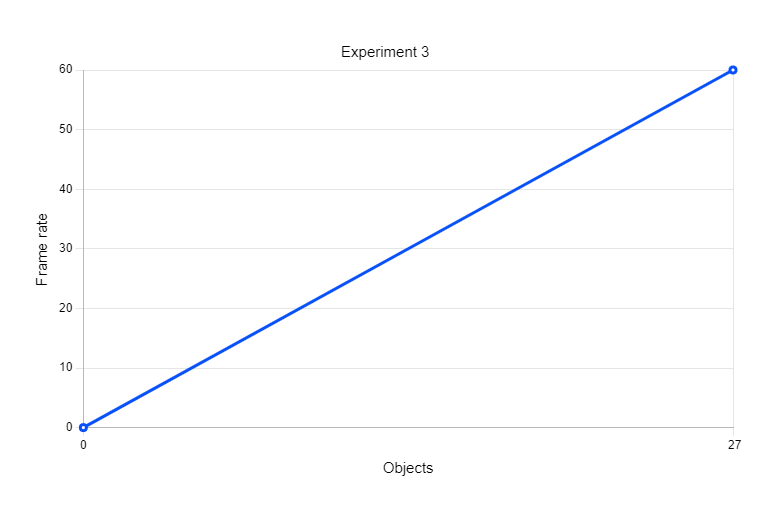
My third experiment that I ran was to use the same number of objects as the previous experiments, but this time use the spatial hashing algorithm to handle collision checking. With using this approach, I saw bigger improvement on collision check. The number of collisions checks dropped from 20-200 to 55-60 from the previous experiment. The frame rate also stayed at steady rate of 60. The reason spatial hashing has an improvement over the Quad-Tree approach, is because of the way I am updating my Quad-Tree in my application. Now I am currently re-constructing the Quad-Tree every frame, which in-turn would lead to slower performance. Also, the spatial hashing algorithm is faster, because it uses a HashMap, because of how fast it is to retrieve data. The Quad-Tree implementation I have does not use any HashMap's which results in longer times to retrieve data.

**Data Comparing Number of Objects in respect to the frame rate of the game using figure 1.4 - figure 1.6.**

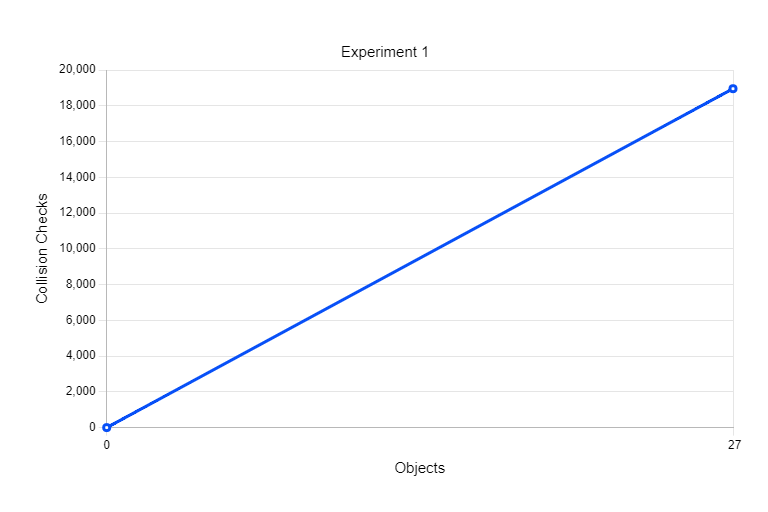


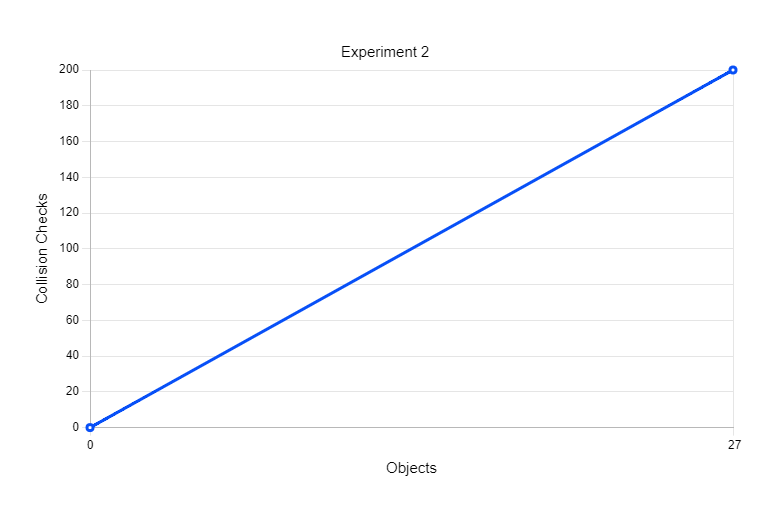
**Figure 1.4**

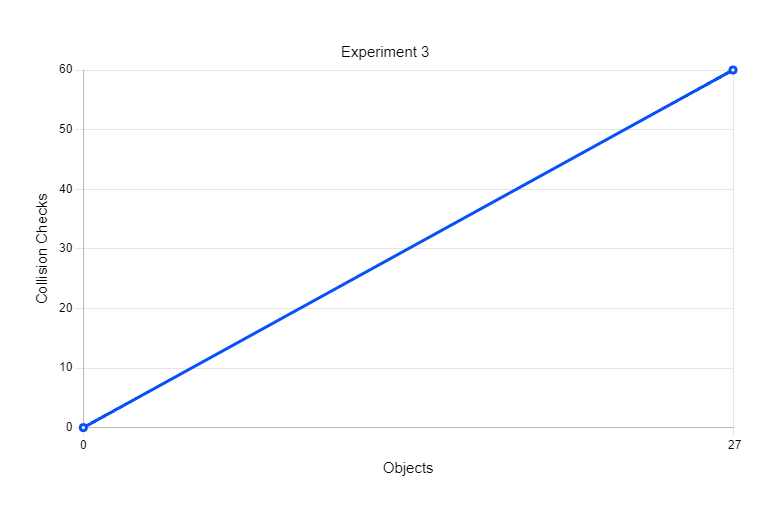
**Figure 1.5**

**Figure 1.6**

**Data Comparing Number of Objects in respect to the number of collisions occurring per frame using figure 1.7 - figure 1.9.**

**Figure 1.7**

**Figure 1.8**

**Figure 1.9**

With all this data collected we can see that spatial partitioning algorithms have a significant improvement on the number of collisions occurring and the frame rate of my machine while these collisions checks are occurring.

**Project Milestones**

**Quad-Tree:**

The first major millstone that I overcame was the implementation of the Quad-Tree. The overall implementation took over a month to implement after the Christmas holidays which were 2-3 weeks over the time I first thought, it would take to implement. The reason I developed this Quad-Tree after Christmas was because before the Christmas holidays, I had implemented features and had to demo them. Some of the supervisors mentioned that it would be best to investigate a hierarchical approach to handling collision which would be better than what I had back then. After this demo I researched into a multitude of different hierarchical structures like this and Spatial Hashing.

The biggest reason for this long development time was more of my understanding on how to implement a tree like a Quad-Tree and use that to handle collisions faster. Also, I found it challenging to incorporate a quad-tree using SFML because I never really knew how to design and implement a tree structure like it and how to render all the different nodes that are subdivided into other squares. The next biggest challenge from this millstone was assigning the different objects within the game to correct nodes within the quad-tree. I came into a lot of issues where objects where incorrectly assigned to wrong node, which took quite a bit of time to debug which slowed my progression a bit.

**Spatial Hashing:**

The second major milestone was the implementation of a spatial hashing grid algorithm. Which took me a lot less time to implement than the Quad-Tree algorithm. This is because in my second year of college, we had some lectures talking about spatial hashing and how you would implement something like that in SFML, which were building blocks for my implementation I have right now. The biggest hurdles I had to overcome from this millstone was fully understanding how hashing works and it can be used to indexed diverse types of data like Objects in my application into a 1D hash table. After I fully understood what hashing was, it took only a short amount of time to implement a solution in my application.

**Swapping between no Grid, Quad-Tree Grid and Spatial Hashing Grid:**

**Th**e last milestone I had to overcome was to implement a way for the user to change the grid during run-time of my application. This millstone was by far the easiest millstone to overcome has a lot of the background work was finished I just had to connect all the puzzles together and work together. The first thing I added was the ability to run the game with no grid. So, with this grid, you would be handling collisions using a brute force method. The next step was the include the Quad-Tree grid and have that be able to be swapped out with No Grid during run-time. This was easy to implement has I was able to store the relevant data easily with the Quad-Tree. I only had to re-update the grid when the user swapped back to the quad-tree.

After this I made sure the user was able to swap between the spatial grid, no grid, and the quad-grid. This took a bit of time to implement has there were cases the application would crash, so I had to find edge cases from these crashes. The last step of this millstone was the ability to view data such as frame rate, number of collisions occurring per frame, the number of objects within the scene and the type of grid the user had active. This was easy to implement and took only a few hours to implement.

# Major Technical Achievements

After finishing my research project, I feel that I have understood how important it is to have a particularly effective way of handling collisions because if you do not, it will lead to situations where your game will not be able to have a vast amount of collidable Objects, as having to many without a proper fast way of handling them will lead to serve performance impacts on users' machines. I also feel great about having a way of displaying in a visual sense how important these types of spatial partitioning algorithms are in today's video game development era. I feel proud. I am proud of researching, understanding, and implementing solutions to Quad-Trees and Spatial Hashing algorithms. Lastly, I feel more excited than ever to research deeper into this topic further cover other types of spatial **part**itioning, as there are other types of Quad-Trees such as Oct-Trees or HyperOctTrees and would love to delve into them.

# Project Review

**What went right?**

I believe my finished research project has everything I wanted to implement from my project proposal. I think the way my project displayed visually the number of collisions occurring per frame, the current frame rate of the game, the number of objects within the game and the current active grid type was very well made. The reason for this is because you can see the difference the quad-tree and the spatial hashing grid has over the no grid, which is what my project is all about. To finish off, my grids are very easy to tweak, you can tweak the cell size of the spatial hashing grid, so you can have smaller cells the objects can interact with, which can lead to people experimenting with what the best cell size is for a custom spatial grid.

**What went wrong?**

One of the issues that I had when I was doing my project, was originally I had a different project proposal, which was to make a Physics engine in Godot, but after looking into and writing scripts in Godot I came to understand that there was no real reason to develop any physics engine in Godot. The reason for this is because Godot already had a very extensive Physics library. My next proposal was to make an engine for a 2D platformer game, after looking into it, I felt that I could investigate a research project, about how to make collisions more optimized in games in respect to scale.

Because of these new project proposals, it took time out of my project, which would have been used to look into more algorithms for my project, which in-turn would give more data to display.

**What (if anything) is still outstanding/missing (i.e., still left to do)?**

Everything I wanted to implement, and showcase is already in my final project, but if were to have more time. I would spend more time researching different algorithms and how these algorithms behave differently than one another and see what the use cases for those algorithms in video game development are.

**If starting again, how would you approach this project differently?**

If I were to start again, I would for sure collect a lot more data about the different spatial **part**itioning algorithms I have implemented and send that data through an API to be stored in a website database on a cloud service such as Anvil. So, I can utilize python libraries such as pandas to create graphs between all the different algorithms we have.

**What advice would you have for someone attempting a similar project in the future?**

I would advise, if anyone were to do a similar project like my one, is to look use my above implementation but have different approaches such as, you could find a way of finding the perfect cell size for the spatial grid or investigate distinct types of spatial **part**itioning algorithms and how they can perform better in certain use cases than the algorithms I have implemented.

**Were your technology choices the right or wrong ones?**

I believe that the technology choices I have made were the correct ones. Has I personally found the SFML library to quite easy to use and it is very well documented. If there are problems anyone encounters using SFML, there probably is a forum on the internet that answers the problem you are facing. Which leads to faster productivity.

**What were the implications of your technology choices?**

By using the SFML library I have safe countless hours of trying to understand how some of the lower-level graphic libraries work such as SDL and GLFW. By using SFML I also made it very easy to use modern c++ and the reason for this is because all of the SFML library is made in modern c++, so a lot of the new functionality in the c++ language works great with SFML.

# Conclusions

From the implementation I currently have of the different spatial **part**itioning algorithms significantly improves performance on machines with simulations or games that have a large scale in terms handling collision checks between hundreds and thousands of objects. Nevertheless, I do believe there could be improvements on my implementation. One of these improvements would be on my Quad-Tree implementation. My application as of right now, re-constructs the quad-tree every single frame which will lead to performance impact when there are an extremely substantial number of objects in my scene. A solution to this problem would be to only re-construct the quad-tree grid when an object leaves it cells to another cell and when new objects are created on top of the grid.

# Future Work

If anybody wanted to expand on what I have now, I would suggest looking into spatial **part**itioning algorithms that handle collision checks in a 3D domain. Some of these algorithms could be Oct-Trees or Hyper-Oct-Trees and comparing Quad-Trees, Spatial Hashing, Oct-Trees, and Hyper-Oct-Trees against each other to see which one of these algorithms perform better in the 3D domain. Another thing I would expand on, would be the data collection. By this I mean, the creation of a database website using a framework like Anvil for instance and sending the relative data between each different algorithm. Such as collision checks, frame rate and the number of objects within the scene. With this data, it will make it far easier to understand what the best algorithms are for collision handling. My last recommendation would be to create **part**itioning algorithms, that can work in both 2D and 3D, which would lead to a tool that other people can use in their games and simulations.

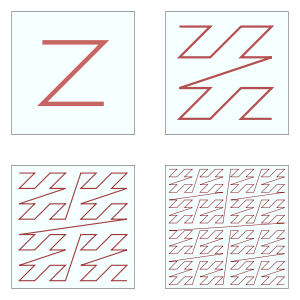
# References

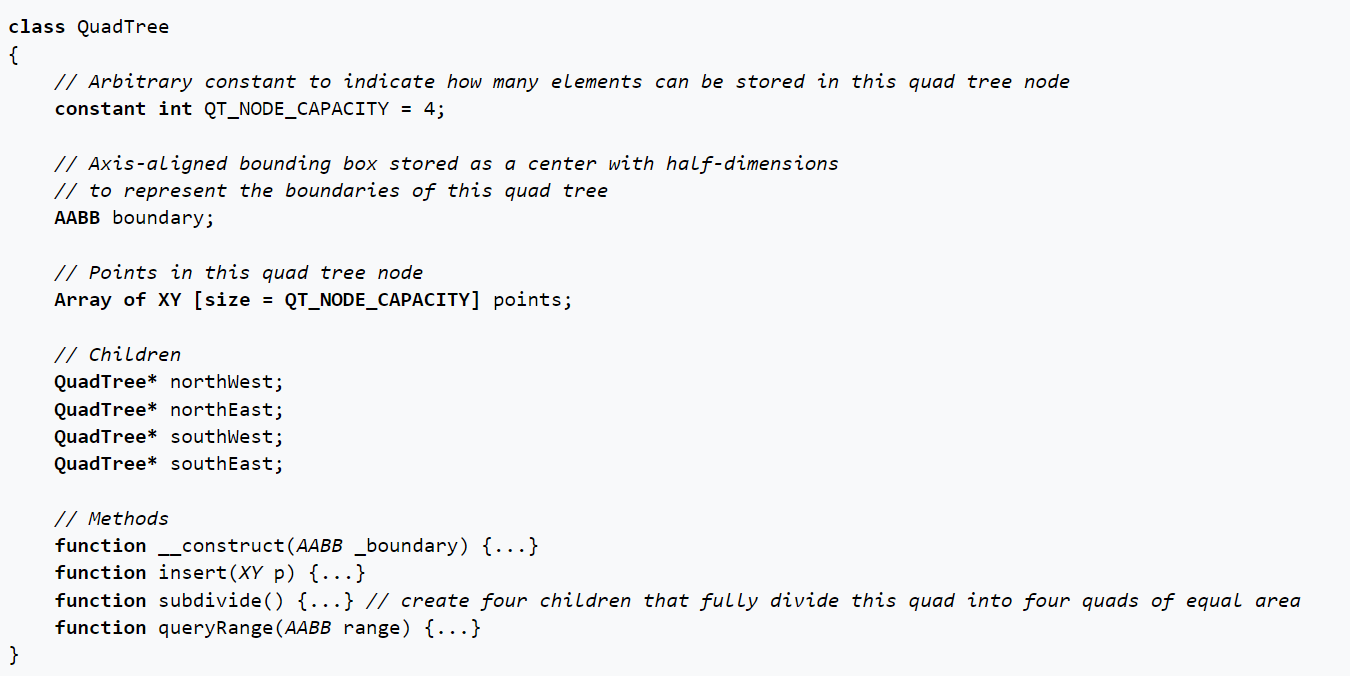
# **[1].** Anthony D’Angelo (2015) A Brief Introduction to Quadtrees and Their Applications [Online] Link to the paper can be found on this link: <http://people.scs.carleton.ca/~maheshwa/courses/5703COMP/16Fall/quadtrees-paper.pdf>

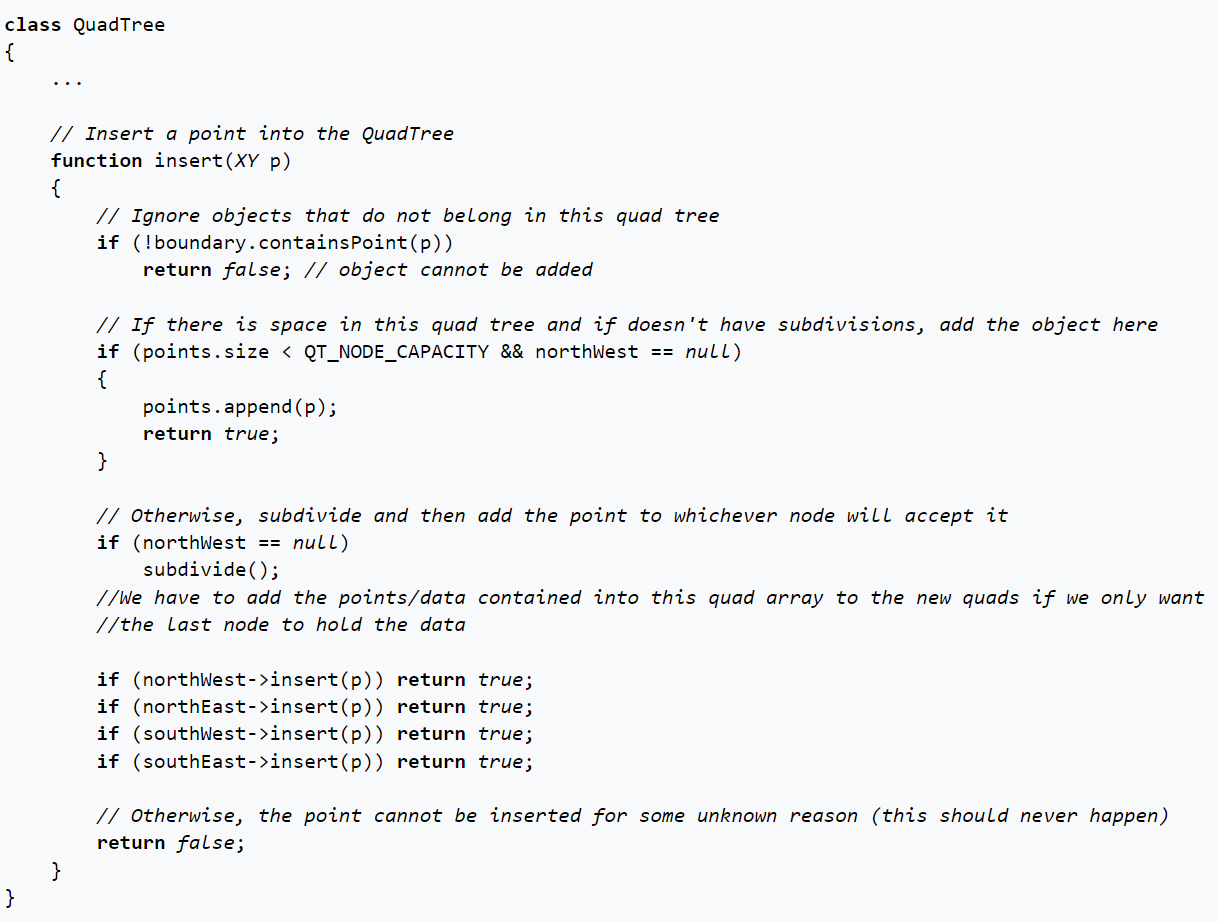
# **[2].** Optimization of Large-Scale, Real-Time Simulations by Spatial Hashing [Online] Link to the paper can be found on this link: <https://www.cs.ucf.edu/~jmesit/publications/scsc%202005.pdf>

# **[3].** Quadtree [Online] Link to the wiki page can be found on this link: <https://en.wikipedia.org/wiki/Quadtree>

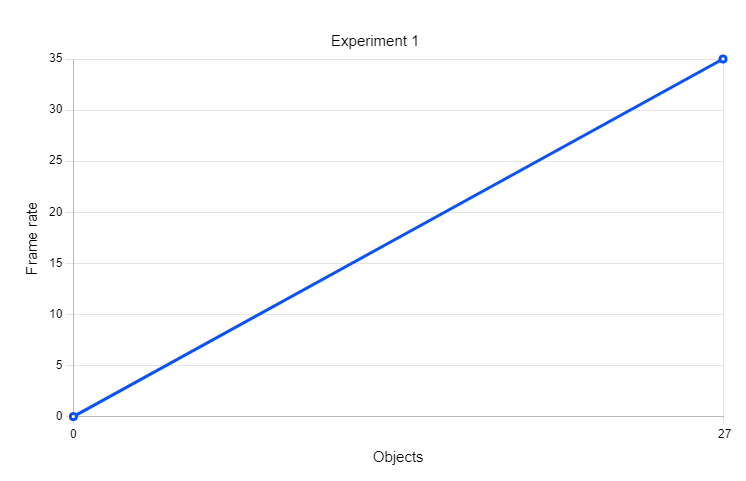
# Appendices

**Figure 1.1 displaying what a z-order curve is.**

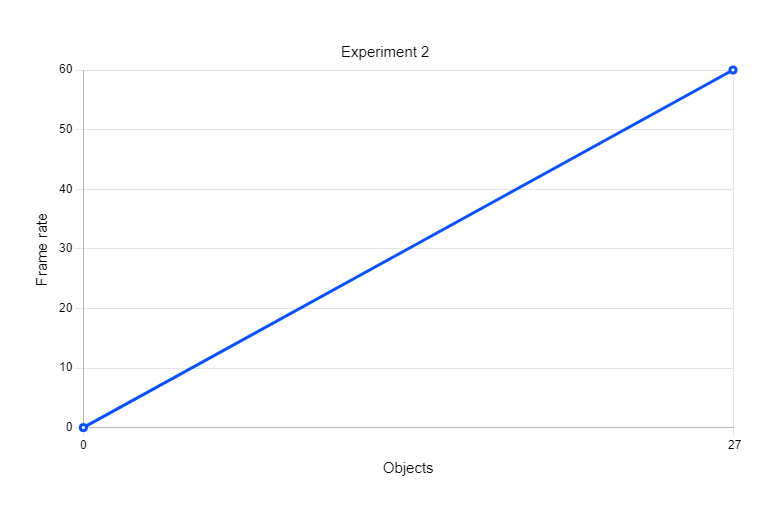


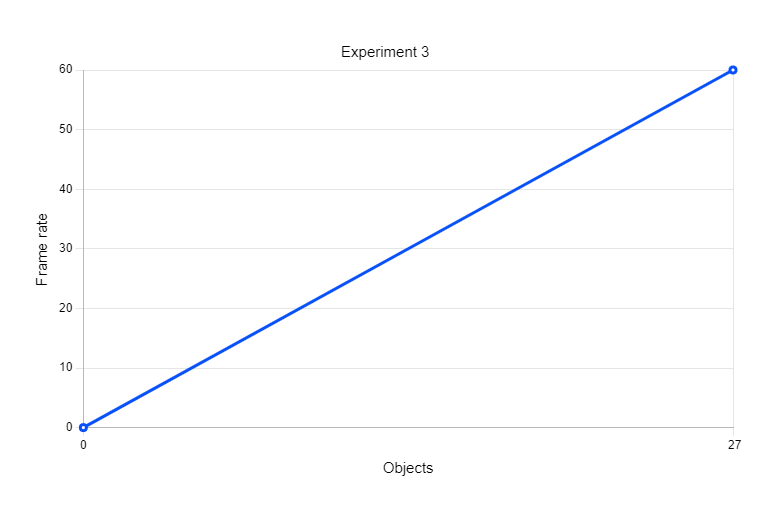
**Figure 1.2 highlights pseudocode on how to implement a Quad-Tree**

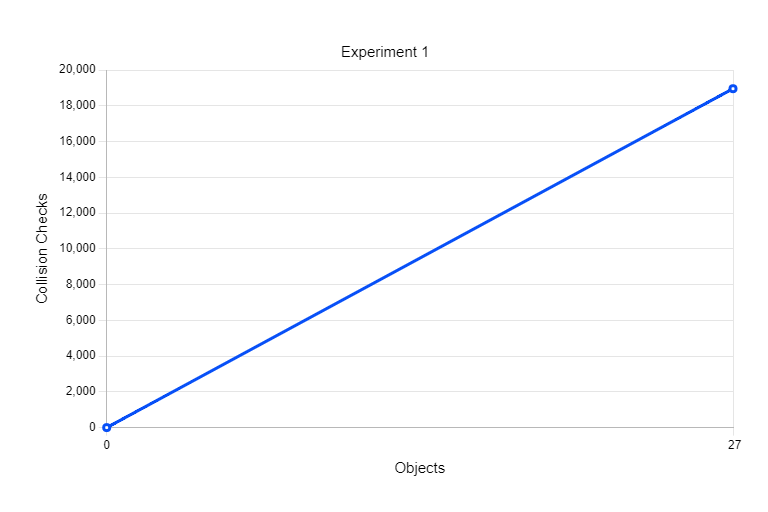
[**Wiki**](https://en.wikipedia.org/wiki/Quadtree) **This wiki page is where I got the pseudocode for the implementation of a quad-tree.**

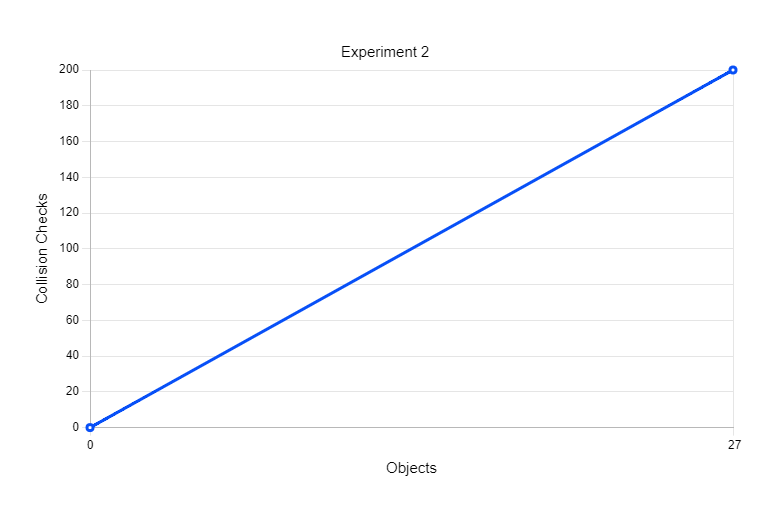


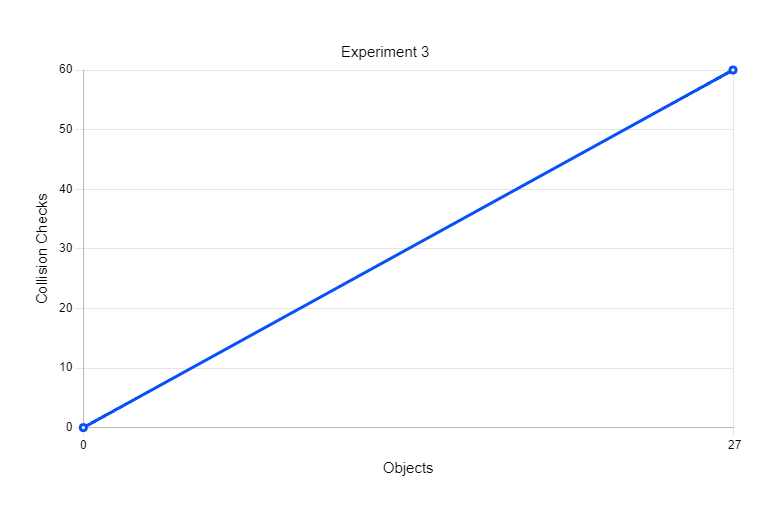
**Figure 1.4**

**Figure 1.5**

**Figure 1.6**

**Figure 1.7**

**Figure 1.8**

**Figure 1.9**