COMP3888 - phase 3.1

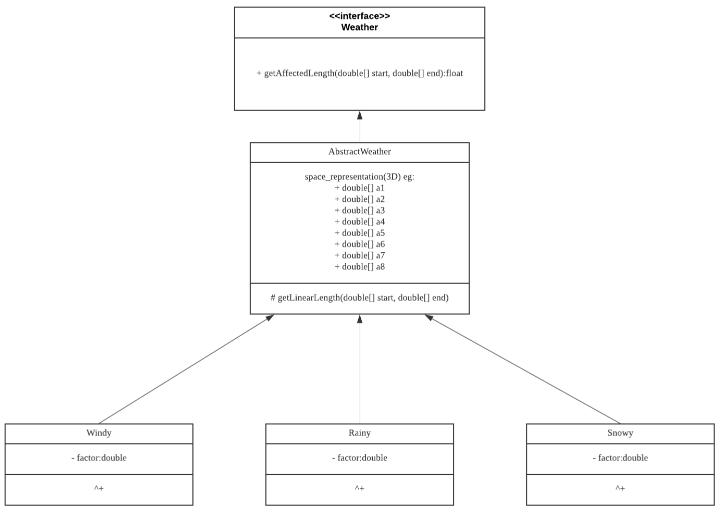
October 2020

1 Introduction

It is the time to include weather in our simulation! This document describes the implementation guidelines.

2 Classes

Weather, as specified in phase 3, the best way to represent them should be using polynomial object oriented programming. Below provides an simple UML which illustrate this idea.

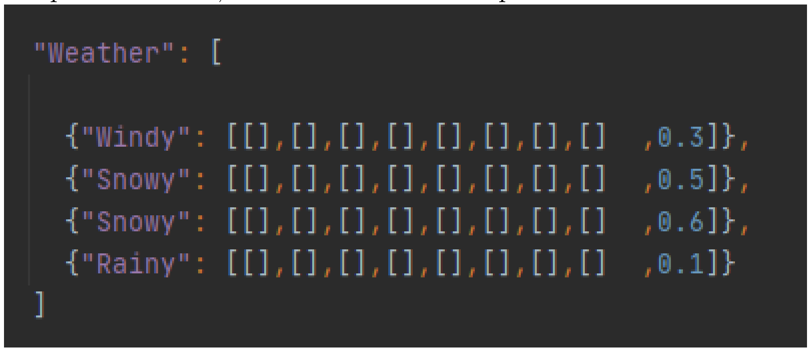


- **getLinearLength** method uses mathematical way to calculate the length of a line which passes though the area. E.g. if a weather affected area is represented as a cube like (0,0,0), (0,1,0), (1,0,0), (1,1,0), (0,0,1), (0,1,1), (1,0,1), (1,1,1). Then, getLinearLength((-1,-1,-1),(2,2,2)) should return 1.73.

- **getAffectedLength** method calculates the **EXTRA** virtual distance between two 3D points by taking into consideration of the weather. E.g. using the affected length calculated above, the **extra** vitual distance can possibly be 0.2. So, $getAffectedLength(...) = (...) * 1.73 = 0.2$. Now, the path cost can be updated to $1.73 + 0.2 = 1.9$
- Imagine multiple weather objects affecting different parts of a path. And we have many paths. This is how this is going to work.
- The factors stored in concrete objects are highly depending on simulator or actual world. It shall be provided via JSON file, same as the type of weather.

3 JSON

The main purpose of content adding to JSON is to specify which type of weather object to create, and its internal influence factor. Compare this example with the previous UML, it shall be trivial to implement.

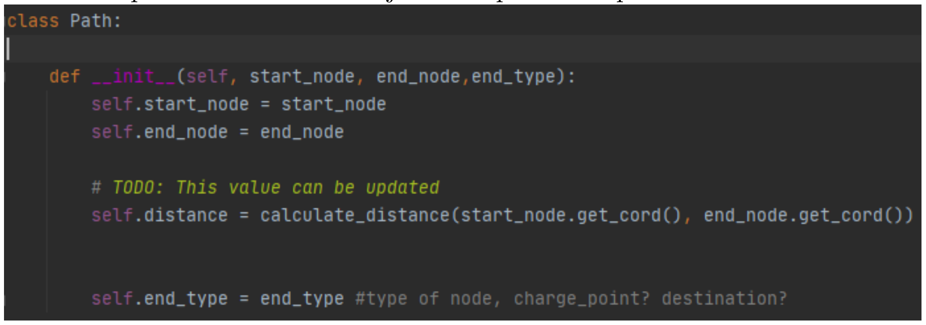


```
"Weather": [

  {"Windy": [[],[],[],[],[],[],[],[]  ,0.3]},
  {"Snowy": [[],[],[],[],[],[],[],[]  ,0.5]},
  {"Snowy": [[],[],[],[],[],[],[],[]  ,0.6]},
  {"Rainy": [[],[],[],[],[],[],[],[]  ,0.1]}
]
```

4 Code

Use the implemented weather objects to update the paths.



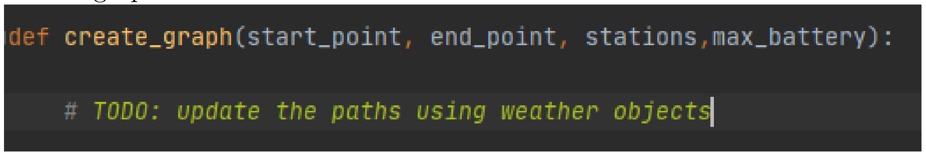
```python
class Path:

    def __init__(self, start_node, end_node,end_type):
        self.start_node = start_node
        self.end_node = end_node

        # TODO: This value can be updated
        self.distance = calculate_distance(start_node.get_cord(), end_node.get_cord())


        self.end_type = end_type #type of node, charge_point? destination?
```

And in graph construction as well



```python
def create_graph(start_point, end_point, stations,max_battery):

    # TODO: update the paths using weather objects
```