Relatório de ESINF – Sprint 2

Turma 2DO _ Grupo 151

1200605 _ André Ferreira

1180611 _ Diogo Silva

1180682 _ Francisco Ferreira

1190708 _ João Ferreira

1200625_ Sérgio Lopes

Data: 01/12/2021

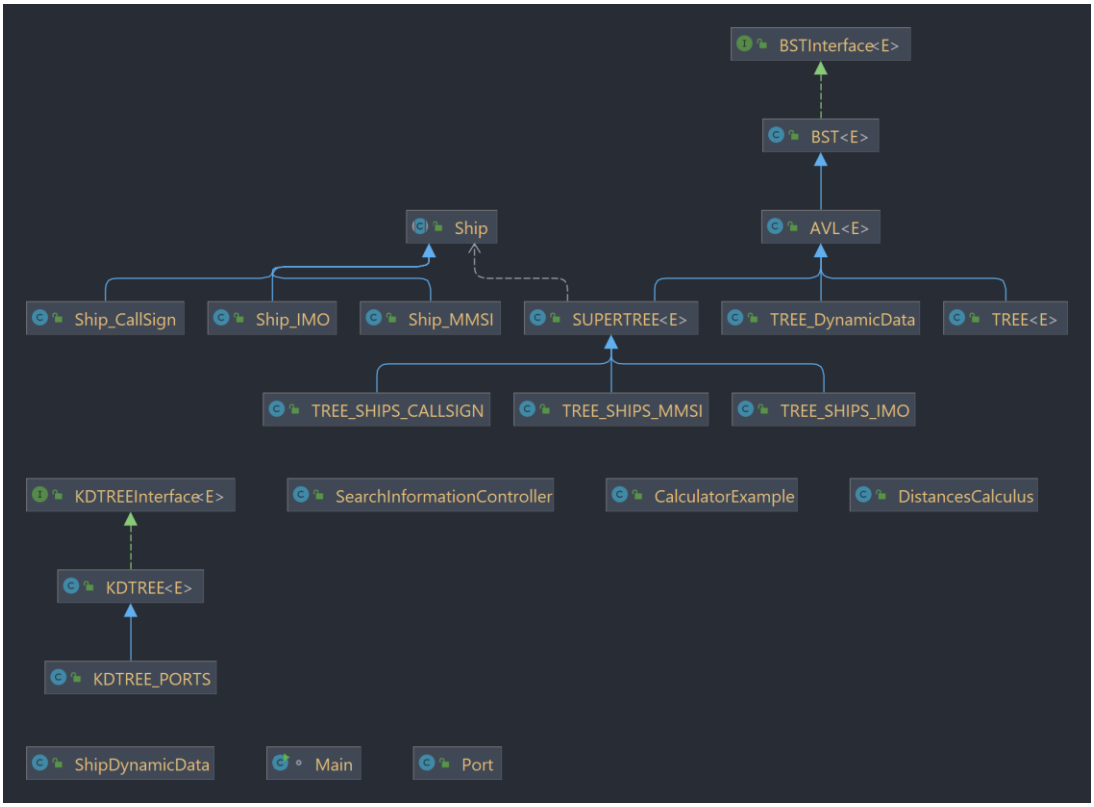
Índice

Parte I – Introdução e estrutura do trabalho

I.1 Introdução ao Sprint 2

Neste Sprint 2, o foco na área de trabalho do grupo manteve-se na criação de uma **BST 2D-Tree.** A 2D-Tree advém da **kd-tree** que é uma árvore binária em que cada nó é um ponto k-dimensional, neste caso, 2D. Cada nó não-folha pode ser considerado implicitamente como um gerador de um hiperplano que divide o espaço em duas partes, conhecido como semi-espaço. Os pontos à esquerda do hiperplano são representados pela subárvore esquerda desse nó e pontos à direita do hiperplano são representados pela subárvore direita. A direção do hiperplano é escolhida da seguinte maneira: cada nó na árvore é associado a uma das k-dimensões, com o hiperplano perpendicular a esse eixo dimensional.

I.2 Diagrama de Classes



Parte II – Desenvolvimento do Sprint 2

II.1.a. Planeamento do 1º User Story

Para a resolução da primeira User Story ("Criar uma BST 2D-tree com as localizações dos portos (ports.csv)."), é necessário criar uma ***BST 2D-Tree*** de modo a guardar os dados que constam nos ficheiros "bports.csv" e "sports.csv". Para tornar isso possível criamos a classe ***KDTREE e KDTREE_PORTS***, onde irá guardar todos os dados dos portos presentes nos ficheiros, tais como o continente, país, código, latitude e longitude do porto. Além disso, foi também criado a classe genérica para os portos, ***Port***, de modo a aceder mais facilmente a estas características dos portos.
Na classe ***KDTREE_PORTS,*** o método responsável pela criação da KD-Tree é a *createKDTree()*, realizando assim o pedido pelo exercício 1 do Sprint 2.

II.1.b. Análise de complexidade do 1º User Story

Depois de analisar a complexidade do método de criação, o grupo concluiu que o método createKDTree() que é o método que cria a a 2D-Tree tem uma complexidade de O(n^2). Isto porque o *while* tem como complexidade O(n) e o método insertNodes que se encontra dentro deste create tem como complexidade O(n^2), sendo que no final: O(n) + O(n^2) = O(n^2).

II.2.a. Planeamento do 2º User Story

No exercício 2, era necessária uma funcionalidade capaz de encontrar o porto mais próximo de um navio dado o seu **CallSign**, numa determinada data. Para isso, foi criado um método na classe ***KDTREE***, chamado de *findNearestNeighbor()*, que recebe como parâmetro um node do tipo Node2D<E> que ao início é a *root*, dois doubles (x e y) que serão o x e o y do nó, um *closestNode* do tipo Node2D<E> que ao início, ao correr este método, é a *root* e um boolean chamado de *divX*, que ao início declaramos como true, para saber que o nível inicial onde é comparado pelo x, sendo que para o próximo nível passa a false para se comparar pelo y.

Este método tem como objetivo retornar ao utilizador o porto mais próximo da mensagem do navio com a data que o utilizador escolheu, ou se não tiver nenhuma mensagem com uma data que coincida com a data que o utilizador escolheu, é retornada a mensagem mais recente anterior à data escolhida pelo utilizador, conseguindo assim obter o resultado esperado.

II.2.b. Análise de complexidade do 2º User Story

O grupo concluiu que a complexidade do método criado para satisfazer a 2ª User Story é O(n) pois estamos a tratar de um método recursivo, justificando assim a complexidade.

Contribuição de cada membro do grupo

Tal como o Sprint 1, todo o trabalho (estrutura geral, user stories e testes) contou com a cooperação de toda a equipa para a resolução do mesmo, de modo que todos ajudaram para a resolução das duas user stories, tanto como dos testes e da análise da complexidade.