Informatic Large Practical

Coursework 2 Report

Jingkai Yuan

December 3, 2021

**Part 1**

**Software architecture description**

**1.1 Overview**

This part describes the software architecture of this drone delivery application and explains why these classes are identified as being the right ones.

**1.2 App**

The App is the main function in my java program. As a static function of the program entry, it can be recognized by the JVM and loaded automatically. It will accept input parameters from the user and initialize Database.java, MenuParser.java, and DroneMap.java based on these parameters, and finally initialize Drone.java. Then drone will start preparing the order and finding the flight path. In addition, this class will save the order information and drone’s flight path into the database. Finally, the related information will be printed in the console so that researchers and developers could use them to work in the future

**1.3 Database**

In this project, we need to interact data with the Apache Derby database software. This class contains four functions for reading from and writing to the database (two for each). It will be initialized in the main function and accept parameters (including database port number, day, month, and year).

The reading of the database will be used once the Drone object is initialized in the main function; the first reading function will call the second reading function. In each iteration of the reading database, all the order information will be saved to initialize an Order object.

The writing of the database will be used once the drone has finished the path-finding algorithm and recorded the flight path, then in the main class, it will call the database writing function to save this information into the database.

**1.4 Drone**

The Drone class contains the main methods for drone's path-finding algorithm. This class is mainly responsible for the pathfinding of the drone and stores the flight path so that it can be written to the database later.

In this class, there are two main methods. The first one is to find the flight path for order delivery as far as possible with enough battery. The second one is used once the battery is under 50 or all the orders have been delivered, then it will find the path to go back to Appleton Tower. These two methods heavily rely on the methods that are in the DroneUtils.java, Including selecting landmarks, judging the no-fly zone and confinement area, and processing the order of orders that the drone will deliver, etc. Once the pathfinding algorithm is done, all the points that the drone has arrived at will be saved in a list of type-Point and all the moves from one point to another point will be saved in a list of type-FlightPath.

The life cycle of the drone object will be initialized from the beginning until the end of the entire program.

**1.5 DroneMap**

The DroneMap class contains all fields and methods that are related to the positions on the map. It accesses the files and their content on the website through the class WebConn, obtains the no-fly zones and landmarks, and stores them as lines and points separately. It also contains the method that will store the default values of the confinement area as lines.

In addition, this class also contains a method that is responsible for printing the route and creating the visuals of the drone flight path in GeoJSON format. A text file, "drone-DD-MM-YYYY.geojson", will be automatically generated by this method. The visualization of the flight path can be shown by opening this file on geojson.io.

The life circle of this class will be started before the drone is initialized and it will live until the drone has finished its pathfinding algorithm.

**1.6 DroneUtils**

In class DroneUtils, it contains all the helper functions that will be used in the drone's pathfinding algorithm. The methods for checking if a line will cross the no-fly zone or confinement area will call the methods in the DroneMap to get them represented as lines. The class also contains the methods that will be used before the path-finding algorithm, such as order sorting, initialization of shop, and pick-up locations.

**1.7 FlightPath**

The FlightPath class is responsible for recording the drone's flight path when the drone is doing the pathfinding algorithm. For each move, it will save the start longitude, start latitude, toward the angle of this move, end longitude, end latitude, and the order number to which this move belongs.

The life circle of this class will be started once the drone is initialized in the main function.

**1.8 LongLat**

This LongLat class is responsible for representing the longitude and latitude for the drone, no-fly zones, landmarks, and confinement areas on the map. It also contains the method to calculate the distance between two points and check if two points are close to each other, and contains the method to give the drone's next position, given the direction.

**1.9 MenuParser**

The class MenuParser is responsible for parsing the menus file and its content on the website. It contains an inner class called Menu so that we can use this inner class's attributes to receive the JSON format data on the website.

It contains a method to calculate the price of each order and it will be called once the order is initialized during the database reading. I put this method here is because once the menu parsing is done, we directly use this method inside the MenuParser class to calculate the price of each order and assign this value, instead of putting this method somewhere else.

**1.10 Order**

The class Order contains all the attributes of an order. It includes the order number, delivery date, customer, delivery-to location, order items, pick-up locations, order total price, and a Boolean value to check if this order has been finished or not.

The Order class will be initialized when the data are read from the database, some of the attribute values will be assigned by calling methods from other classes (e.g., the price). The path-finding algorithm will iterate each order and use these attributes the find the drone's flight path to pick up the food and deliver it to the customer.

**1.11 WebConn**

The class WebConn is responsible for creating the HTTP Client, sending HTTP requests, and receiving HTTP responses. Other classes will use this class to get the response and keep processing(parsing) these raw data in JSON format.

**1.12 WordParser**

The class WordParser is responsible for parsing the words file and its content on the website. It contains an inner class called Word so that we can use this inner class's attributes to receive the JSON format data on the website.

Given a three-word string, and it will use the class WebConn to get the corresponding response.

**1.13 UML Diagram**

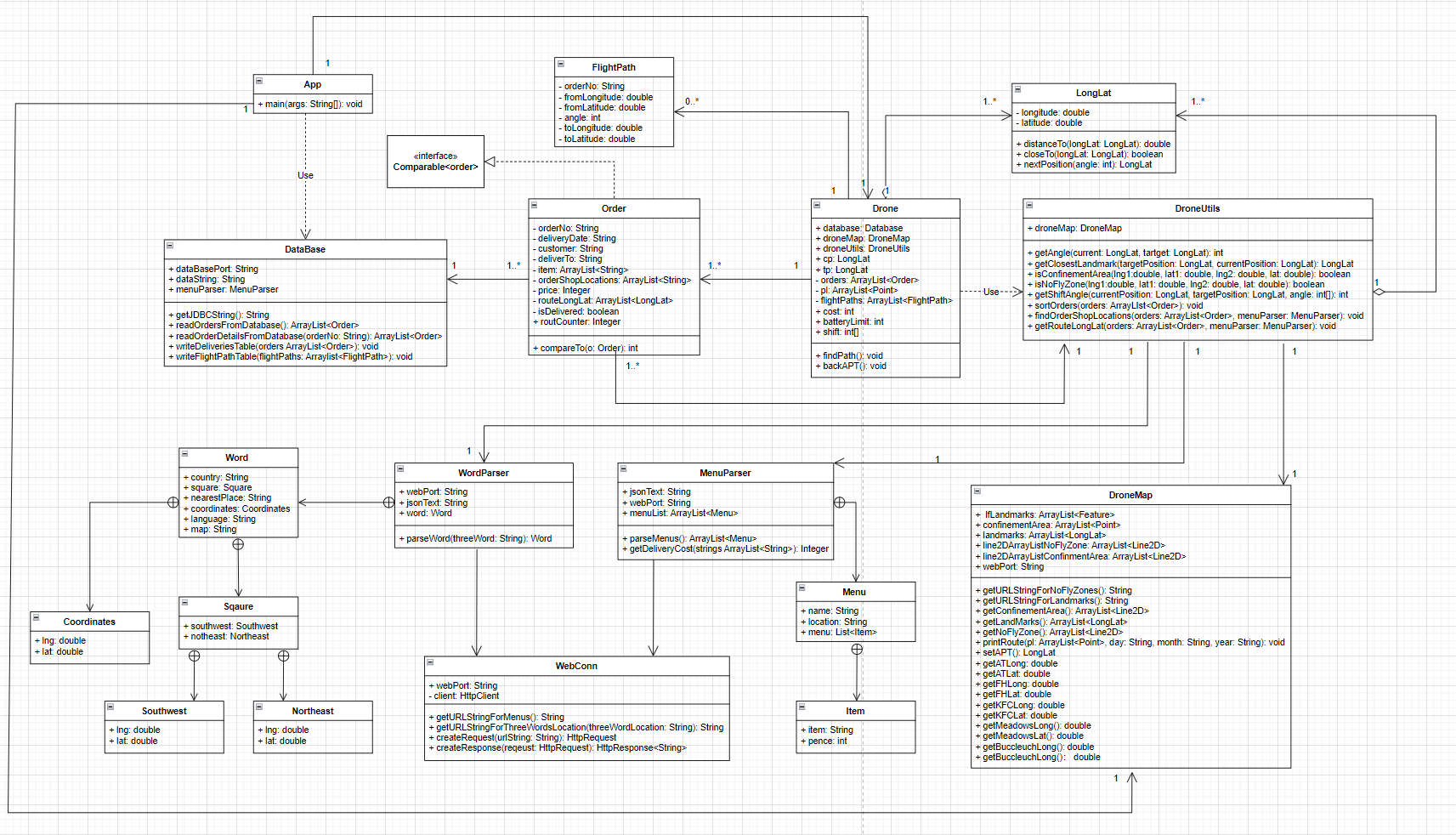


Figure 1-1 The UML diagram

**1.14 Sequence Diagram**

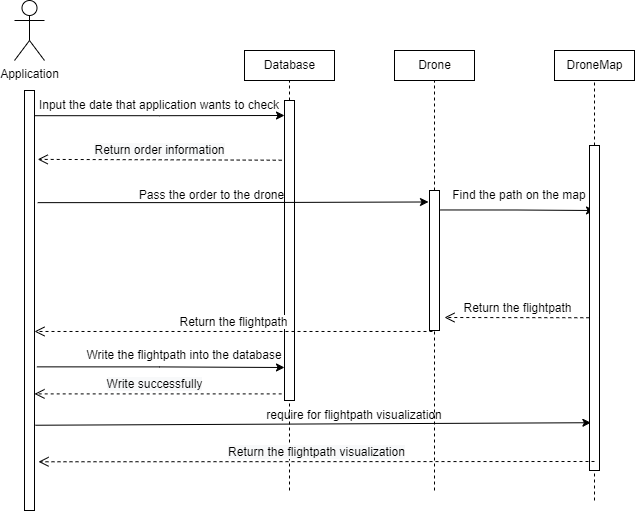


Figure 1-2 The Sequence diagram for querying the flightpath

**Part 2**

**Drone control algorithm**

**2.1 Goal**

The goal of the drone's control algorithm is to maximize the sampled average percentage monetary value. However, there might be a situation that we cannot finish all the orders within the limit of the drone's battery. This means that we need to find an algorithm to complete as many orders as possible with larger amounts within the limited power of the drone.

**2.2 Control Algorithm**

The main drone control algorithm, *findPath()*, is in the class Drone. The algorithm is mainly trying to find the closest path by using the landmarks when the drone's moving from one target to another one will cross the no-fly zone.

**2.2.1 Preparation**

Before the drone starts to iterate each order to find the flight path, the drone will initialize the <Point> list and the <FlightPath> list, setting them ready to record the drone’s motion trajectory.

Since the drone is launched each day from the top of the Appleton Tower, we will set the current position of the drone at Appleton Tower and record the starting point into the <Point> list.

**2.2.2 main structure**

Once the preparation is done, the path-finding algorithm will get started.

**In this algorithm, it contains three loops in total.**

**Loop to iterate each order**

The first loop is to iterate each order, and the second loop is to iterate each target in this order. One thing needs to be notified is that the order’s pick-up locations is sorted in the order of reading from the database which is in a random sequence. According to the document, each order may have at most two food pick-up locations; Therefore, there might be a situation that the sum of the distance from the current position to the second store and then to the first store is longer than the default pick-up locations sequence.

To solve this, we are going to sort the pick-up locations. Firstly, an if-else clause will be used to check how many pick-up locations does the current order have. For the order that has two pick-up locations, we will calculate and compare the distances of these two pick-up locations from the current position respectively. We will sort the pick-up locations in ascending order with respect to distance. Because the current position is a global variable, we can get the drone’s real-time position while a new order is being shipped.

**Loop to iterate each target**

Once we sorted the pick-up locations, the order’s target has been to the ideal sequence. Then we are going to iterate these targets which includes the pick-up location(s) and deliver-to location. In each iteration of target, we will check whether the line connecting the drone’s current position and the target position will pass through the no-fly zone, and there will be two main situations

**1. The line between the target position and current position will cross the no-fly zone**

In this situation, since the drone cannot directly fly to the target position, we will use landmarks to solve this.

According to the document, any point on the map can fly directly to at least one of the two landmarks. Therefore, we will let the drone go to the landmark first and then go to the target position in this iteration.

The logic of selecting a landmark is to determine whether the two landmarks will pass through the no-fly zone from the current position and target position, and then select the one that will not pass through the no-fly zone. If the two landmarks from the current position and target position will not pass through the no-fly zone, for each of the landmarks, we need to calculate the sum of the distances from the current position to landmarks and the distance from the target position to landmarks. Compare two results and choose the landmark that has the lower total distance.

**2. The line between the target position and current position will not cross the no-fly zone**

When the drone can directly go to the target without crossing the no-fly zone, then we will just let the drone go to the target by iterating each move.

**Loop to iterate each move**

This is the innermost layer in the three-layer loop, and it controls every move of the drone.

According to the regulations in the document, the drone can only move every step within the range of 0 to 350 in multiples of 10, this does not allow us to move the drone in a straight line with an invalid angle. Therefore, every time we calculate the angle from the current position to the target position, we will round this angle to a multiple of 10. For instance, if the angle we get is 45, then we will round this angle to 50; Then, since we moved slightly to the left in the correct direction, the next time we calculated the angle may be around 40, hence we will round that number to 40. *(see Figure 2-1)*

图示

描述已自动生成

*Figure 2-1*

Once the angle is determined, we will check each move to see if this step will cross the no-fly zone or the confinement area, if it crosses, we need to correct the angle by shifting the angle +10 or -10 cumulatively. For example, if the current angle is 50 and will let the drone cross the no-fly zone, then we will shift this angle to left which is 50+10 = 60; If 60 is still invalid, then we will shift this angle to right which is 50-10 = 40; If 40 is still invalid, then we will shift this angle to left again with cumulative increment which is 50+20 = 70. This angle shift algorithm will keep looping until a valid angle is found.

In addition, in the case we check that the line between current-target and target-target will cross the no-fly zone, the drone will move to the landmark first and then move to the target position. While the drone is moving from the current target to the landmark, whenever a new move is made, we will check if the drone crosses the no-fly zone from the current position. If it does, we will break the loop to the landmark. This greatly optimizes our path finding algorithm since we can remove those redundant moves. *(see Figure 2-2)*

图示, 示意图

描述已自动生成

*Figure 2-2*

For each move, the drone will record its flightpath and the points it has arrived. Both moving and hovering will cost the drone’s battery for 1. Since the drone needs to fly back to Appleton Tower no matter whether all orders have been delivered. Therefore, in each move, we will check the drone’s left battery. If the battery is under 50, the drone needs to end the current order and use the same method mentioned above to fly back to Appleton Tower (50 battery is enough for drone to fly back at any point in the confinement area).

**2.3 Results**

The example output of this application has been visualized through the *geojson.io*

*(see figure 2-3 and 2-4)*

|  |  |  |  |
| --- | --- | --- | --- |
| *Date Moves* | *Date Moves* | *Date Moves* | *Date Moves* |
| *01-01-2022 299* | *07-07-2022 727* | *01-01-2023 959* | *07-07-2023 1065* |
| *02-02-2022 302* | *08-08-2022 810* | *02-02-2023 954* | *08-08-2023 1410* |
| *03-03-2022 291* | *09-09-2022 709* | *03-03-2023 1200* | *09-09-2023 1509* |
| *04-04-2022 325* | *10-10-2022 828* | *04-04-2023 1077* | *10-10-2023 1669* |
| *05-05-2022 528* | *11-11-2022 955* | *05-05-2023 1260* | *11-11-2023 1672* |
| *06-06-2022 486* | *12-12-2022 941* | *06-06-2023 1163* | *12-12-2023 1696* |

Just for testing the result, we will not set the battery limit to 1500, but 2000. From the result, we can see that the drone could finish all the orders from 01-01-2022 to around 08-08-2022. However, it is true that the order cannot be fully delivered between September and December in 2023. To solve this, we might consider improving the algorithm. The application runtime for each of the results is no more than 5 seconds.

**2.4 Possible Improvement**

1. We can record the shops and deliver-to locations the drone has arrived and save them into a list. Before the iteration of each target, we will check if the next target is in the list. If it is in the list, we will reproduce the path we traveled last time. If not, we put this target into the list.

2. Use the A\* path-finding algorithm to automatically find the best path

**2.5. Conclusion**

This algorithm is relatively efficient to find the path for the current number of orders. However, if the number of orders increases, and the application requires less time of executing the algorithm to find the flightpath for the drone, it would be a challenge to trade off the chose of algorithm to balance the efficiency and effectiveness.

**2.6 Resources**

1. MapBox Java SDK, https://docs.mapbox.com/android/java/guides/

2.Java.awt.geom. Line2D,

https://docs.oracle.com/javase/7/docs/api/java/awt/geom/Line2D.html

**地图

描述已自动生成**

Figure 2-3 drone-31-12-2022.geojson

**图示

描述已自动生成**

Figure 2-4 drone-31-12-2023.geojson