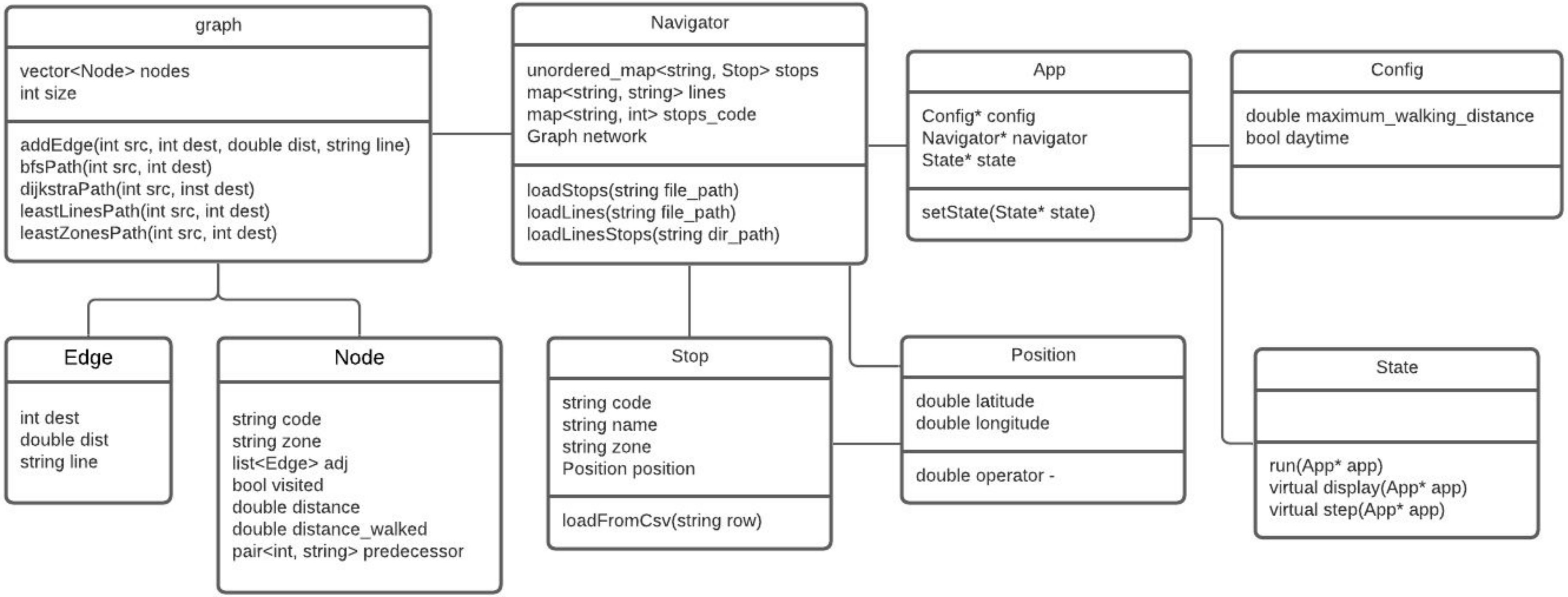
Transport - Navigator

Turma 9 Grupo 69

- André Barbosa up202007398
- Guilherme Almeida up20200886
- José Luís Rodrigues up202008462

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Class Diagram



Dataset

All the file operations are done in *Navigator and occur when the program starts*. The loading happens as such:

- Read all the stops from *stops.csv* and store them in a map<string, int> and an unordered_map<string, Stop>.
- Read all the lines from *lines.csv* and stores them in a map<string, string>.
- For every line in the line map, load each corresponding file and store all the connections in the graph.

Graph

The program uses only **one** graph. It contains every stop and all the connections associated with each stop. The graph as a vector of nodes and each node represents a stop. To access a stop, first get the corresponding int from the stops map.

Using maps allows **insertions** on the graph in **O(1)** time and **lookups** in **O(1)** average.

Origin/Destination

The program allows the user to either start on a specific stop or input a position (with coordinates).

If the user chooses by **stop**, the program displays all the lines so the user can look for the stop he/she wants.

If the user inputs a **coordinate**, the closest stops to that location are suggested. Finding the nearest stops takes O(n) time.

Different Path Options

After deciding the origin and destination stops, the user is allowed to choose which path does suit best his needs.

- **Fewest stops** - path with the least amount of stops, which is generated by a breadth-first search algorithm with time complexity of O(V + E).

- **Least distance** - path that has less length, generated by the algorithm of Dijkstra optimized by the use of *minHeap* provided in class. Time complexity of O(E log|V|).

Different Path Options

- **Least line changes** - path that uses the least amount of lines, generated by the algorithm of Dijkstra, with time complexity of O(E+log|V|).

- **Cheapest path** - path that passes by the least amount of zones, generated by the algorithm of Dijkstra, with time complexity of O(E+log|V|).

Stop changes

The program supports stops changes mid-path. Sometimes, to generate a better option, it is worth it to walk to a nearby stop.

In the settings tab the user can define the **maximum walking distance**. This means that the program is not gonna suggest stop changes that exceed that amount. Changing the distance has a time complexity of O(n^2), but it’s faster for shorter distances.

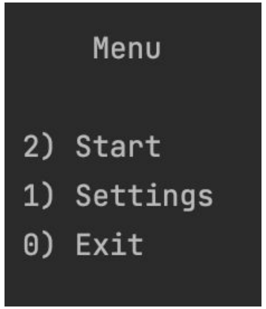
Additional Settings

Besides the functionalities mentioned before, the program also allows:

- The **time of the day** is defined by the user. This means only lines in the correct time frame (night or day) are used.
- The user has the option to **block** lines and stops. When something is blocked, no path will go through there.

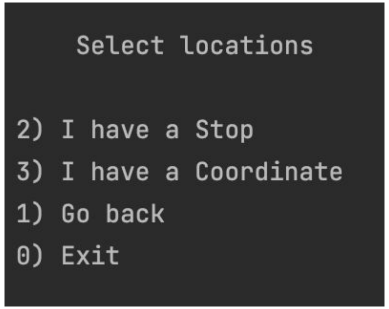
Menu Driven Interface

The program offers a simple Menu driven user friendly interface. To navigate through the several states/menus the program prompts different options for the user to choose between.



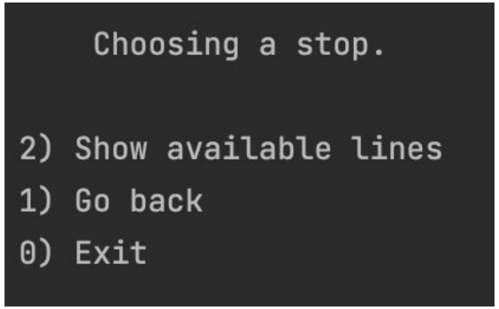
```
     Menu


2) Start
1) Settings
0) Exit
```



```
     Select locations


2) I have a Stop
3) I have a Coordinate
1) Go back
0) Exit
```



```
   Choosing a stop.


2) Show available lines
1) Go back
0) Exit
```

Best Feature

The program ended up being very versatile. The user has a whole variety of option to create a path. Besides the four different path options, there’s even different restrictions that can be applied so the calculations in general are more comprehensive.

Difficulties

This project was a good opportunity to implement the algorithms discussed in class. Handling real data gave the project a more practical sense.

On the other hand, the short time frame given for the development increased it’s difficulty significantly. Besides being able to handle the data and perform calculations, the program also includes a user interface, which added to the workload.