QuadTree Visualizer

(Fourth Year / Sem VIII)

Submitted in fulfilment of the requirement of
University of Mumbai For the Degree of

Bachelor Of Engineering
(Computer Engineering)

By

AMEY MAHENDRA THAKUR	**TU3F1819127**
HASAN MEHDI RIZVI	**TU3F1819130**
MEGA SATISH	**TU3F1819139**
AJAY RAMESH DAVARE	**TU3F1718006**

Under the Guidance of
Prof. Randeep Kaur Kahlon

terna
Knowledge to Empower

TERNA ENGINEERING COLLEGE
Department of Computer Engineering
Plot no.12, Sector-22, Opp. Nerul Railway station,
Phase-11, Nerul (w), Navi Mumbai 400706
UNIVERSITY OF MUMBAI

2021-2022

Internal Approval Sheet

Terna Engineering College

NERUL, NAVI MUMBAI

CERTIFICATE

This is to certify that the major project entitled **"QuadTree Visualizer"** is a bonafide work of

Submitted to the University of Mumbai in partial fulfilment of the requirement for the award of the **Bachelor of Engineering (Computer Engineering)**.

AMEY MAHENDRA THAKUR	**TU3F1819127**
HASAN MEHDI RIZVI	**TU3F1819130**
MEGA SATISH	**TU3F1819139**
AJAY RAMESH DAVARE	**TU3F1718006**

Under the guidance of

Prof. Randeep Kaur Kahlon

GUIDE **HOD** **PRINCIPAL**

Approval Sheet

Project Report Approval

This Major-Project Report entitled

“QuadTree Visualizer”

by the following students is approved for the degree of
Bachelor of Engineering in "Computer Engineering".

Submitted by:

AMEY MAHENDRA THAKUR	**TU3F1819127**
HASAN MEHDI RIZVI	**TU3F1819130**
MEGA SATISH	**TU3F1819139**
AJAY RAMESH DAVARE	**TU3F1718006**

Examiners Name & Signature:

1. ____________________________

2. ____________________________

Date:

Place:

Declaration

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced The cartoon sources task-specific. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

AMEY MAHENDRA THAKUR	**TU3F1819127**	______________
HASAN MEHDI RIZVI	**TU3F1819130**	______________
MEGA SATISH	**TU3F1819139**	______________
AJAY RAMESH DAVARE	**TU3F1718006**	______________

Date:

Place:

Acknowledgement

We would like to express our sincere gratitude towards our guide **Prof. Randeep Kaur Kahlon**, project coordinator **Prof. Pramil Mate** and project convener **Prof. Vishwajit Gaikwad** for the help, guidance, and encouragement they provided during the project development. This work would have not been possible without their valuable time, patience and motivation. We thank them for making our stint thoroughly pleasant and enriching. It was great learning and an honour being their student.

We are deeply thankful to **Dr. Archana Mire** (H.O.D Computer Department) and the entire team in the Computer Department. They supported us with scientific guidance, advice and encouragement, they were always helpful and enthusiastic and this inspired us in our work.

We take the privilege to express our sincere thanks to **Dr. L. K. Ragha**, our Principal for providing encouragement and much support throughout our work.

Abstract

We propose to develop a program that can show a QuadTree view and data model architecture. Nowadays, many digital map applications have the need to present large quantities of precise point data on the map. Such data can be weather information or the population in towns. With the development of the Internet of Things (IoT), we expect such data will grow at a rapid pace. However, visualizing and searching in such a magnitude of data becomes a problem as it takes a huge amount of time. QuadTrees are trees used to efficiently store data of points in a two-dimensional space. In this tree, each node has at most four children. QuadTrees allow us to visualize the data easily and rapidly compared to other data structures. This project aims to build an application for interactively visualizing such data, using a combination of grid-based clustering and hierarchical clustering, along with QuadTree spatial indexing. This application illustrates the simulation of the working of the QuadTree data structure.

TABLE OF CONTENTS

List of Figures

List Of Abbreviations

Acronym	Abbreviation
SPA	Single Page Application
SDK	Software Development Kit
SDLC	Software Development Life Cycle
COR	Coefficient of Restitution
API	Application Programming Interface

Chapter 1

Introduction

1.1 Aim & Objective of the Project

This project aims to provide a web application for visualising the QuadTree structure. QuadTree. The users should be able to understand the working of the QuadTree and experience the simulation provided on the web application. This Visualizer provides an interactive environment where users can change configurations of the QuadTree and environment conditions at runtime.

1.2 Scope of the Project

Since QuadTrees are a type of tree data structure in which each internal node has exactly four children, they are most often used to partition a two-dimensional space by recursively subdividing it into four quadrants or regions. The regions may be square or rectangular or may have arbitrary shapes. The QuadTree is used as a utility as part of the Maps SDK for iOS Utility Library. They've also been heavily used in image compression algorithms and higher-level design of 8-bit games like Mario.

Eventually, we believe that QuadTrees can be used for memory management in a big and hierarchical database. It is one of the most crucial places we can use the QuadTree and it can be used to access varied data points and make searching efficient and fast.

1.3 The Organisation of the Report

- Chapter 1 gives a brief overview of the aim of developing this project. Also, it defines the objectives and the scope of the project.
- Chapter 2 of the report includes a brief description of the quadtree and the literature survey on the existing systems. It also defines the problem statement and proposes a new system as a solution.
- Chapter 3 elaborates on the proposed methodology used in our proposed system as well as the distinct architectures of the components. It gives a detailed explanation of QuadTree data structure, its types, its limitations, the different operations performed in QuadTree, and its complexity. Along with this, some features added to projects are also discussed such as collision and coefficient of restitution. The working model and the workflow of the application are explained in this section along with the testing performance process.
- Chapter 4 provides information about the system requirements, hardware and software, and the tools and technologies used while implementing the project
- Chapter 5 includes the experimental setup of the project and the software development model, the Big Bang model which is used in the project.
- Chapter 6 depicts the timeline of the project with a Gantt Chart detailing the project schedule
- Chapter 7 shows the implementation and the results.
- Chapter 8 gives the conclusion of the project.
- References used while developing the project are mentioned at the end of the report.

Chapter 2
Literature Survey

2.1 Brief History of QuadTree

A QuadTree is a tree data structure in which each node has zero or four children. Its main peculiarity is its way of recursively dividing a flat 2-D space into four quadrants. The data associated with a leaf cell varies by application, but the leaf cell represents a "unit of interesting spatial information".

The subdivided regions may be square or rectangular or may have arbitrary shapes. This data structure was named a QuadTree by Raphael Finkel and J.L. Bentley in 1974.

2.2 Existing Systems

The paper “An effective way to represent quadtrees” published in 1982 by Irene Gargantini, proposes a new structure very similar to QuadTree, called a “linear quadtree” and different algorithms used to represent that structure. The linear QuadTree saves 66% of the computer storage required by regular QuadTrees. Later on, in “Optimal quadtree construction algorithms”, 1987, a paper written by Clifford A.Shaffer, and Hanan Samet, introduces an algorithm for constructing a QuadTree in time proportional to the number of blocks in a given picture is described. In 2016, Qing Cai, Yimin Zhou published a paper called “A quadtree-based hierarchical clustering method for visualizing large point dataset” which proposes introducing a new clustering method with QuadTree spatial indexing. It explains a grid-based, partitioning, hierarchical clustering method on QuadTree file system storage. The above Literature survey is illustrated in the table below:

Author's Name	Title and Year of Publication	Findings
Qing Cai, Yimin Zhou	A quadtree-based hierarchical clustering a method for visualizing large point dataset, 2016	This paper introduces a new clustering method with quadtree spatial indexing. It explains a grid-based, partitioning, hierarchical clustering method on quadtree file system storage.
Clifford A.Shaffer, Hanan Samet	Optimal quadtree construction algorithms, 1987	In this paper, an algorithm for constructing a quadtree in time proportionate to the number of blocks in a given picture is described.
Irene Gargantini	An effective way to represent quadtrees, 1982	This paper proposes a new structure very similar to a quadtree, called a "linear quadtree" and different algorithms used to represent that structure. The linear quadtree saves 66% of the computer storage required by regular quadtrees.

2.3 Problem Statement

The importance of data nowadays has increased significantly, as we are living in a data-driven society. Many digital map applications have the need to present large quantities of precise point data on the map. With the development of the Internet of Things, we expect such data will grow at a rapid pace. However, visualising and looking for a data point in such a magnitude of data becomes a problem. We are proposing the structural implementation of a type of binary tree called QuadTrees and its subsequent visualisation for users to interact with. This system provides a real-time simulation of the working of the quadtree where the user can see the division of each node in the quadtree.

Chapter 3
Proposed Methodology

3.1 A Brief Introduction To QuadTrees

The QuadTree is a data structure for organizing objects based on their locations in a two-dimensional space. By definition, a QuadTree is a tree in which each node has at most four children. QuadTree implementations ensure that as points are added to the tree, nodes are rearranged such that none of them has more than four children. Figure 3.1 below illustrates the general concept of quadtree data structure.

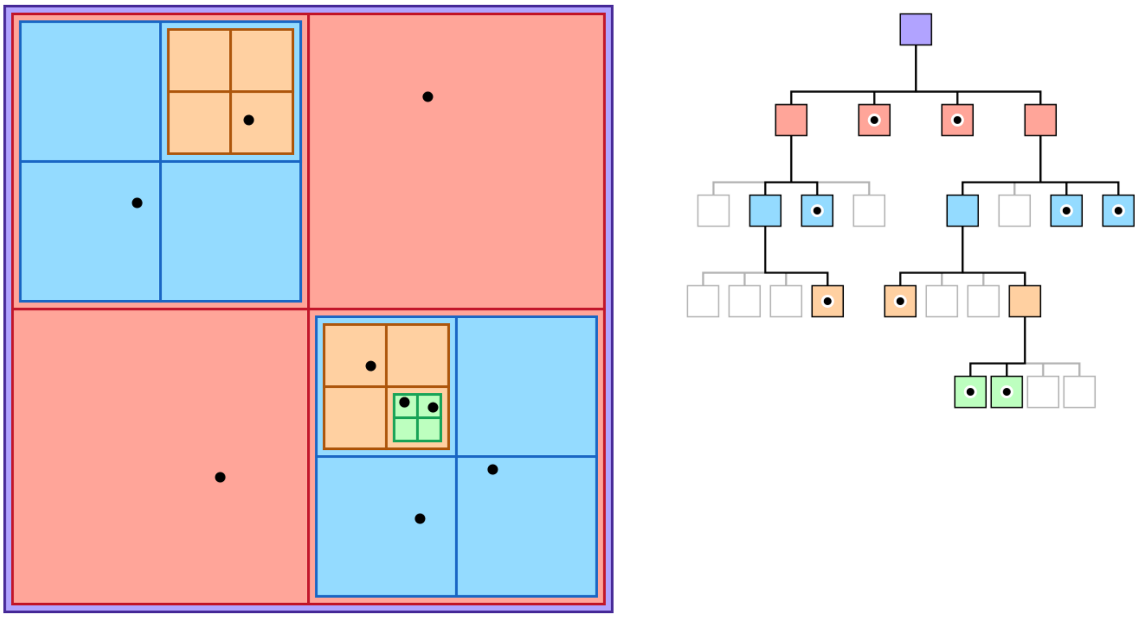


Figure 3.1: QuadTree Data Structure

The QuadTree partitioning strategy divides space into four quadrants at each level. When a quadrant contains more than one object, the tree subdivides that region into four smaller quadrants, adding a level to the tree. A similar partitioning is also known as a Q-tree. QuadTrees are a way of partitioning space so that it's easy to traverse and search.

3.2 Applications of QuadTree

It is used extensively in computer graphics, image compression and is also used to represent spatial relations.

Visualizing data points with a quadtree and checking and detecting collisions. Collision detection is the computational problem of detecting the intersection of two or more objects. Collision detection is a classic issue of computational geometry and has applications in various computing fields. Figure 3.2.1 shows the use case of Quadtree Visualizer.

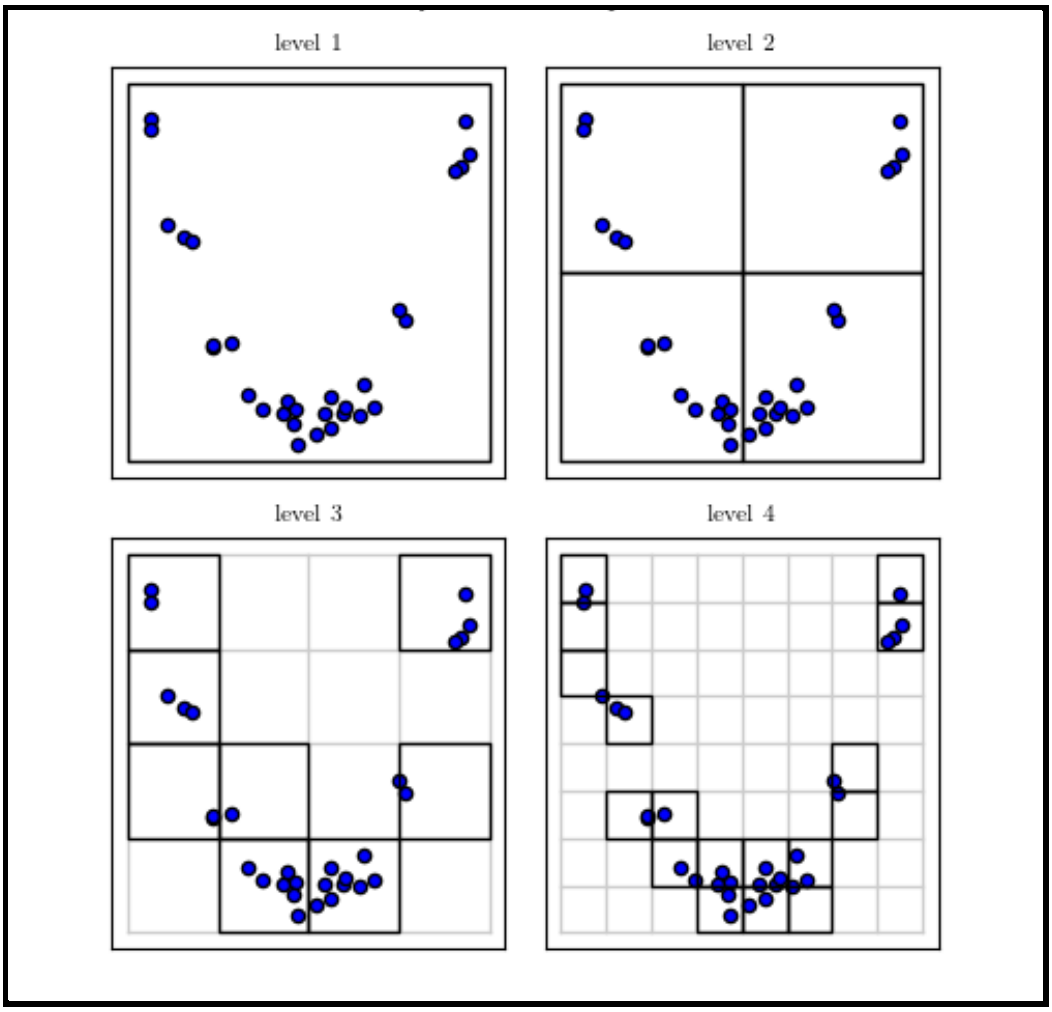


Figure 3.2.1: QuadTree Visualizer

Quadtrees are also implemented for spatial indexing while searching a particular point or location in a map. QuadTrees are very efficient as they can sparse through the maps very easily and quickly compared to other methods. Figure 3.2.2 shows the use case of QuadTree Spatial Indexing

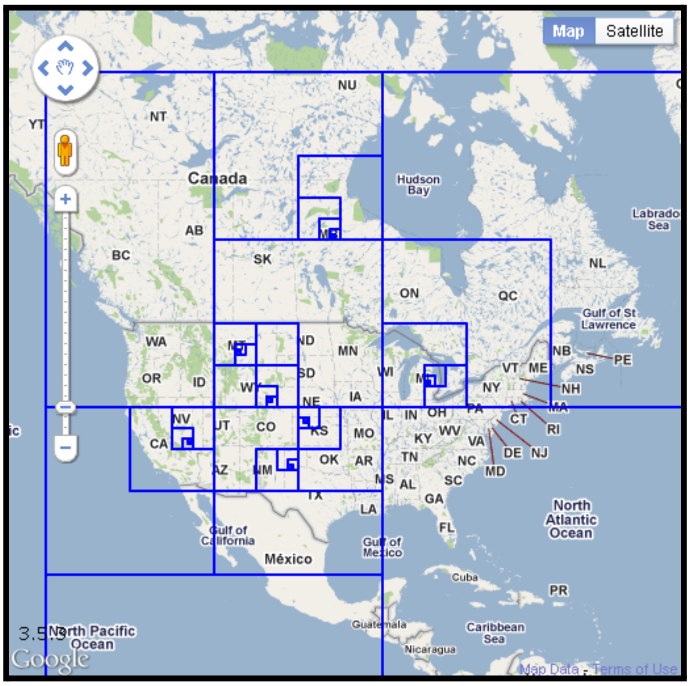


Figure 3.2.2: QuadTree Spatial Indexing

Quadtrees, for example, can handle a sparse Mario level a billion tiles across, where one of the tiles contains the finishing spot. A quadtree will split the arrival spot into different cells and still use gigantic cells for the empty spaces. Figure 3.2.3 shows the use case of QuadTree in Gaming.

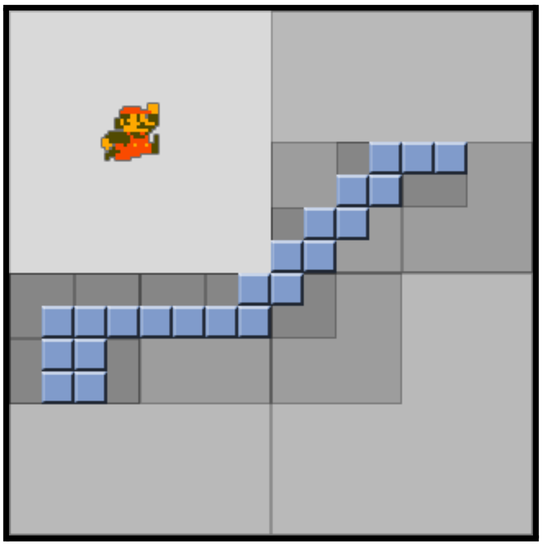


Figure 3.2.3: QuadTree in Gaming

3.3 Some possible use cases of QuadTree

3.3.1 Hit detection:

For example, there are a bunch of points in space, like in the maps above. If we want to find some arbitrary point P is within that lot of points. This becomes a hectic task. We could compare every single point to P, but if there are 1000 points and none of them was P, that will be 1000 comparisons to find out that particular point P.

Alternatively, we could get a very fast lookup by keeping a grid (a 2D array) of booleans for every single possible point in this space. However, if the space these points are on is 1,000,000 x 1,000,000, and we need to store 1,000,000,000,000 variables.

In this case, a QuadTree would be a better option. To search for P, the QuadTree will find out which quadrant it is inside. Then, it will find out what quadrant within that quadrant it is inside. It will only have to do this at most seven times for a 100x100 space (assuming points can only have integer values), even if there are 1000 points in it. For a 1,000,000x1,000,000 space, it's a maximum of 20 times. After it finds its way to that rectangle node, it merely needs to see if any of the four children equal P.

3.3.2 Sparse Data using QuadTree:

QuadTrees are ideal for sparse data to search for a particular point. QuadTrees help with gathering information about which collisions in an environment are worth testing by only making computations between objects in similar nodes/quads.

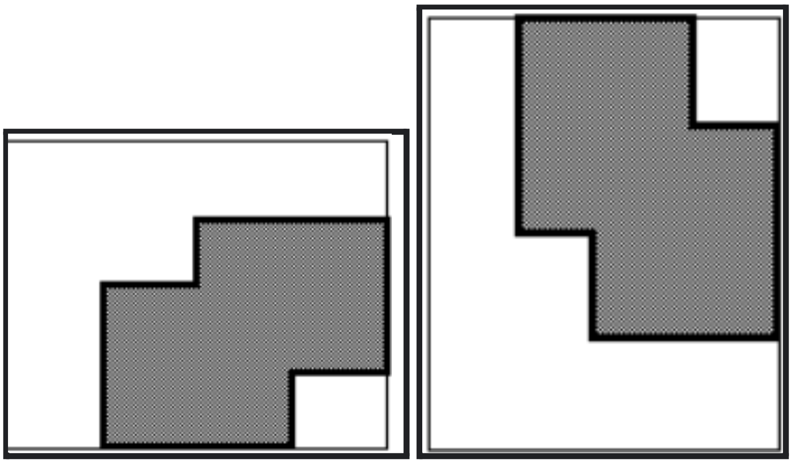
QuadTree nodes split into four evenly-sized leaf nodes when the number of objects inside them reaches a certain capacity. Objects are inserted into a fresh QuadTree every iteration, which places each object in its deepest possible node.The QuadTree algorithm improves upon the naive $T(n) = \theta(n2)$ algorithm and achieves $T(n) = O(n2)$, $T(n) = \Omega(nlog(n))$. QuadTrees based on pixels are incidentally a type of trie.

3.4 Limitations of QuadTree

The main disadvantage of QuadTrees is that it is almost impossible to compare two images that differ only in rotation or translation. This is because the QuadTree representation of such images will be so totally different.

The algorithms available for rotation of an image are restricted to rotations of 90 degrees (or multiples thereof). No other rotation is available, nor is there a facility for translation.

Figure 3.4.1 shows the original image and Figure 3.4.2 shows the rotated images. As we can see, it is not possible to compare two images that are different in terms of rotation.



(3.4.1) First Image **(3.4.2) Rotated Image**

Figure 3.4

3.5 Types of QuadTree

There are three types of quadtrees:

1. **Point** QuadTree
2. **Edge** QuadTree
3. **Polygonal Map** QuadTree.

All forms of QuadTrees share some common features:

- They decompose space into adaptable cells.
- Each cell (or bucket) has a maximum capacity. When maximum capacity is reached, the bucket splits.
- The tree directory follows the spatial decomposition of the QuadTree.

3.6 Working of the QuadTree

The image below shows how a quadtree changes with insertion:

- Divide the current two-dimensional space into four boxes.
- If a box contains one or more points in it, create a child object, storing in it the two-dimensional space of the box.
- If a box does not contain any points, do not create a child for it.
- Recurse for each of the children.

Figure 3.6 below shows the working of the quadtree model while inserting a point E.

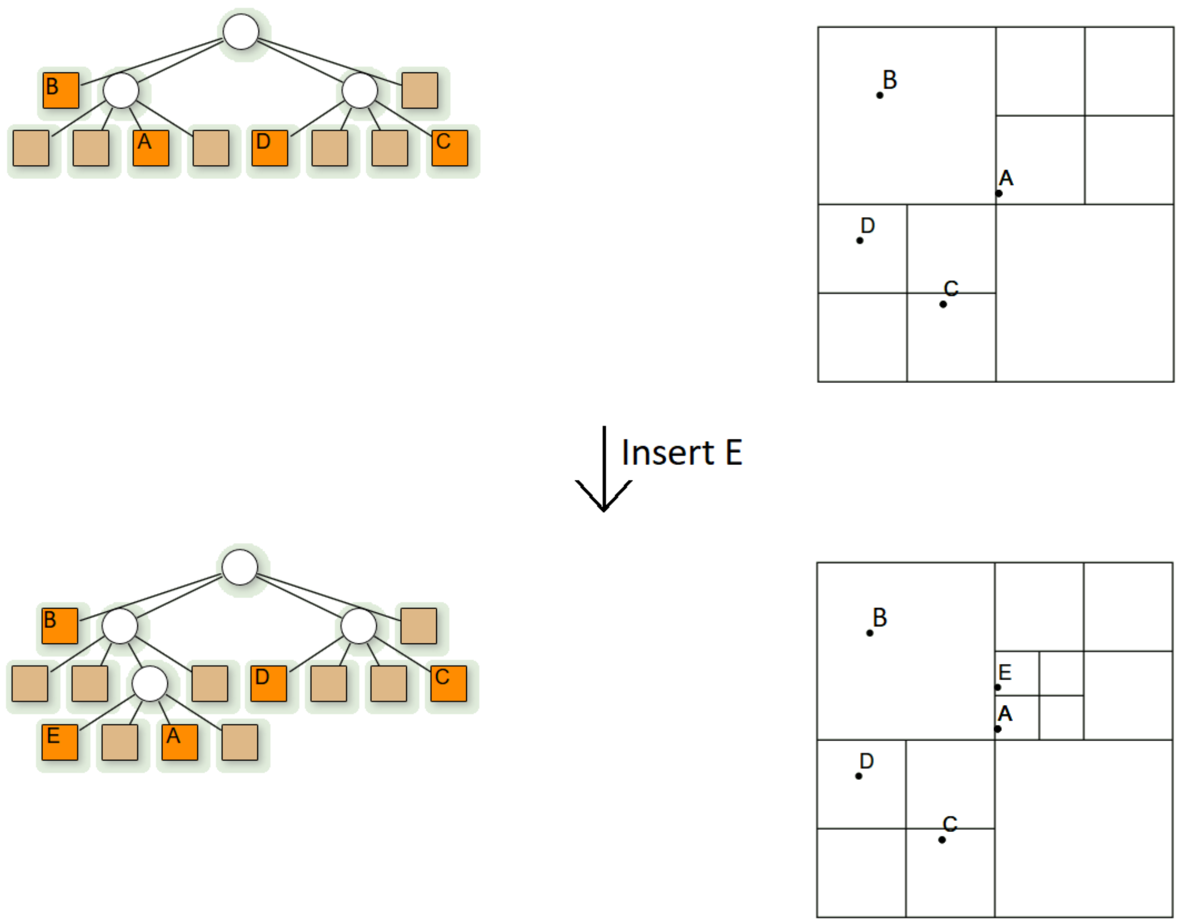


Figure 3.6: Working of QuadTree

3.7 Algorithm

Three types of nodes are used in quadtree:

1. **Point node:** It is used to represent a point. It is always a leaf node.
2. **Empty node:** It is used as a leaf node to represent that no point exists in the region it represents.
3. **Region node:** This is always an internal node. It is used to represent a region. A region node always has 4 children nodes that can either be a point node or an empty node.

3.8 Insertion in QuadTree

- Insertion: This is a recursive function used to store a point in the quadtree.
    1. Start with the root node as the current node.
    2. If the given point is not in the boundary represented by the current node, stop insertion with error.
    3. Determine the appropriate child node to store the point.
    4. If the child node is empty, replace it with a point node representing the point. Stop insertion.
    5. If the child node is a point node, replace it with a region code. Call insert for the point that just got replaced. Set the current node as the newly formed region node.
    6. If the selected child node is a region node, set the child node as the current node. Go to step 2.

3.9 Search in QuadTree

- Search: This is a boolean function used to determine whether a point exists in 2D space or not.
    1. Start with the root node as the current node.
    2. If the given point is not in the boundary represented by the current node, stop searching with error.
    3. Determine the appropriate child node to store the point.
    4. If the child node is empty, return FALSE.
    5. If the child node is a point node and it matches the given point return TRUE, otherwise return FALSE.
    6. If the child node is a region node, set the current node as the child region node. Go to step 2.

3.10 Complexity

3.10.1 Time complexity:

- Find: **O(log2N)**
- Insert: **O(log2N)**
- Search: **O(log2N)**

3.10.2 Space complexity:

- **O(k log2N)**, where k is the count of points in the space and Space is of dimension $N \times M, N >= M$

3.11 Collision in QuadTrees

Since the data points are constantly moving in the visualizer, collision is bound to happen. Collision is an encounter between two bodies, here we have data points in the form of circles. The Quadtree visualization sits atop a 2D Collision System with a configurable coefficient of restitution, used to adjust between elastic and inelastic collisions. Collision detection can be an expensive operation. Quadtrees are one way you can help speed up collision detection.

3.12 Coefficient of Restitution

The ratio of final velocity to the initial velocity between two objects after their collision is known as the coefficient of restitution. The restitution coefficient is denoted as ‘e’ and is a unitless quantity, and its values range between 0 and 1.

The measure of the colliding materials’ nature is represented by a number known as the coefficient of restitution. The coefficient of restitution provides us with information about the elasticity of the collision. Collisions in which there is no loss of overall kinetic energy is known as a perfectly elastic collision. This type of collision has the maximum coefficient of restitution of e = 1. A collision, where maximum kinetic energy is lost, is known as a perfectly inelastic collision. They have a coefficient of restitution of e = 0. Most real-life collisions are in between. The mathematical formula of the Coefficient of Restitution is given as follows:

$$\textbf{Coefficient of Restitution (e)} = \frac{Relative\ Speed\ After\ Collision}{Relative\ Speed\ Before\ Collision}$$

From the above equation, you notice that you always divide the smaller number by a more significant number. Therefore, the coefficient of restitution is always positive.

3.13 Workflow of QuadTree

Figure 3.13 illustrates the workflow of the quadtree application. Next.js is responsible for both client and server-side scripting.

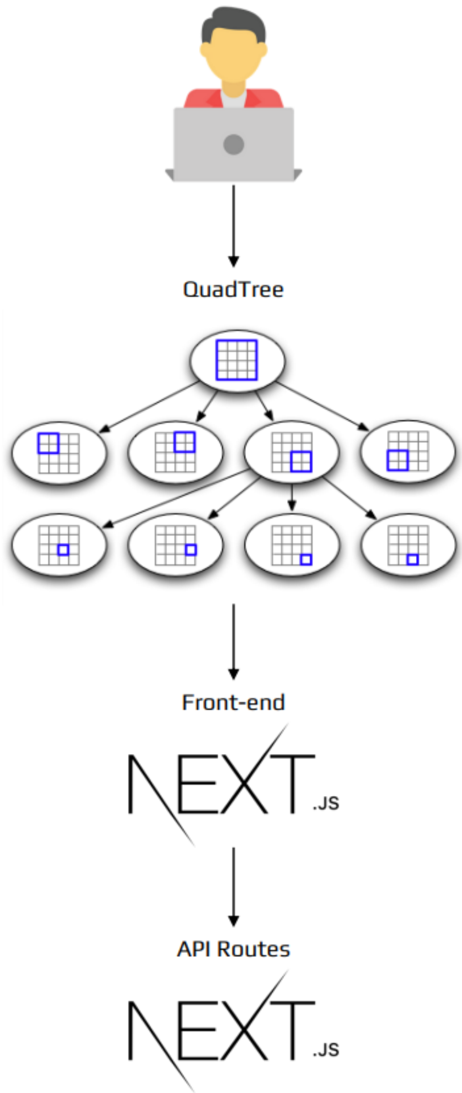


Figure 3.13: Workflow of QuadTree

3.14 Model Architecture

An architecture model is a partial abstraction of a system. It is an approximation, and it captures the different properties of the system. It is a scaled-down version and is built with all the essential details of the system. Architecture modelling involves identifying the characteristics of the system and expressing them as models so that the system can be understood. Architecture models allow the visualization of information about the system represented by the model. Figure 3.14 shows the model architecture of the web application.

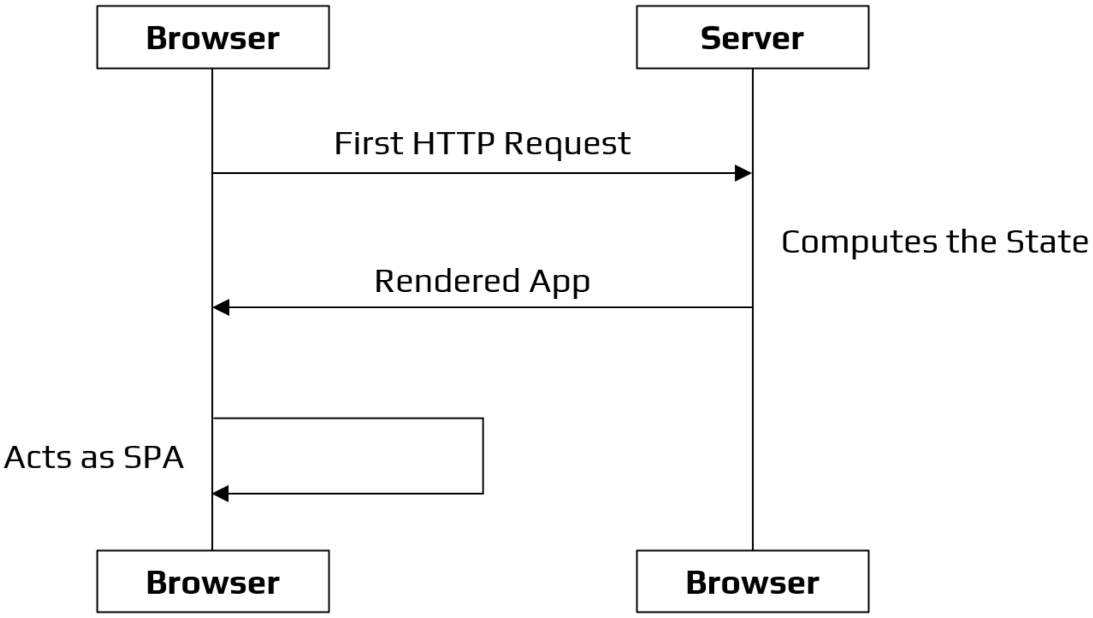


Figure 3.14: Model Architecture

3.15 Performance Testing Steps

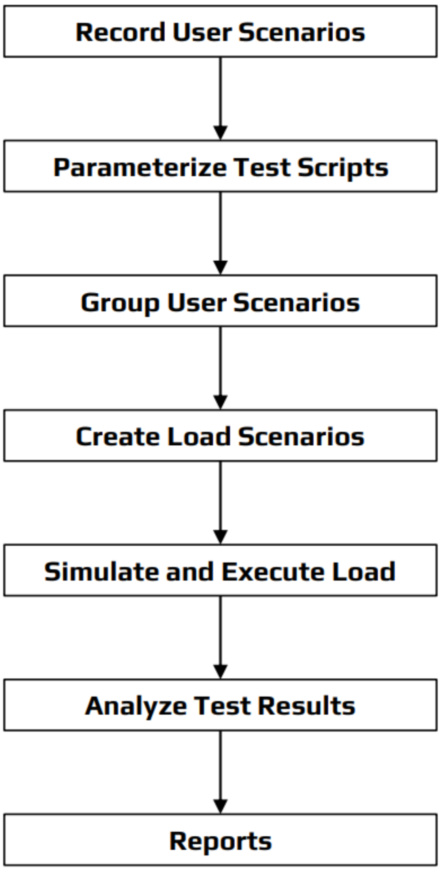


Figure 3.15: Performance Testing Process

Performance testing is a non-functional software testing technique that determines how the stability, speed, scalability, and responsiveness of an application hold up under a given workload. Figure 3.15 depicts the seven phases involved in performance testing.

1. Identify the testing environment

Identifying the hardware, software, network configurations and tools available allows the testing team to design the test and identify performance testing challenges early on. Performance testing environment options include:

- A subset of production system with fewer servers of lower specification
- A subset of production system with fewer servers of the same specification.
- Replica of productions system.
- Actual production system.

2. Identify performance metrics

In addition to identifying metrics such as response time, throughput and constraints, identify what are the success criteria for performance testing.

3. Plan and design performance tests

Identify performance test scenarios that take into account user variability, test data, and target metrics. This will create one or two models.

4. Configure the test environment

Prepare the elements of the test environment and instruments needed to monitor resources.

5. Implement your test design

Develop the tests.

6. Execute tests

In addition to running the performance tests, monitor and capture the data generated.

7. Analyze, report, and retest

Analyze the data and share the findings. Run the performance tests again using the same parameters and different parameters.

Chapter 4
Requirements

4.1 Software Requirements

- GitHub
- VSCode
- Web Browser

4.2 Hardware Requirements

- 4 GB RAM
- Any Operating System

4.3 Tools Used

- NPM Dependencies
- Command Prompt/Terminal

4.4 Technologies Used

- HTML 5.0
- CSS 3.0
- JavaScript, v. ES13
- TypeScript, v.4.6.3
- Node.js, v17.9.0
- Next.js, v10.0.5
- React, v17.0.1

Chapter 5
Design

5.1 Experimental Setup

- Since we are using Next.js in our project, we first need to have Node.js.
- The web application works on http://localhost:3000.
- To run the application locally, we need to install the packages required using the npm command: npm install package.json
- Figure 5.1.1 shows the command prompt with the packages installed using the npm install commands.

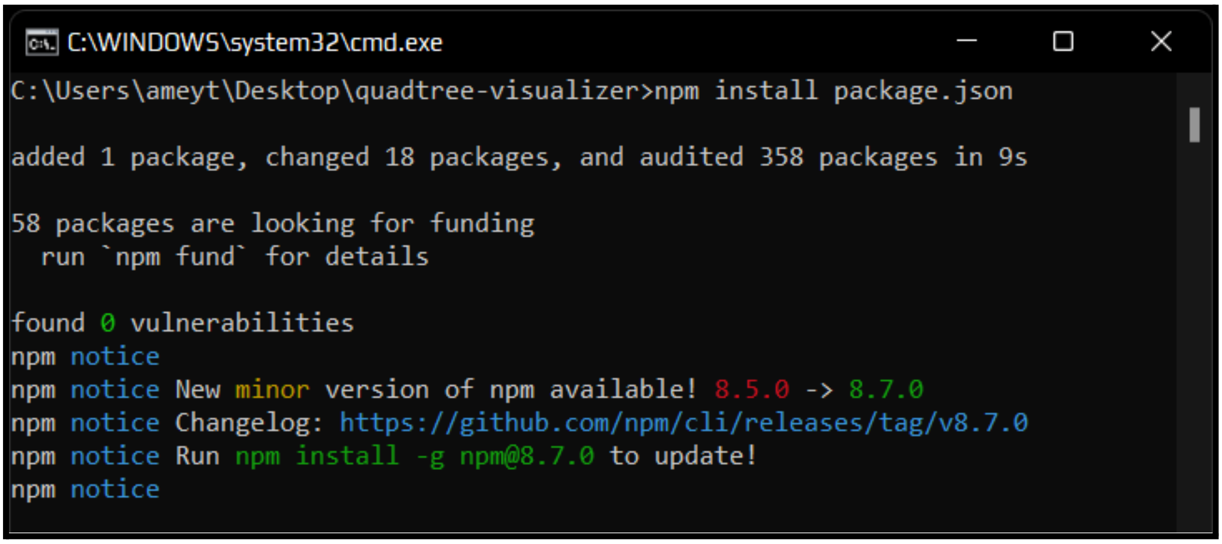


Figure 5.1.1: Command: npm install package.json

- After installing all the dependencies, we then run the command: npm run dev.
- Figure 5.1.2 shows the npm command used to run the website on the server.

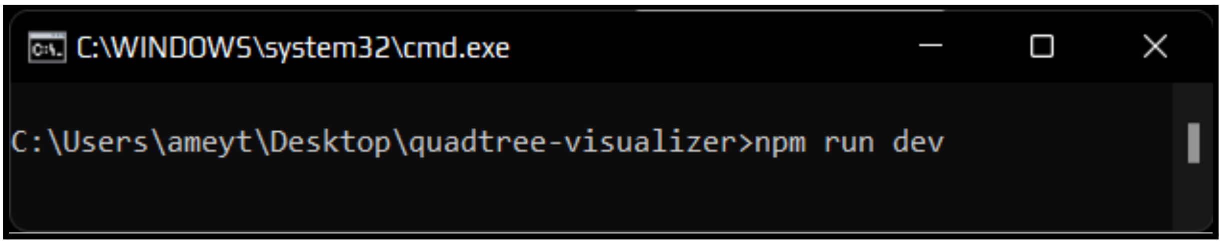


Figure 5.1.2: Command: npm run dev

- After we run the command: npm run dev. It will run the developer server.
- Figure 5.1.3 depicts the compilation and running of the server. The server is working on http://localhost:3000.

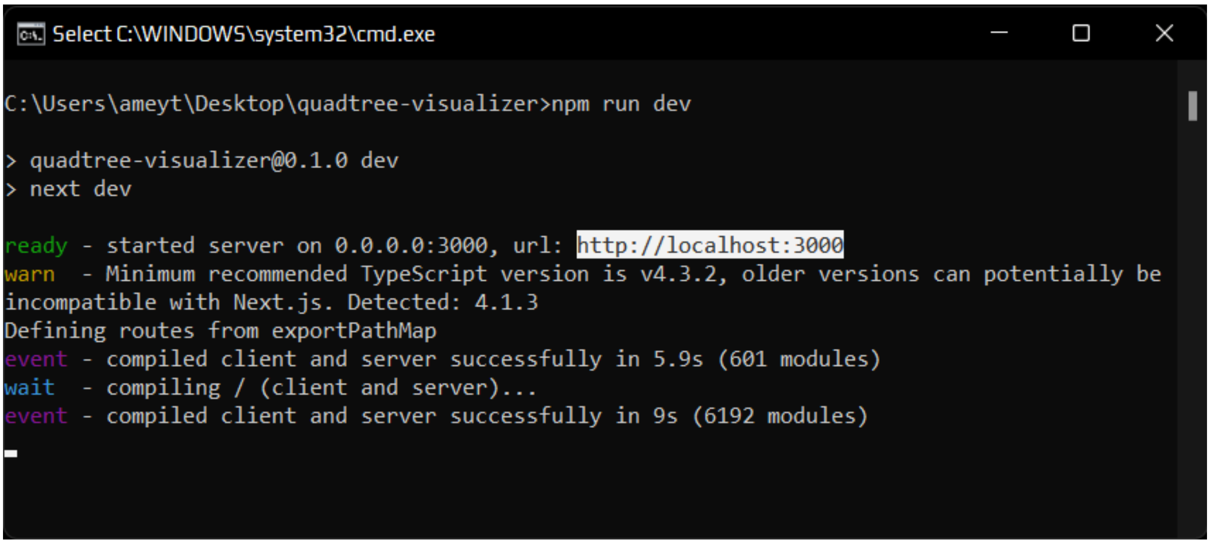


Figure 5.1.3: Compilation & Server Hosting

5.2. SDLC Model

The Big Bang model is an SDLC approach in which no precise procedure is followed. The development process begins with the necessary funds and efforts as inputs, and the result is software-generated, which may or may not meet the needs of the client.

This Big Bang Model has no set method or procedure and requires very little forethought. Even if the consumer is unsure of what he wants, the needs are implemented without much thought on the spot. This model is effective for this project because it gives the flexibility to work freely in any environment and gather requirements as and when necessary. This model is ideal for small projects with one or two developers working together and is also useful for academic or practice projects. It is an ideal model for the product where requirements are not well understood and the final release date is not given. Figure 5.2 illustrates the Big Bang Model of Software Development Life Cycle

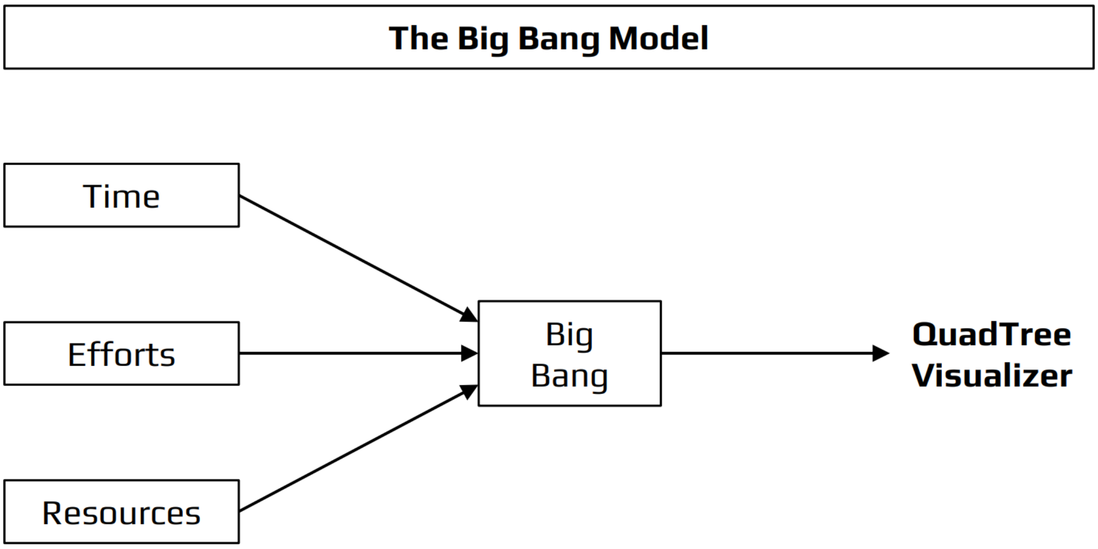


Figure 5.2: SDLC - Big Bang Model

5.2.1 How is the Big Bang Model efficient for this project

Since we started working on this project with a blank slate and very little past work to refer to, we believed the Big Bang model would be ideal for a project operating at this scale, with two to three developers working together and it has also been widely renowned for its use in academic or practice projects. It is an ideal model for the product where requirements are not well understood and the final release date is not given.

Chapter 6
Project Timeline

6.1 Gantt Chart

Figure 6.1.1 below shows the project schedule for the QuadTree Visualizer and Figure 6.1.2, 6.1.3 & 6.1.4 shows the Gantt chart for the relative dates.

QuadTree Visualizer

By Amey Thakur, Hasan Rizvi, Mega Satish & Ajay Davare

Project Start: Wed, 9-15-2021

Display Week: 22

TASK	PROGRESS	START	END
Research Paper			
Conceptual understanding	100%	9-15-21	9-30-21
Literature Survey	100%	10-1-21	10-22-21
High-level analysis of algorithm	100%	10-23-21	11-12-21
Problem Statement			
Comprehensive Study of Core Concepts and Terminologies	100%	11-13-21	11-28-21
Design			
Define Wokflow	100%	11-29-21	12-13-21
Design Architecture	100%	12-14-21	12-28-21
Selection of Necessary Tools and Technologies	100%	12-15-21	12-29-21
Implementation			
Selection of Server Hosting	100%	12-30-21	1-14-22
Server Configuration	100%	1-15-22	1-30-22
Deployment of Core Module	100%	1-31-22	2-21-22
Deployment of Supporting Modules	100%	2-22-22	3-9-22
Final Deployment	100%	3-10-22	3-25-22

Figure 6.1.1: Project Schedule

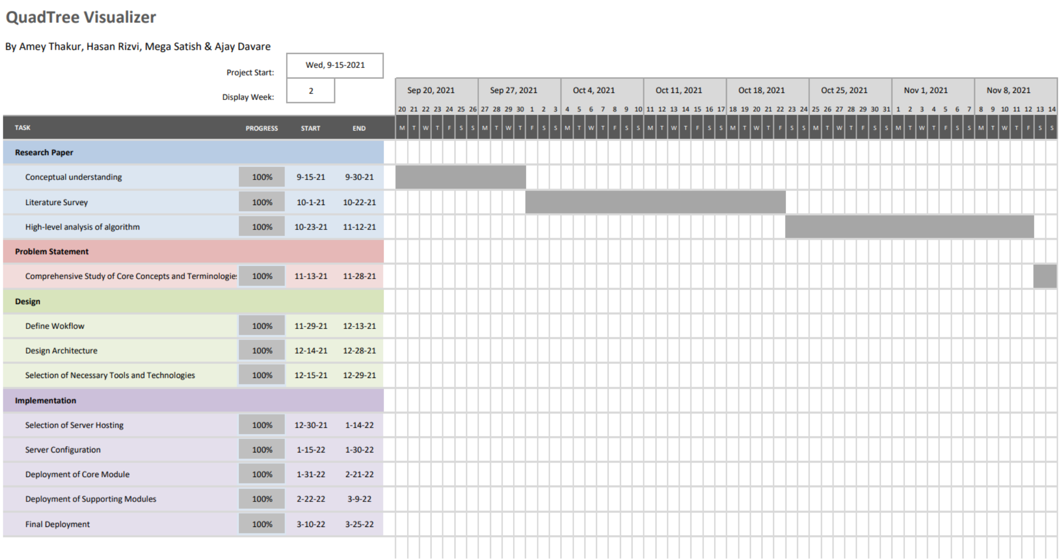


Figure 6.1.2: Gantt Chart View from September 25th to November 6th, 2021

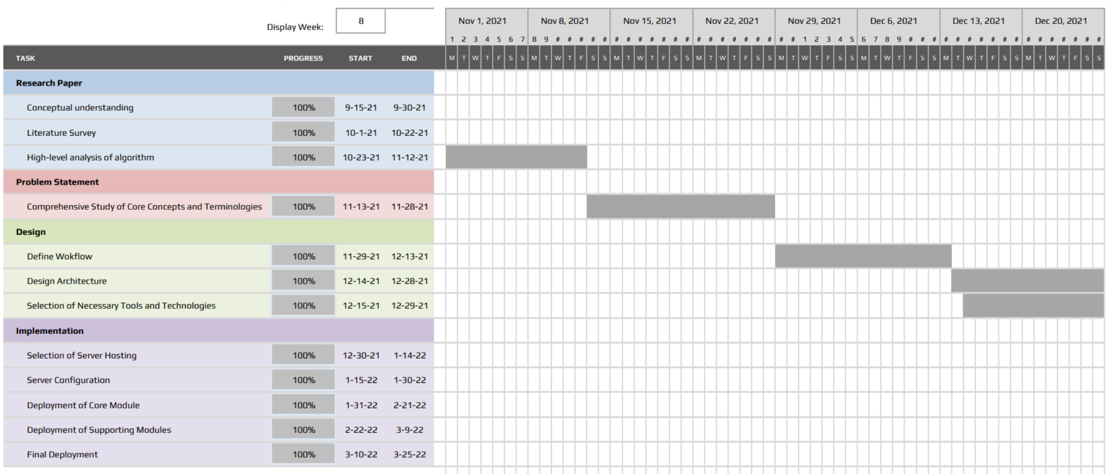


Figure 6.1.3: Gantt Chart View from November 1st to December 21st, 2021

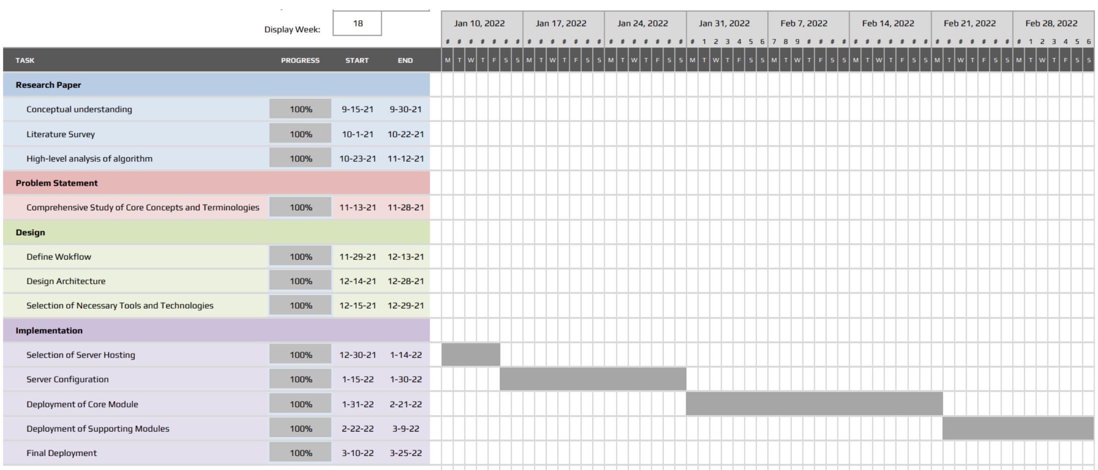


Figure 6.1.4: Gantt Chart View from January 16th to February 28th, 2022

Chapter 7
Implementation and Results

7.1 Implementation

- - Quadtree.ts – -

```
// A QuadNode object must have a way to tell if it is inside a given node's bounds

export interface QuadObject {
  insideRect: (rect: Rect) => boolean // whether the object in fully contained within a Rect
}

 // Node of a QuadTree
 // carries data about its:
 //  - bounds
 //  - depth
 // - children
 // - containing objects

export class QuadNode {
  public leaves!: Array<QuadNode> | null
  public quadObjects = new Array<QuadObject>()
  constructor(
```

```
    public bounds: Rect,
    private depth: number) { }

  // cleanup references down the QuadTree recursively

  clear(): void {
    this.quadObjects = new Array<QuadObject>()
    this.leaves?.forEach((leaf: QuadNode) => leaf.clear())
    this.leaves = null
  }

 // process any updates recursively down the tree

  process(quadNodeProcedure: (quadNode: QuadNode) => void): void {
    quadNodeProcedure(this)
    this.leaves?.forEach((leaf: QuadNode) => leaf.process(quadNodeProcedure))
  }

 // Initialise the sub-quads of the current Node,
 // and test if any object fit into a deeper quad

  subdivide(): void {
    // calculate new bounds of sub-quads
    const midW = this.bounds.w / 2
    const midH = this.bounds.h / 2
    const newDepth = this.depth + 1
    this.leaves = [
      new QuadNode(new Rect(this.bounds.x, this.bounds.y, midW, midH), newDepth),
            new QuadNode(new Rect(this.bounds.x + midW, this.bounds.y, midW, midH),
newDepth),
            new QuadNode(new Rect(this.bounds.x, this.bounds.y + midH, midW, midH),
newDepth),
        new QuadNode(new Rect(this.bounds.x + midW, this.bounds.y + midH, midW, midH),
newDepth)
```

```
    ]

    // place current particles into newly created groups
     // removes the object from the current array if it fits into another node

    this.quadObjects.forEach((object: QuadObject) => {
      this.leaves?.forEach((leaf: QuadNode) => {
        if (leaf.insert(object))
          this.quadObjects.splice(this.quadObjects.indexOf(object), 1) // remove from the current
level
      })
    })
  }

// Inserts an object into the deepest point of the QuadTree it belongs
// returns whether the object fit into the bounds of the currently attempted quadNode

  insert(quadObject: QuadObject): boolean {
    // test if the quad bounds contains the object
    if (!quadObject.insideRect(this.bounds))
      return false

    // directly insert if max depth is reached
    if (this.depth >= QuadTree.maxDepth)
      return !!this.quadObjects.push(quadObject) // length should always be non-zero for push
-> truthy

    // Node is safe to push object into
    // first try the leaves
    if (this.leaves)
      for (const leaf of this.leaves)
        if (leaf.insert(quadObject))
          return true
```

```
    // if no leaves, or leaves fail to cover object, push current node
    this.quadObjects.push(quadObject)

    // test if max capacity for the node has been reached
    if (!this.leaves && this.quadObjects.length > QuadTree.capacity)
      this.subdivide() // divide and redistribute

    // object has been placed into an array by this point
    return true
  }
}

 // Root Reference for a QuadTree
 // primary interface for operations

export class QuadTree {
  static maxDepth = Math.ceil(Math.log2(1000 / 5) / 2) + 1 // default for as small as 5 pixels
on a 1000x1000 grid
  static capacity = 5
  public quadRoot: QuadNode
  constructor(
    public bounds: Rect,
    public quadObjects: Array<QuadObject>) {
    this.quadRoot = new QuadNode(bounds, 0)
  }
    // Updates the QuadTree will most recent object positions and then recursively calls a
procedure
   // @param quadNodeProcedure function to call on each node of the tree

  process(quadNodeProcedure: (quadNode: QuadNode) => void): void {
    this.quadRoot.clear() // refresh the QuadNodes
     this.quadObjects.forEach((quadObject: QuadObject) => this.quadRoot.insert(quadObject))
// insert updated objects
    this.quadRoot.process(quadNodeProcedure) // call any user-defined, per-node procedure
```

```
 }

  // Inserts a QuadObject into the deepest level of the QuadTree it belong
  // @param quadObject object to insert into the QuadTree

 insert(quadObject: QuadObject): void {
   this.quadObjects.push(quadObject) // update master object array
   this.quadRoot.insert(quadObject) // descend object into tree
 }
}
```

- - **Package.json (Dependencies used) - -**

```
"dependencies": {
   "@material-ui/core": "^4.11.2",
   "@material-ui/icons": "^4.11.2",
   "next": "10.0.5",
   "react": "17.0.1",
   "react-dom": "17.0.1"
 },

 "devDependencies": {
   "@types/node": "^14.14.20",
   "@types/react": "^17.0.0",
   "@typescript-eslint/eslint-plugin": "^4.12.0",
   "@typescript-eslint/parser": "^4.12.0",
   "eslint": "^7.17.0",
   "eslint-plugin-react": "^7.22.0",
   "gh-pages": "^3.1.0",
   "sass": "^1.32.2",
   "typescript": "^4.1.3"
 }
```

7.2 Results

7.2.1 Homepage

- Figure 7.2.1 illustrates the homepage of the web application. Here we can visualize a QuadTree with the data points and the different divisions of the QuadTree.

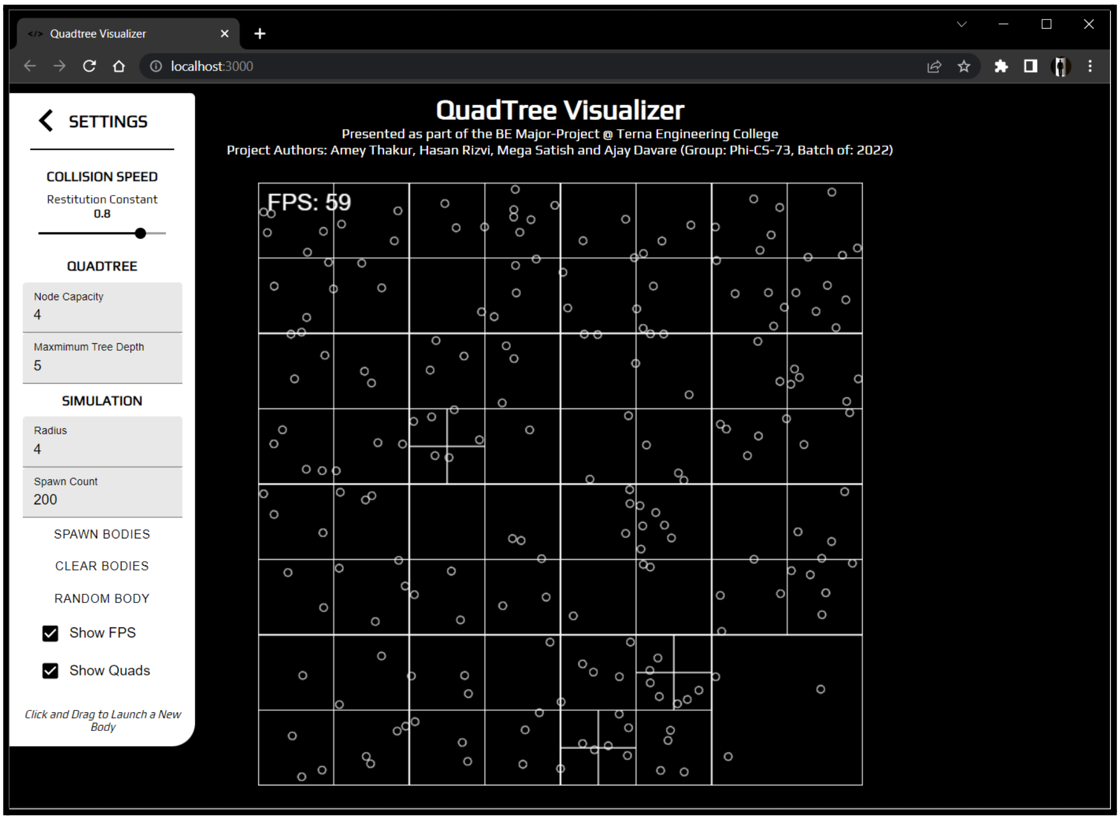


Figure 7.2.1: Homepage

7.2.2 Clear Quadtree

- Figure 7.2.2 shows a clean quadtree without the data points. Since there are no points, we can see only the square.

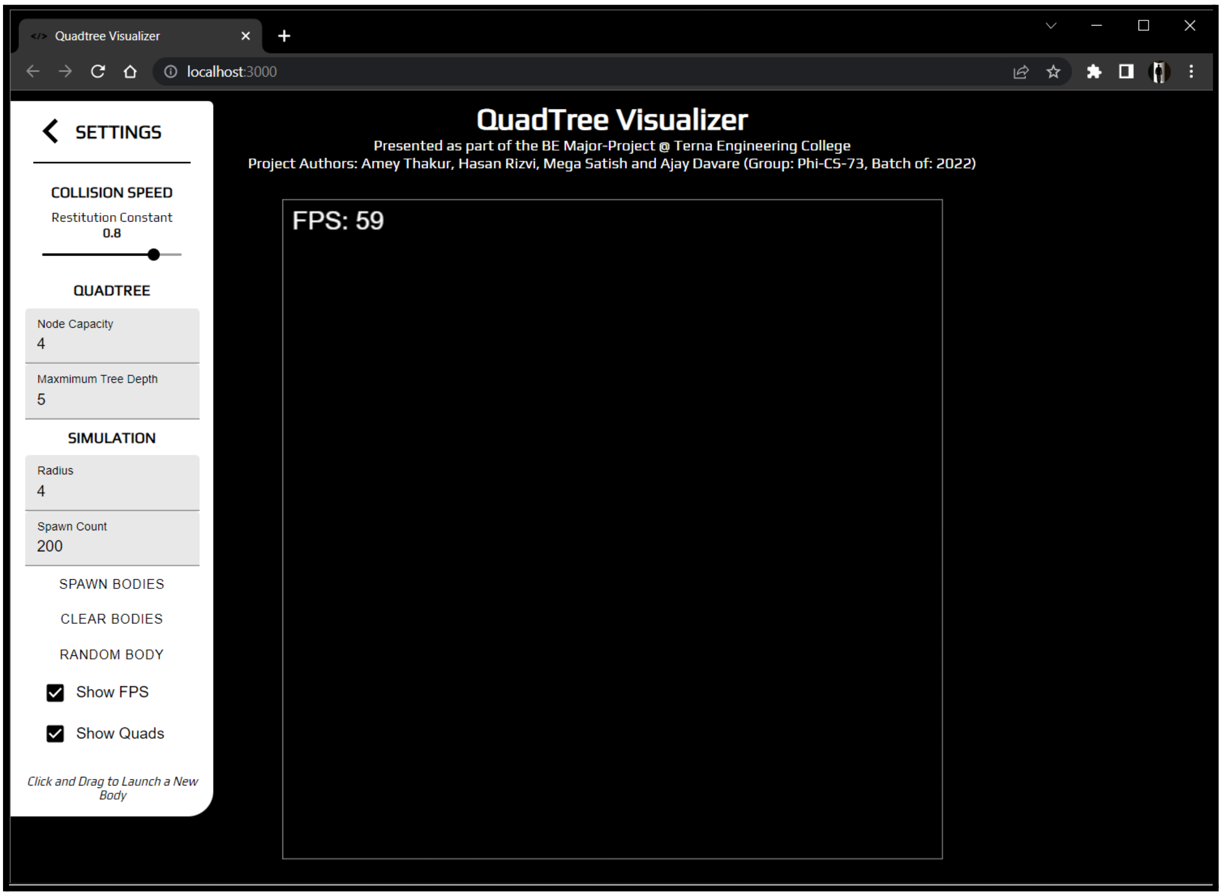


Figure 7.2.2: Clear QuadTree

7.2.3 Spawn Bodies

- Figure 7.2.3 depicts the spawn circles in the quadtree. Here we can see the clear division of the quadtree.

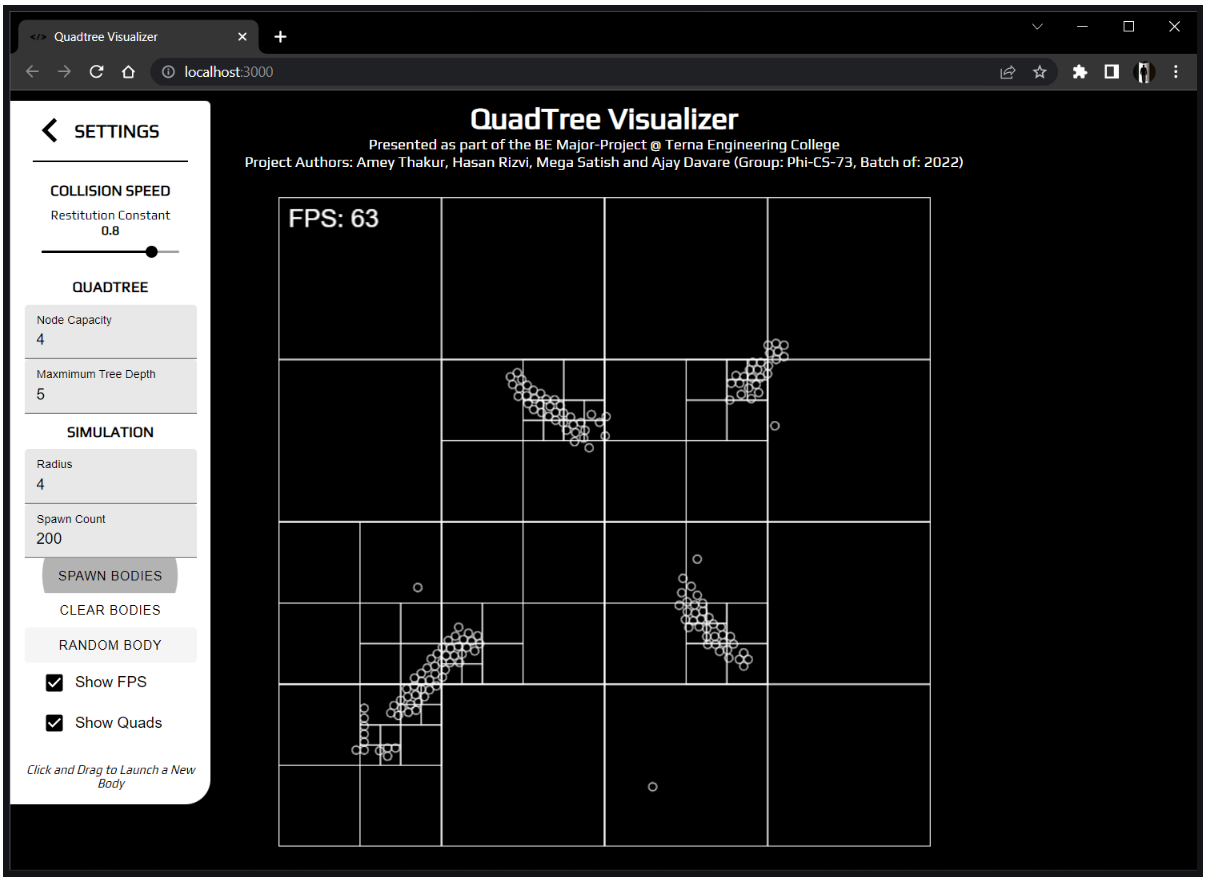


Figure 7.2.3: Spawn Bodies

7.2.4 Random Bodies

- Figure 7.2.4.1 & Figure 7.2.4.2 show the random bodies generated randomly in the QuadTree

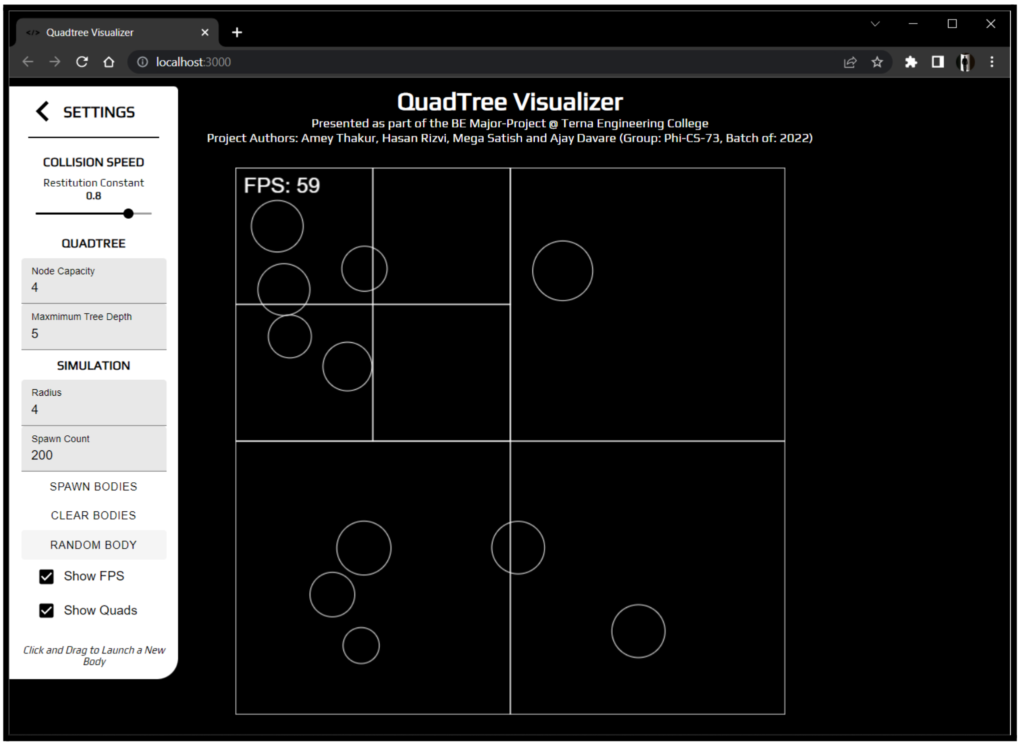


Figure 7.2.4.1: Random Bodies

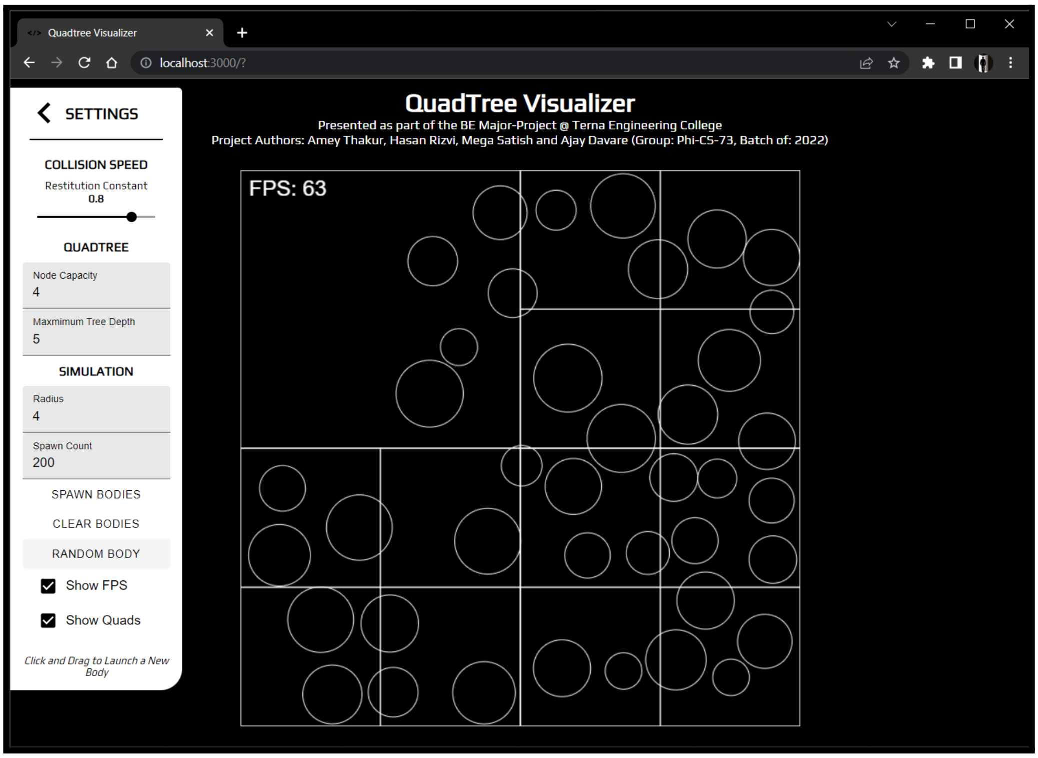


Figure 7.2.4.2: Random Bodies

7.2.5 Random & Spawn Bodies

- Figure 7.2.5 shows the combination of both random and spawn bodies in the QuadTree.

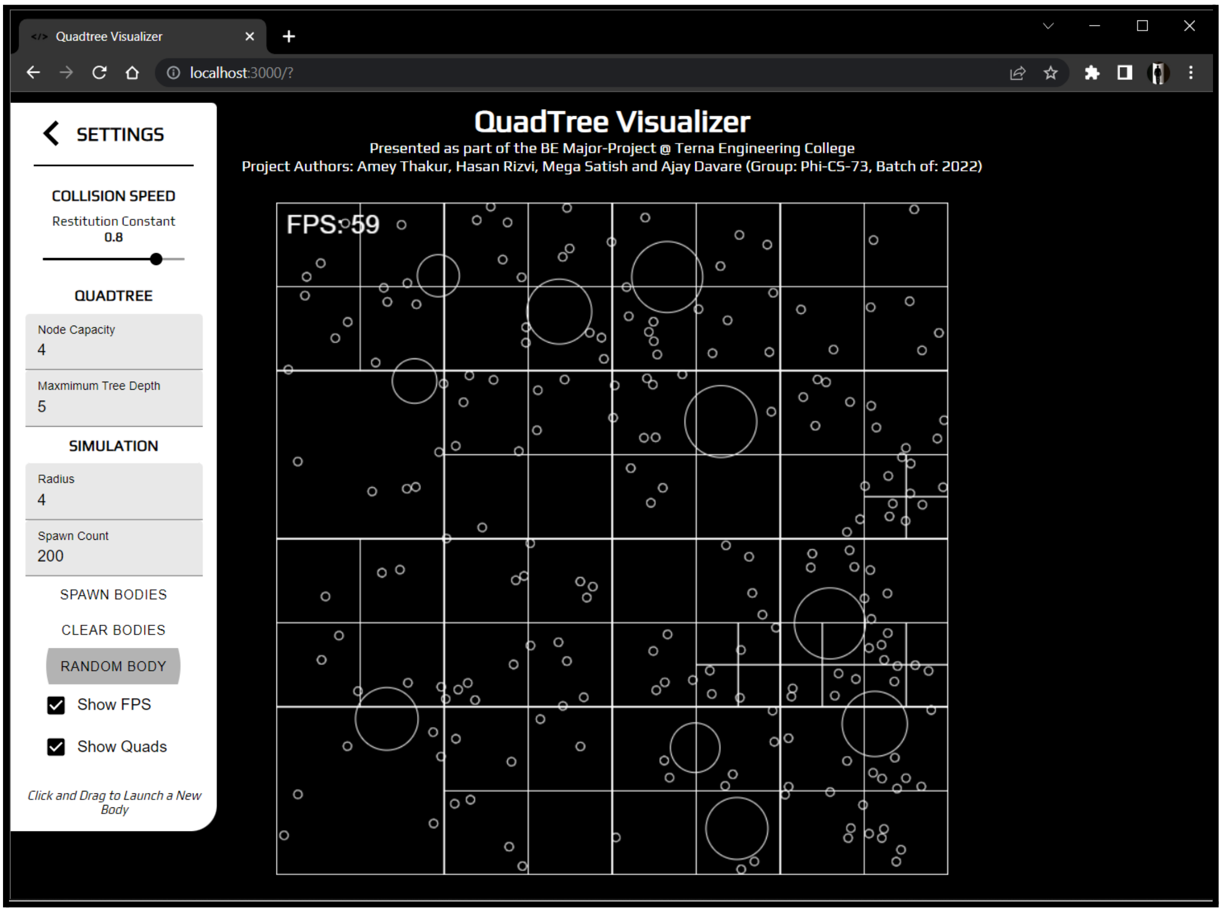


Figure 7.2.5: Random & Spawn Bodies

7.2.6 Control Panel

- Figure 7.2.6 illustrates the control panel. The control panel here is used to simulate different environments in the quadtree such as types of bodies, the coefficient of restitution and the frames per second of the movement of the bodies.

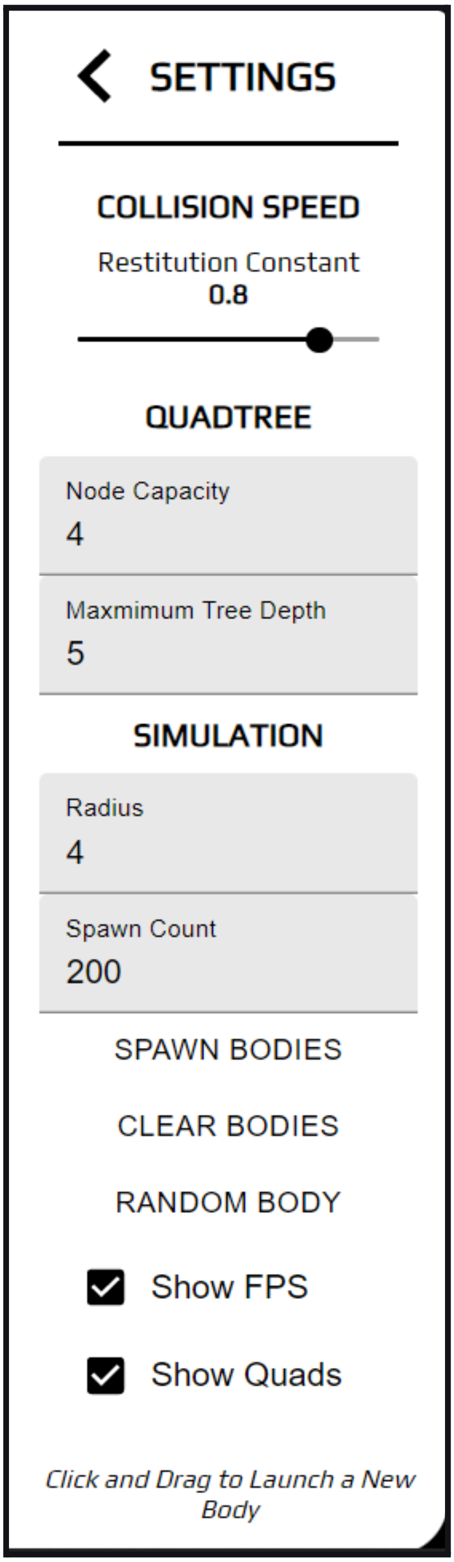


Figure 7.2.6: Control Panel

Chapter 8
Conclusion

We explored a type of tree data structure named Quadtree, that can be used to represent 2-D spaces. In this process, we learnt how/why they are used in a range of applications from scaling up internet services to handle millions of requests per minute to their ever-present use in geolocation-based services like Maps and how we can build applications/libraries to implement the same in our apps/services. It can be concluded quadtrees are extremely powerful data structures that are still heavily under-utilised in both the industry and community applications. By the time of completion of this project we've learned to develop scalable and reusable codebases for large projects, understood the fundamentals of API build and interaction and understood function in a time-bound manner and collaborate at scale across various tasks and disciplines.

References

[1] Q. Cai and Y. Zhou, "*A quadtree-based hierarchical clustering method for visualizing large point dataset*", 2016 Sixth International Conference on Information Science and Technology (ICIST), 2016, pp. 372-375, DOI: 10.1109/ICIST.2016.7483441.

[2] "*An effective way to represent quadtrees*", Communications of the ACM, Volume 25, Issue 12, Dec 1982 pp 905–910, doi:10.1145/358728.358741.

[3] "*Optimal quadtree construction algorithms*" Computer Vision, Graphics, and Image Processing, Volume 37, Issue 3, March 1987, pp 402–419, doi:10.1016/0734-189X(87)90045-4.

[4] Tilkov, Stefan, and Steve Vinoski. "*Node. js: Using JavaScript to build high-performance network programs*." IEEE Internet Computing 14, no. 6 (2010): 80-83.

[5] Fenton, Steve, Fenton, and Spearing. "*Pro TypeScript*." Apress, 2014.

[6] Cantelon, Mike, Marc Harter, T. J. Holowaychuk, and Nathan Rajlich. "*Node. js in Action.*" Greenwich: Manning, 2014.

[7] Wieruch, Robin. "*The road to react: Your journey to master plain yet pragmatic react. js*." Robin Wieruch, 2017.

[8] Thakkar, Mohit. "*Next. js.*" In Building React Apps with Server-Side Rendering, pp. 93-137. Apress, Berkeley, CA, 2020.