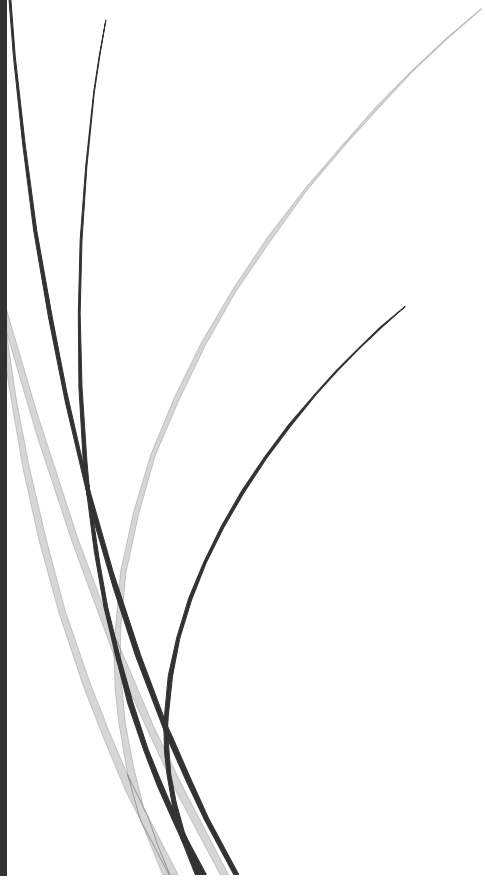
01-12-2021

ESINF Report

LAPR3 INTEGRATIVE PROJECT

SPRINT 2



Group 96:

- ✓ Miguel Morais (1181032)
- ✓ Rodrigo Rodrigues (1191008)
- ✓ Tiago Ferreira (1200601)
- ✓ Mariana Lages (1200902)
- ✓ Eduardo Sousa (1200920)
- ✓ Miguel Jordão (1201487)

Class Diagram Explanation

For this sprint, we implemented a two-dimensional tree so we could easily search for a specific port in the program. Its nodes have two coordinates, the x and the y, which will be the longitude and latitude coordinates of the port, respectively. The Node class also has two comparative methods, one comparing the longitude of two ports and the other the latitude. Our two-dimensional tree has a few methods implemented. However, we will only explain the ones we used for this sprint.

Starting with the insert method, which inserts the ports into the tree. The method inserts the port into the two-dimensional tree, the leftmost element having the smallest x and y, and the rightmost element having the highest x and y. The method uses the class MedianElement, which divides a group of ports into five groups and tries to find the median of each group, then the program compares all the medians so it can find the median of those five medians. After having the middle median, the class provides methods to organize the rest of the medians by the smallest to the biggest, organizing them left to right.

The two-dimensional tree has also the nearestNeighbour() method implemented, which searches the nearest port given a ship's MMSI. The method compares each node distance between each existing port and the ship's current position using recursion.

For this sprint, we had to implement the class PortImporter, which imports ports from a file in a specific format, saving each one of them into the two-dimensional tree.

Most of the User Stories implemented use SQL scripts to work and the connection to the data base.

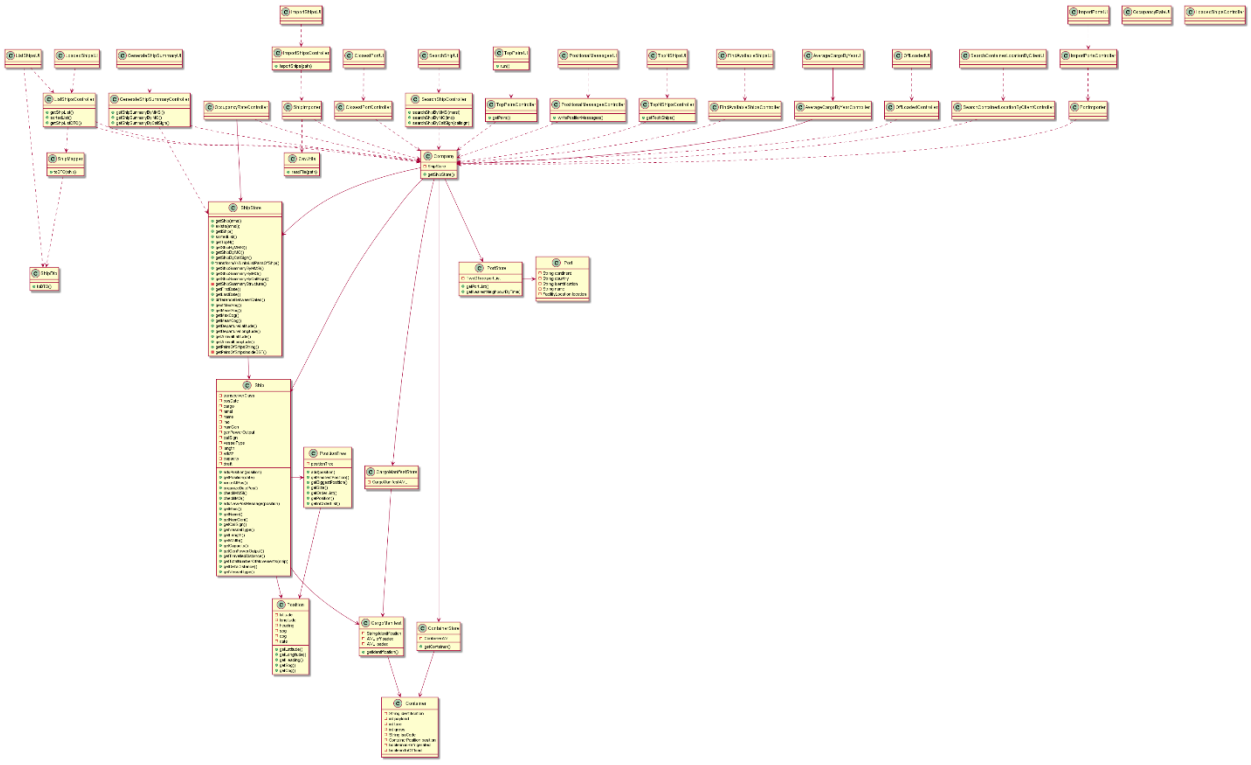


Figure 1 Global Class Diagram - Sprint 2

US201 Complexity Analysis

In US201, it is asked to import all the ships to a balanced kd-tree. Since the tree rotation technique can't be used to balance the tree, the way we did it is to order the list of points to insert in order to select the median of the points given the dimension of the node we want to introduce (example: in a 2d tree, the first level is ordered by the x coordinate, the second level by the y coordinate, the third level by the x coordinate and so on) and insert this point. Another and also better way, in terms of complexity, is to use a selecting technique, like quickSelect, to pick the median element and insert that median element in the node. The original way to insert in the balanced tree was to use the quickSelect in order to get the median element. The approach initially used by the team used a variation of quickSelect where, instead of using a random pivot, we used the median as a pivot, as seen in https://en.wikipedia.org/wiki/Median_of_medians. This approach as complexity O(n), making the insert of complexity $O(n(log(n)))$ (inserting in the kd-tree is $O(\log(n))$ since the original array is divided in chunks of 5 elements. To find the median of this chunk, we need to put all the median in another list and find the median of this other list. However, we had troubles in implementing this approach given the lack of time and amount of work in this sprint, so we opted to use the first approach even knowing that, in the median case, it is less efficient). The approach we used has complexity of $O(n(log(n))^2)$. The ordering algorithm we used is the Timsort and it has a complexity of $O(n(\log(n)))$ since inserting in the kd-tree is $O(\log(n))$ and, inside each insertion, we order the array, making the complexity of our method having a complexity of $O(n(log(n))^2)$.

US202 Complexity Analysis

In US202, it is asked to find the closest port of a ship given its CallSign, on a certain DateTime. To find the nearest port, we used the method NearestNeighbor(), which, by having the last positional message of the ship, it calculates the closest port of that positional message. In order to do that, the program calculates the distance from each port to the ship's position by using recursion. To compare each distance, we have two variables called closestDist and closestNode, which are compared to each distance. If that distance is a smaller value than the actual closestDist, the closestDist becomes that smaller distance and the closestNode becomes the current node in that recursion. The disCoor variable is used to compare the current node x or y value with the ship's position x or y, depending on the divX boolean variable which makes the program decide if it is going to be calculated either by the x or y value. ClosestDist will be negative if the current node x or y value is bigger than the position's ship or positive if the position's ship x or y value is bigger. The next nodes will be decided by the program depending on the disCoor and disCoor2 values. In the worst-case scenario, it is necessary to check the whole tree and find the closest port, which, in this case, will have a $O(n)$ complexity. If the closest node is the root itself, the program will still run the whole 2D-tree, however, the complexity will only be $O(\log(n))$.

- Worst case: $O(n)$
- Best case: $O(log(n))$

Contributions

- US201
  - Code: Eduardo Sousa (1200920)
  - Tests: Eduardo Sousa (1200920)
  - Complexity Analysis: Eduardo Sousa (1200920)

- US202
  - Code: Tiago Ferreira (1200601), Miguel Jordão (1201487)
  - Tests: Tiago Ferreira (1200601), Miguel Jordão (1201487)
  - Complexity Analysis: Tiago Ferreira (1200601), Miguel Jordão (1201487)