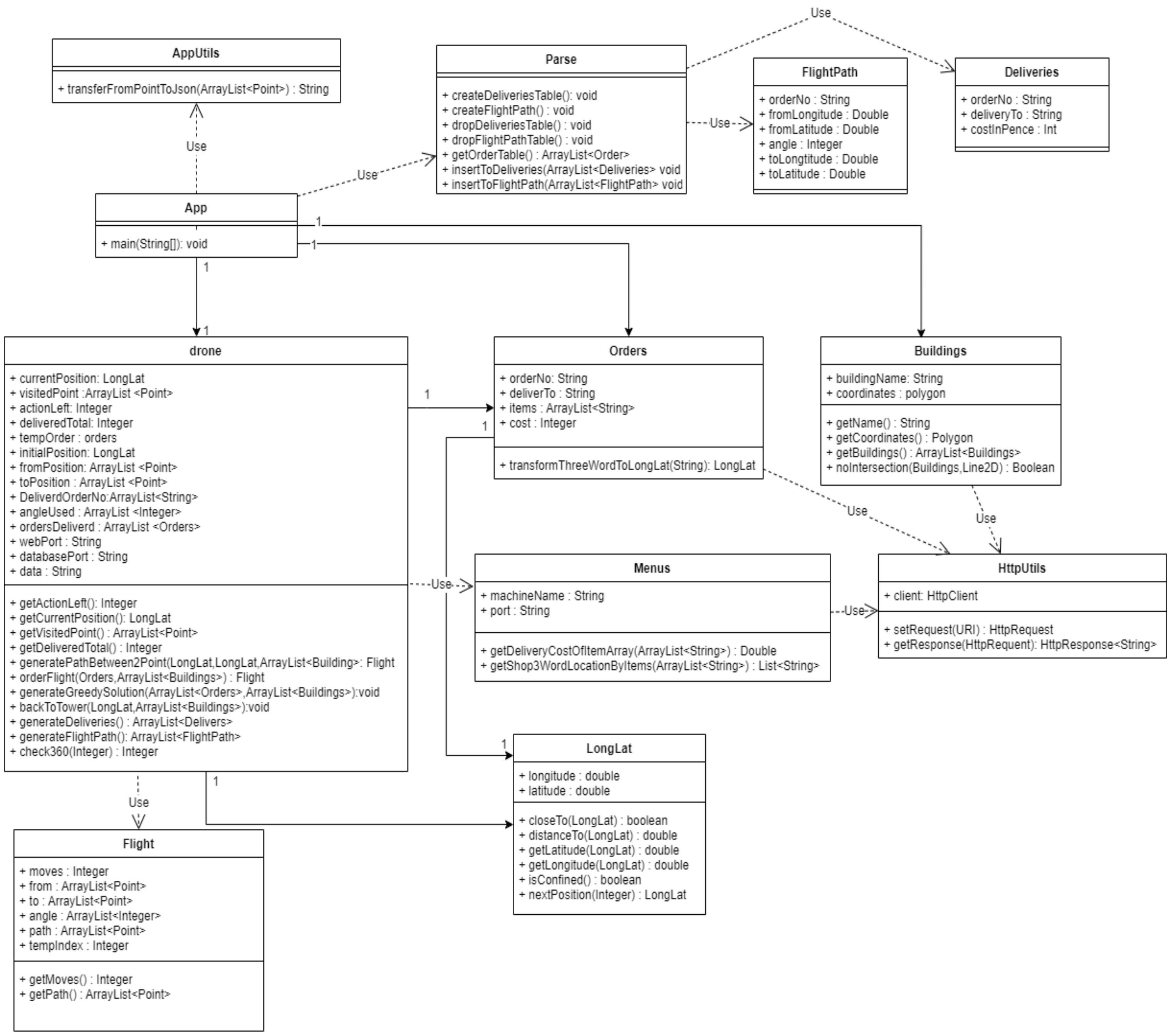
ILP report

S1935167

Qiancheng Fan

1. Software architecture description

1.1 The UML diagram of the program.



AppUtils
+ transferFromPointToJson(ArrayList<Point>) : String
Use
Parse
+ createDeliveriesTable(): void
+ createFlightPath() : void
+ dropDeliveriesTable() : void
+ dropFlightPathTable() : void
+ getOrderTable() : ArrayList<Order>
+ insertToDeliveries(ArrayList<Deliveries> void
+ insertToFlightPath(ArrayList<FlightPath> void
Use
FlightPath
+ orderNo : String
+ fromLongitude : Double
+ fromLatitude : Double
+ angle : Integer
+ toLongtitude : Double
+ toLatitude : Double
Use
Deliveries
+ orderNo : String
+ deliveryTo : String
+ costInPence : Int
Use
App
+ main(String[]): void
1
1
1
1
drone
+ currentPosition: LongLat
+ visitedPoint :ArrayList <Point>
+ actionLeft: Integer
+ deliveredTotal: Integer
+ tempOrder : orders
+ initialPosition: LongLat
+ fromPosition: ArrayList <Point>
+ toPosition : ArrayList <Point>
+ DeliverdOrderNo:ArrayList<String>
+ angleUsed : ArrayList <Integer>
+ ordersDeliverd : ArrayList <Orders>
+ webPort : String
+ databasePort : String
+ data : String
+ getActionLeft(): Integer
+ getCurrentPosition(): LongLat
+ getVisitedPoint() : ArrayList<Point>
+ getDeliveredTotal() : Integer
+ generatePathBetween2Point(LongLat,LongLat,ArrayList<Building>: Flight
+ orderFlight(Orders,ArrayList<Buildings>) : Flight
+ generateGreedySolution(ArrayList<Orders>,ArrayList<Buildings>):void
+ backToTower(LongLat,ArrayList<Buildings>):void
+ generateDeliveries() : ArrayList<Delivers>
+ generateFlightPath(): ArrayList<FlightPath>
+ check360(Integer) : Integer
1
1
Orders
+ orderNo: String
+ deliverTo : String
+ items : ArrayList<String>
+ cost : Integer
+ transformThreeWordToLongLat(String): LongLat
Buildings
+ buildingName: String
+ coordinates : polygon
+ getName() : String
+ getCoordinates() : Polygon
+ getBuildings() : ArrayList<Buildings>
+ noIntersection(Buildings,Line2D) : Boolean
Use
Use
Use
Menus
+ machineName : String
+ port : String
+ getDeliveryCostOfItemArray(ArrayList<String>) : Double
+ getShop3WordLocationByItems(ArrayList<String>) : List<String>
Use
HttpUtils
+ client: HttpClient
+ setRequest(URI) : HttpRequest
+ getResponse(HttpRequent): HttpResponse<String>
1
LongLat
+ longitude : double
+ latitude : double
+ closeTo(LongLat) : boolean
+ distanceTo(LongLat) : double
+ getLatitude(LongLat) : double
+ getLongitude(LongLat) : double
+ isConfined() : boolean
+ nextPosition(Integer) : LongLat
Use
1
Flight
+ moves : Integer
+ from : ArrayList<Point>
+ to : ArrayList<Point>
+ angle : ArrayList<Integer>
+ path : ArrayList<Point>
+ tempIndex : Integer
+ getMoves() : Integer
+ getPath() : ArrayList<Point>

1.2 Briefly description of implementation process.

This program has 5 input arguments, 3 of them represents the date we want to check, 2 of them represent website port and database port. In app class, first, we use these input arguments to initialize a drone. Then we use website port to get information about no-fly-zone buildings, shops and pick up locations from website. And use database port and the date to get information about all the orders at that day in database. After having that, we can call drone's method generateGreedySolution to generate the flight path with highest possible gain of monetary value by greedy algorithm. Then we call appUtils's method to convert the flight path to geojson format and generate a geojson file after that. Then we call methods in drone class to sort all required data for those tables we need to generate. And use methods from Parse class to create required table in database and insert data to it.

1.3 reason for choosing these classes.

1.3.1 App

The app class is the main class that contains the main () method. It will handle inputs arguments and call other class's method to achieve the main purpose of this program, which is generate flight path of the given date by using the given website port and database port, output a geojson file of the flight path and store deliveries table and flight path table into the database.

1.3.2 AppUtils

The AppUtils class is used to provide helper function for App class when it does output. It contains a method that convert a list of point to a geojson string, App class can use this to convert the generated data to the data type required for output.

1.3.3 Buildings

The buildings class is used to represent a no-fly-zone building. It stores the building's name and its polygon. And it contains some method for the no-fly-zone building. Such as get data of no-fly-zone buildings from website, check whether a line intersect with the building. These methods mainly used when generating the flight path.

1. 3.4 Deliveries

This class is used to represent the deliveries table, to store information in database.

1.3.5 Drone

The Drone class represents the drone, it is responsible to store drone's positions, battery remaining, and a list of visited points in the drone's lifecycle. It's the most important class in this project, it contains most of the methods used in generating greedy flight paths. Including methods that can generate flight path of an order by greedy algorithm, find which order has highest rate of monetary value/moves, and generate the flight path of all moves in drone's life cycle. It also contains some methods on sorting out the data needed for database tables.

1.3.6 Flight

The flight class is used to represent a flight path, it contains several lists of points to stores start and end position of each move, the angle of each move, and each point that has visited. It also stores total move number and an index that used in generating final flight path.

1.3.7 FlightPath

The flight path is used to represent the flight path table to store information in database.

1.3.8 HttpUtils

The HttpUtils class is used to provide helper methods for communicating with webserver. Its main responsibility is to send requests to the website and receive the response data.

1.3.9 LongLat

The LongLat class is used to represent a position with its longitude and latitude. It also has some methods relate to positions. Such as whether a position is in confinement area, what is the next position after moving with an input angle from a position, and what's the distance between 2 positions.

1.3.10 Menus

This class represents the menus folder in the website server, its nested class item and shop are used to represent data of each item and shop. Such as item's name and cost, shop's name and address. It also has some methods used to handle these data, such as calculate total delivery cost by a list of items, find related shop's location by a list of items.

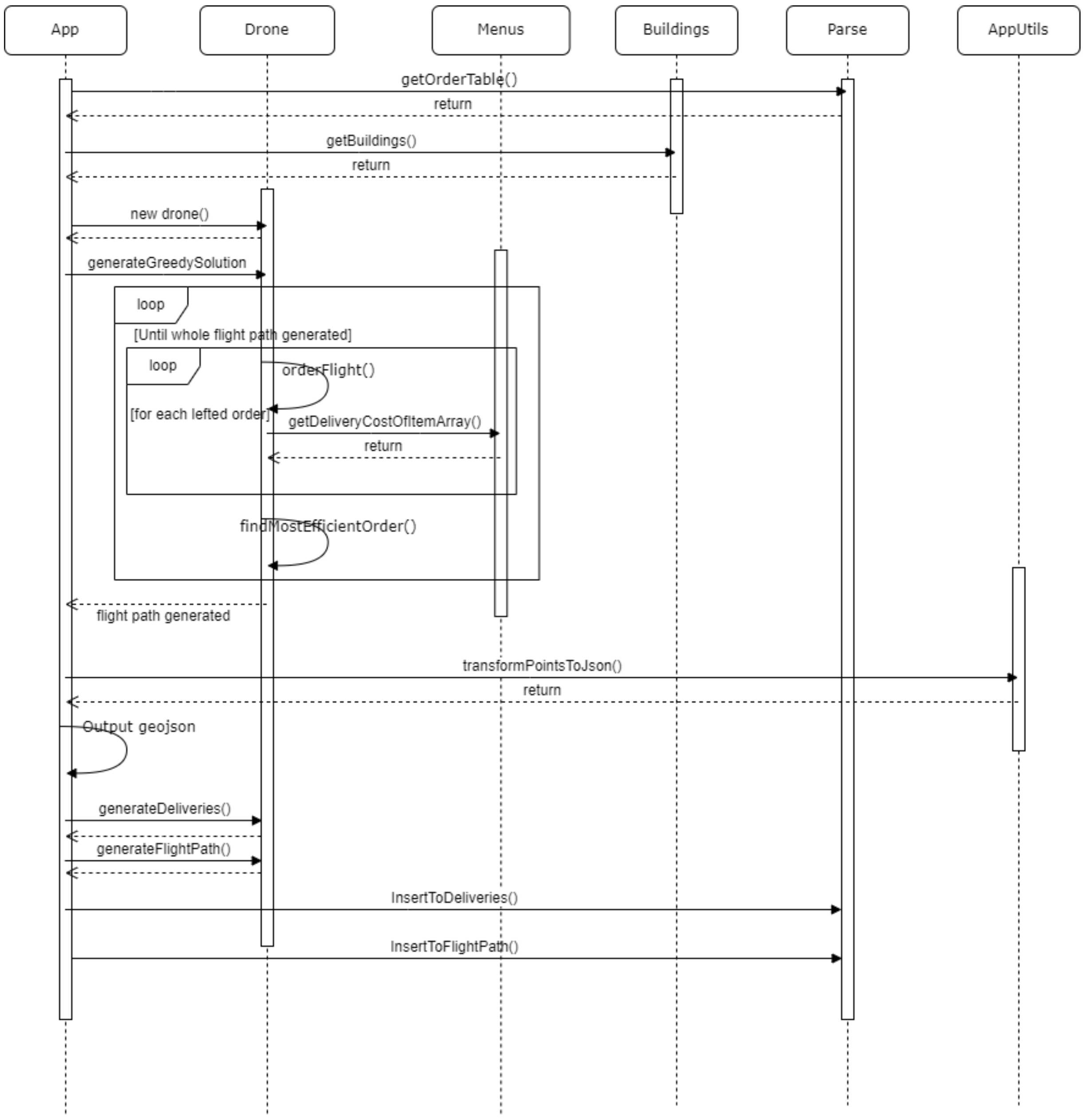
1.3.11 Orders

This class represents an order. It stores data of this order including its name, its deliver address, the list of items it will deliver and the total monetary value of this order. And it contains a method that can convert a 3word address to LongLat to process delivery address and shop address. It also has several nested classes used to represent data when get orders from the database.

1.3.12 Parse

This class is used to communicate with database, it contains helper methods that used to get data from database and write data to the database. App class use its method to write data, drone class and orders class use its method to get data.

1.4 Briefly sequence diagram.



The above sequence diagram briefly explains how the program work and how these classes interact with each other. It only contains some important methods and classes of this program, but you can get the basic idea from it.

2. Drone control algorithm

2.1 Problems that need to solve.

In this project, we need to generate a flight path that can deliver as many orders as we can and return to start position. Drone's battery only allows it to do 1500 moves, and each order need to deliver different items from different shops. The monetary value of each order, and the moves of each order needed is different, so we need to pick order that have highest average percentage monetary value. And the movement of drone also have limitation, the drone can not move across no-fly-zone buildings, it can only move with angle that from 0~350 and it only can be a multiple of 10, it can not move out of confinement area, and each move is a straight line of length 0.00015 degrees.

2.2 Move between 2 points.

2.2.1 Move between 2 points with greedy algorithm at most common case.

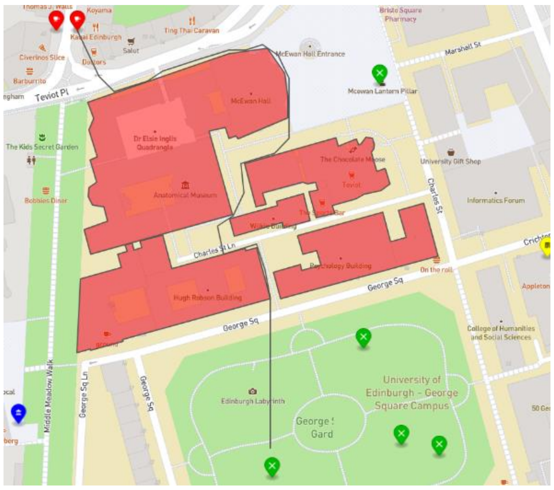


Figure 1

We use figure 1 as example to explain my algorithm. When we simulate each move of drone, it will loop through all 36 possible move angles, if the angle will cause intersection with no-fly-zone buildings or get out of confinement area, we ignore it. In the rest angles, we pick the angle that can make next position closest to destination. We keep do this for each move, until the distance between our current position and destination smaller than 0.00015 degrees. By doing this, we can generate a relatively efficient flight path, you can see the path at figure1. I admit it's not the most efficient path at some case. We can see the path at figure 3 is better than my path at figure 2, but I use it because my path still better than the solution that go to the landmark first and then go to destination. But in some case, this method will cause local optimal problem and the drone will stuck there move on and back forever, in this case, I will use next algorithm to handle it.

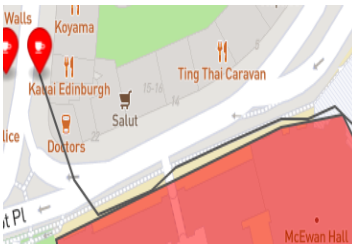


Figure 2

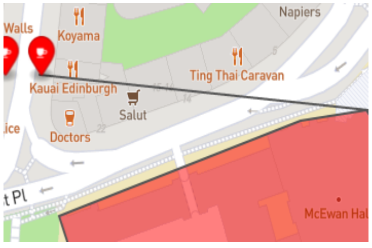


Figure 3

2.2.2 The algorithm used to handle stuck.

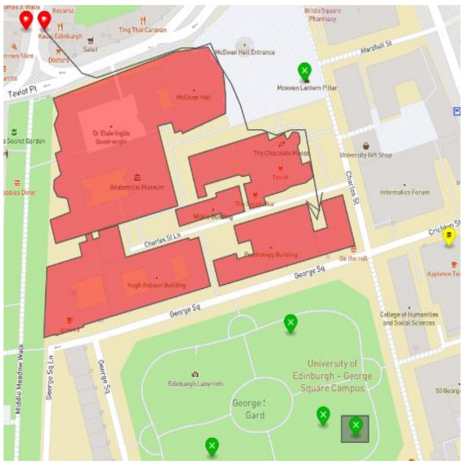


Figure 4

From figure 4 we can see if we use the above method to move from this shop to the green position covered with a rectangle, the drone will stuck there. To deal with it, I set an upper limit of total moves, if the above algorithm's total move exceeds 100, I will stop using this method and store a signal 999. It will tell my program to use the method called handleStuckFlight. This method first finds the angle that make the point as close as possible to destination. And then calculate the position that drone take 15 moves along this angle, check whether the line between current position and this position intersect with any no-fly-zone buildings, if there is intersection, I rotate this angle ten degrees clockwise and do the check again, I keep doing this until I find a valid angle. Then I let the drone do a move along this angle. And do the above thing again for the next move. And for each move, check whether the current position can directly move to destination with no intersection. If the check pass, I will use the above method **2.2.1** for rest flight path to minimize the moves needed. Otherwise, I use this method to move until the distance between our current position and destination smaller than 0.00015 degrees. The flight path generate by this method is show in Figure 5. From the figure we can see by doing this, we avoid the stuck problem. But this method has a potential problem that in many cases, the flight path generate by this method will have more moves. It's why I only call this method when method **2.2.1** has local optimal problem.

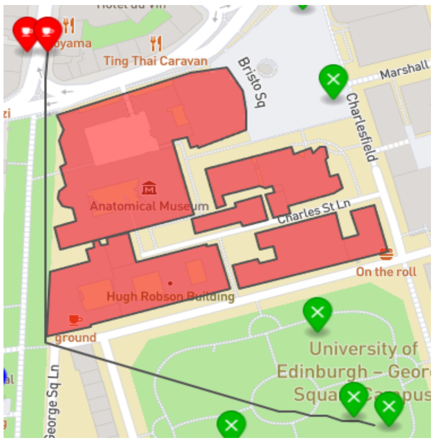


Figure 5

2.3 Generate the flight path of an order

The main part of this is done by the method orderFlight in drone class. In the beginning, I initialize an instance of flight class to store the flight path. An order delivery composed by 2 parts, move to shop to take items, and then sent items to deliver address. An order may have items from several shops, so first I use a for loop go through all related shops, use above moving methods **2.2.1** and **2.2.2** to move to these shops one by one, take a hover when I arrive each of them and add these paths to the flight path I create in the beginning. Then I move to deliver address and do a hover there and add the flight path. The final flight path will be the flight path of delivering this order.

2.4 Find the order that can maximize the average percentage monetary value by greedy algorithm.

The main part of this is done by method findMostEfficientOrder in drone class. It will loop through the order list, use method orderFlight that described in **2.3** to generate flight path for each order, generate total cost in pence of this order by method getDeliveryCostOfItemArray in menus class. And use these 2 data to calculate the percentage monetary value by monetary value/moves. After going through all the orders, it will return the order that has highest percentage monetary value. Since I use greedy algorithm to choose order, it make cause potential issue that some order itself may not have very high percentage monetary value, but its ending position will make follow up orders take less moves, my algorithm cannot solve it, so it still has space for improvement.

2.5 How to back to Appleton tower

It basically just uses 2.2's methods move from current position to Appleton tower.

2.6 Generate the final solution

The main part of this is done by the method generateGreedysolution in drone class. This method has 2 inputs, the list of today's orders and the list of no-fly-zone buildings. It uses while loop go through the order list until the list is empty. With in each loop, it uses findMostEfficientOrder to find the best order, and check drone's battery is enough for delivering this order and return to Appleton tower. If the check pass, I will store the information of this order and its flight path. And remove it from the order list. If the check does not pass, then just remove this order, and do nothing else. By doing it, when the while loop finish, either all orders has delivered, or the remaining battery is not enough for any other delivery.

2.7 Conclusion

In conclusion, since I use greedy algorithm to generate the flight path and chose orders' deliver sequence, my solution is relatively efficient but still have some space for improvement. 2 examples of the generated geojson file for flight path at different date and a table of detailed results of more dates is show below.

3. Results of the algorithm.

Date	Orders delivered	Moves take	Execution time
1/1/2022	4/4	283	0.750
2/2/2022	5/5	303	0.747
3/3/2022	6/6	281	0.792

4/4/2022	7/7	304	0.954
5/5/2022	8/8	510	1.141
6/6/2022	9/9	460	1.225
7/7/2022	10/10	701	1.487
8/8/2022	11/11	805	1.509
9/9/2022	12/12	683	1.721
10/10/2022	13/13	773	1.760
30/11/2023	26/26	1399	4.568
31/12/2023	26/27	1475	6.377

We can see the result of my algorithm is reasonable and the execution time is acceptable. Few seconds execution time is way less than the requirement 60 s. And the ability to go back when the battery is not enough for other orders also works fine, we can see at 31/12/2023, when battery charge not enough for all orders, drone successfully come back to Appleton tower within 1500 moves.

4. Flight path examples.

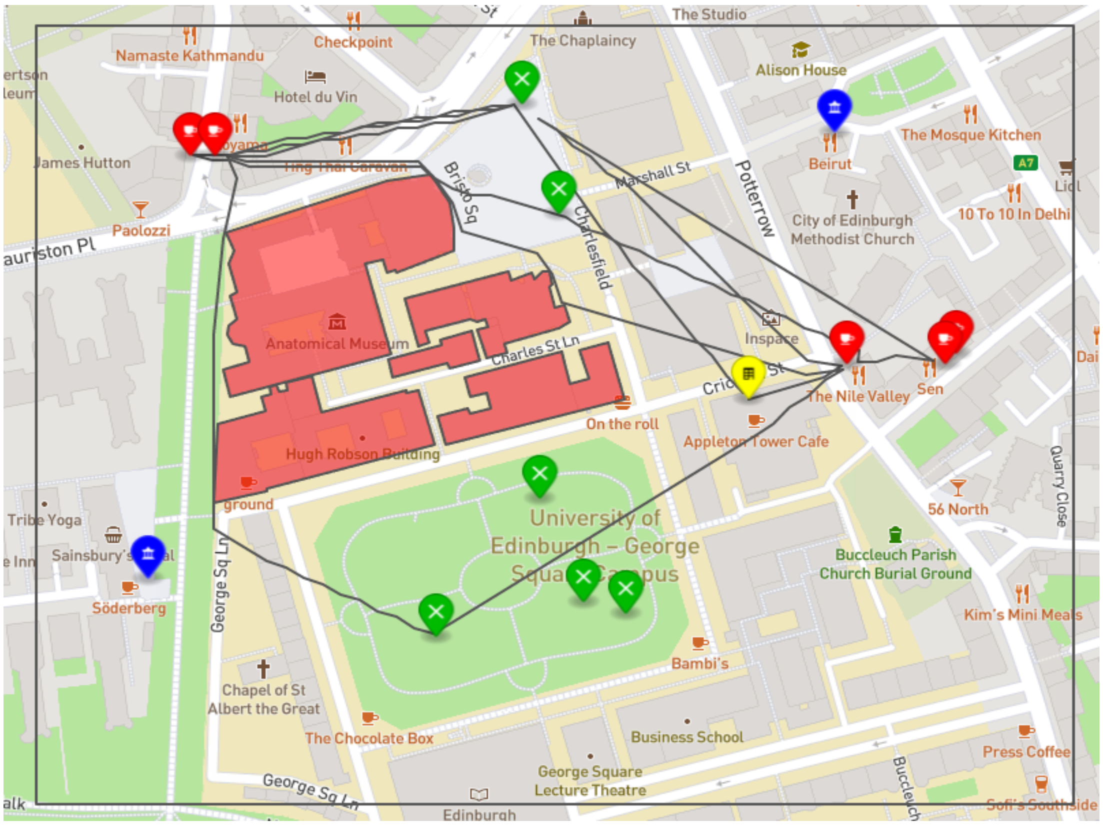


Figure 6 drone-03-03-2022.geojson

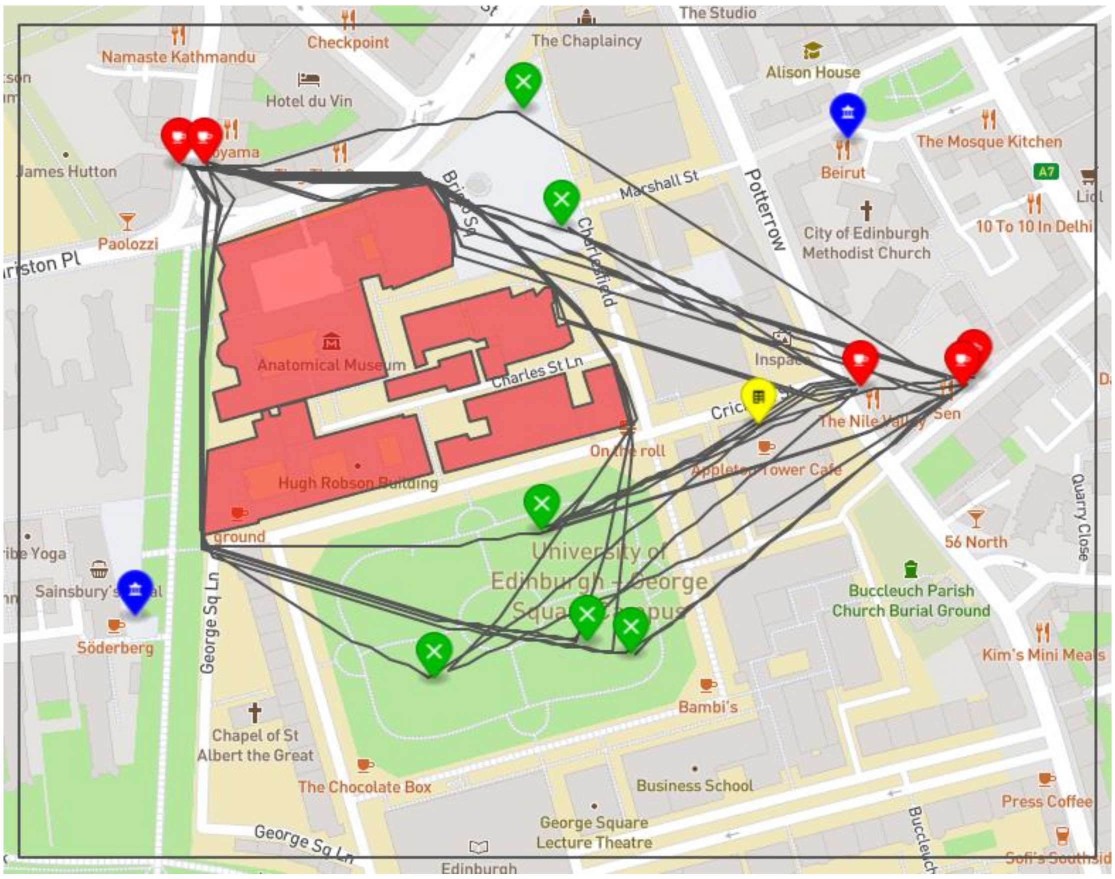


Figure 7 drone-12-12-2022.geojson