Name: Nathan Hutchins
Class and Semester: CSPB2270 , Spring

Project: Spatial Distance using Quadtrees

In many simulations and computer graphics there are millions of vertices, particles, or objects being calculated every second. A common issue that this causes is slowness. Imagine we want to simulate a school of fish. The fish object needs to find other fish objects nearby and correct its swimming direction and speed accordingly to create that schooling effect. If there is no proper data structure set in place, the computer would need to loop through every fish object created and then keep track of the fish within a set range. As expected, this is incredibly slow and has an exponential growth. Every fish needs to calculate the distance from every other fish which is an $O(n^2)$ process. Obviously this is very inefficient and a faster way to help create this schooling effect is by using a Quadtree data structure.

Quadtrees are "used to partition a two-dimensional space by recursively subdividing it into four quadrants or regions."[1] They have a max capacity limit set before inserting elements. When there are too many objects/particles/fish in a region, the tree subdivides into four smaller sub trees. We can then query a region of space, find all sub quadtrees in that region, and then check for all the objects/fish within those subtrees. This drastically reduces the amount of calculations needed when checking for things like distance between objects, collisions, density, and many other use cases. Instead of checking this distance between every fish created, we can instead check the distance between only the handful of fish within certain subtrees selected.

All italicized and underlined functions will be private.

Data Structs and Classes:

- Particle
  - Stores the x,y position of the particle
- Bounding_Box
  - Stores the center position of the box (which will be a particle object)
  - The dimensions of the box
- QuadTree
  - 4 pointers to sub quadtrees
  - Array of Particles in the current node
  - Bounding_Box (this is the area the node covers)
  - Capacity (amount of particle objects allowed in the node before splitting)

Methods (functions in the objects):

Bounding_Box:

- *Contains(Particle)*
  - Returns true or false depending if the particles is in the box
- *Intersects(Bounding_Box)*
  - Check if the given box intersects with another box. (Used for query when finding boxes in a user determined range)

QuadTree:

- *Subdivide()*

[1] https://en.wikipedia.org/wiki/Quadtree
https://www.youtube.com/watch?v=OJxEcs0w_kE&ab_channel=TheCodingTrain

        - This will be called from insert when the number of particles in a section is greater than the capacity. Then move all particles into subtree nodes
    - Insert(Particle)
        - This will insert a particle into the tree node
    - Query(Bounding_Box)
        - Given a range, return all the particles in that range.

Testing: I plan to create a couple different unit tests that will do the following:

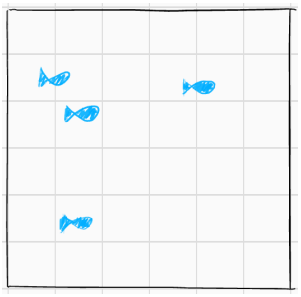
1. Place a certain number of particles at random
    a. Be sure the correct numbers were added to the tree
    b. Be sure no node has more particles than the capacity set
2. Place particles in specific x,y positions knowing how many subdivides this should create
    a. Check if the tree subdivided correctly
    b. Check if the particles were moved to the sub nodes correctly
    c. Check the capacity and total count of particles
3. Create a simple tree knowing what the sub trees and particles should look like
    a. Query the tree for certain ranges and test if the correct particles come back

Why Quadtrees: There are some other data structures that could be used for this type of problem. Octrees are very similar but split into 8 subtrees. The reason I'm not using this data structure is because it is commonly used for 3D spaces and I only want to focus on a 2D space for this project. A binary data structure could work also, but that would only split along one axis, so it won't be as fast or efficient as evenly subdividing into four regions.

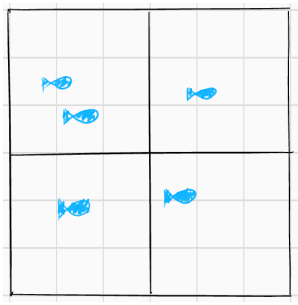
Ideally I would like to be able to visually test this by plotting simple points on a flat surface. However that might be a little bit of a stretch goal for the timeline we are aiming for. So this is only tentative.

Visual Examples

Full Node with 4 fish.



Adding a 5th fish overflows the tree causing it to subdivide.



Adding more fish and showing how a quadtree looks when displayed like other trees.