Arnav Sinha - s1951693

Informatics Large Practical

Report - Coursework 2

Software Architecture Description

Structure Overview

The application is formed through a collection of 14 vital classes: *App, Client, Database, DronePath, GeoJson, Landmark, LongLat, MenuDetails, Menus, NoFlyZone, Orders, PathFinding, LngLatCoord and What3Words*. These classes together form an application that takes in a date, web server port and database server port and outputs the result of an autonomous drone's flight around a defined area, collecting and delivering orders and avoiding the no fly zones.

App

This is the main class containing the simulation of the drone flight over the geographical area of the confinement zone. The user input is read into the program through the *App* class (day, month, year, web server port and database server port). There are two major functions this class performs-

1. Populating the Database - Two tables are created in the database namely *'deliveries'* and *'flightPath'* by calling the *createTable function* in the *Database* class. Then all the orders are initialized with the required restaurant locations and the delivery cost by calling the *deliveryDetails* function from the database class. Finally, both the tables are populated by the orders placed on that day and the *sampled average percentage monetary value* is calculated to check the performance of the drone.
2. Setting up the map - The *GeoJSON* class helps in setting up the drone path on the map by first converting the coordinates into points by calling the *addLocation* function and then saving the data on a GeoJSON file for output purposes.

Apart from these, the *App class* also prints useful information about the drone application like the number of moves left and the *sampled average percentage monetary value* for the developers who would further work on the system to analyse.

Arnav Sinha - s1951693

Client

We have contents on the web and the database server that we need access to from our java code. This *Java HTTP Client* helps in accessing web and database server content. This class essentially helps in parsing the URL(*menus, what3words, landmarks, NoFlyZone and Database*) and that we pass through the *GET request* and save the response in a string object for further use.

Database

This class is mainly used for accessing the database, creating the two *'deliveries'* and *'flightPath'* tables[1] and finally populating the table to get the relevant data to carry out the operations. The *Database* function is used to access the database derbyDB[1] in which we create the tables to store information. But before this, the *tableManipulation* function is used to drop the tables with names similar to *'deliveries'* and *'flightPath'* so the create table command does not fail with an exception. Now, the two empty tables are created using the *createTable* class so that they can be populated accordingly. Thereafter, the *ordersPerDay* function is used to get the list of orders for the date user wants and the *deliveryDetails* function feeds in the details to the relevant tables.

DronePath

This class does not have any functions as such and is only used to store every drone movement detail. It is used in the *PathFinding* class to make a list of all the initial and final coordinates of a single drone movement and the angle at which the drone moves for storing these details in the *'flightPath'* table of the database. This helps in plotting the GeoJSON map.

GeoJson

This class is used to write the GeoJSON file drone-DD-MM-YYYY which is used for the path plotting. The *addLocation* class is used to first convert a given location in longitude and latitude format to a point format for the geojson output file.

[1] As per the Coursework specification

Arnav Sinha - s1951693

Landmark

There are several landmarks provided on the map(in the form of point locations) so the drone can refrain from passing through the no fly zone. This class is used to parse the provided landmarks and store them in a longitude and latitude format for use while checking and diverting the drone if a no fly zone lies between the drone's path from one location to the other.

LongLat

This class is used to represent a position on the given map in the form of longitude and latitudes and performs specific functions on and with the point which are helpful in creating a robust algorithm. The *isConfined* function checks if the point is in the confinement zone provided in the specification (between Forrest Hill, KFC, Buccleuch St. Bus Stop and Top of the Meadows in this case). Once the next position of the drone is checked and is confirmed that lies inside the confinement zone, the *distanceTo* and *droneAngle* functions are used to calculate the distance and the angle to the next position the drone has to traverse to respectively. Drone moves in a straight line of length 0.00015 degrees[2] and is considered to reach the next location if it is in the proximity of the distance tolerance of 0.00015^2 where it hovers if it is picking up an order from a restaurant or is delivering an order at the drop off location. The movement of the drone to the immediate next position and the hovering is taken care of by the *nextPosition* function and to check if the drone is near the next location we have the *closeTo* function.

MenuDetails

This class gets the restaurant details(like the name and location) and the details of menu items of that particular restaurant(like the menu item and cost of that particular item(in pence)[2]). Then the menu items and their respective price is taken using *eachItemCost* and *eachRestaurantLocation* for performing specific functions in the Menus class in order to populate the 'deliveries' table of derbyDB database.

Menus

This class is used to get the menu details of a particular order(like the restaurant location and the price including the delivery charge) in order to initialize the *'orders'* table with all the delivery details. The menu details of the order are taken from the *MenuDetails* class and

[2] According to the Coursework specification

then the *getDeliveryCost* function is used to calculate the total order cost including the delivery cost and the *getRestaurantLocation* function is implemented to return all the restaurant locations the drone has traversed through. This information is needed in the *App* class where we calculate the *sampled average percentage monetary value*(the total delivery cost) and feed in the drone the restaurant locations to pick the order from(*getRestaurantLocation*).

NoFlyZone

There are some parts of the map that the drone cannot travel through which are labelled as the no fly zones. These are shown as different sized polygons[3]. This class is used to parse all the provided no fly zones in the specification through the *getNoFlyZone* method so the drone can be redirected to the required landmarks if its path passes through this area. To facilitate this functionality, first, all the *LongLat* coordinates of the polygons are stored in a list and then each point in this list is checked to see if it does not intersect with the drone path in the *checkPointInNoFlyZone*[4] method using the *doIntersect* function.

Order

This is the class that is used to store all the important information about the order like the order number, delivery cost, delivery location, restaurant location and landmark position. These are the relevant metrics that help to implement the main drone control algorithm.

PathFinding

This is the class that consists of the major part of the drone control algorithm which is discussed in detail in the next part of the report. This class performs multiple functions starting with storing all the locations the drone has to traverse through to complete a single order. This is done in the *dronePathByOrder* method. Then the drone moves with a path length of 0.00015 degrees per move[5] and if it intersects the no fly zone, it is directed towards the closest landmark using the *addLandmark* function. After the drone is done with all the orders for the day, it has to return to Appleton Tower which is performed by the *finalMoveToAppleton* method. The drone cannot suffice to move for more than 1500 moves

[3] According to the Coursework specification
[4] The no fly zone intersection code block for checking if a LongLat point lies inside the no fl;yzone or not was taken from the GeeksforGeeks site.
Link:https://www.geeksforgeeks.org/how-to-check-if-a-given-point-lies-inside-a-polygon/
[5] As per the Coursework specification

in a day otherwise its battery would die out so the total moves made by the drone is counted and advanced functionality is implemented so that the drone does not discharge midway through delivery. Further details regarding this are discussed in the next section of the report.

What3Words

Each and every location described on a map is in the form of a 3-word format separated by dots to make it easier for the customers. This class is used to get the required location in the 3-word format and convert it into a longitude-latitude coordinate location for the drone algorithm to work on. This other helper class **LngLatCoord** is used to store all the coordinates received.

Collectively, all these classes are important in defining the structure of the program. The entire code is made modular to increase its readability and maintenance. All the relevant classes, methods and variables are properly encapsulated as and where needed by using the access modifiers public and private and the keywords static and final. All the methods and variables are highly relevant to the classes they are in and all the elements are closely related making the code highly cohesive. The classes used have very relevant functionalities in order to implement the task at hand.

Drone Control Algorithm

Aim:

The drone starts from Appleton Tower every day, collects orders for each delivery from the restaurant locations which have that particular item[6] and delivers it to the drop-off location. Further, it repeats this process either until each delivery for the day is done or returns to Appleton Tower before running out of battery if the deliveries for the day extend 1500 drone moves(A drone can only travel 1500 moves in a day before running out of battery). In addition, the drone must follow the life cycle where each unit of movement is exactly 0.00015 degrees.

[6] Assumption that there are no two restaurants with the same item on the menu(according to the coursework specification

Arnav Sinha - s1951693

Procedure:

Traversing all the restaurants via landmarks when needed in an efficient way is an example of the metric Travelling Salesman Problem(since the distance between two locations is always positive). The drone prioritises the orders with a higher total cost(cost of items+delivery charge) and makes those deliveries first. To facilitate this the orders in a day have already been sorted in descending order according to the total cost in the *App* class.

The basic algorithm for the most efficient delivery while adhering to all the specifications provided is described below:-

1. Starting from Appleton Tower, all the restaurant locations and the delivery address for the first order of the day is stored in a list for drone movement[7].
2. If the next location after Appleton Tower is in the confinement zone and does not lie in the no fly zone then the drone moves from Appleton Tower to that location(usually a restaurant)[8].
3. The drone can move only at a threshold distance of 0.00015 degrees[4] in the direction of the next location and the drone is considered to reach the location if it is within the radius of 0.00015 degrees[9] of the next location.
4. Once the drone has reached the restaurant to collect the order or the delivery drop-off location, it hovers there and that is also considered a move.
5. If the next location the drone has to traverse to lies in the no flyzone then the best available landmark is chosen so that the drone can avoid the no flyzone. The best landmark is the landmark that has the least cumulative distance from the initial location to the landmark and the landmark to the next location[10].
6. Once (the drone has done 1400 moves[11]) all the deliveries are done or the drone is about to get discharged, the drone is made to head back to the starting position,i.e., Appleton Tower.

---

[7] Using the *dronePathByOrder* method in the *PathFinder* class.
[8] Using the *droneMovement* method in the *PathFinder* class.
[9] As per the Coursework specification
[10] Using the *addLandmark* method in the *PathFinder* class.
[11] An assumption is made that the drone will deliver the order till it reaches 1400 moves after which it will return to Appleton Tower. This was done to ensure that the drone does not discharge midway a delivery. This will also affect the total sampled percentage to calculate drone efficiency but doing 90% of the deliveries instead of 95% is far better than losing a drone and the food items that it was delivering. This is because of two reasons:1)hign cost of building a drone and cost and wastage of food, 2)Disturbance to the general public.

7. Since the drone cannot make more than 1500 moves in a day, the drone efficiency is calculated by looking at the sampled average percentage monetary value[4].

The steps of the algorithm above are discussed in detail below:-

Getting the Locations

The drone needs to start from Appleton Tower and traverse to the different restaurants to collect the orders and finally deliver it for each order. Once all the deliveries for the day are complete[5], the drone returns to Appleton Tower. Since TSP cannot be optimally solved, we need to provide the best path the drone should take. This is done by delivering the orders with the highest cost price first. The locations are passed to the drone for every order as a *LongLat* list. The first location for every order is the last delivery location(after the first order) until all the orders are done and the drone finally ends up at Appleton Tower.

Moving the Drone

The ordered list of locations for every order is then passed to the drone. The drone can make a move of 0.00015 degrees in a straight line, at any angle between 0 and 350 where the angle is a multiple of 10(we follow the convention that 0 means go East, 90 means go North, 180 means go West, and 270 means go South).

The drone moves from one location to the next location at the angle between them according to the list provided and the list is updated constantly with the required locations for every order in the day. The drone does not stop moving until it is *closeTo* the next location. Once it has reached the next location, it hovers there if the location is a restaurant(to collect the order) or a delivery drop-off location(to drop the order).

The moves of the drone is stored as a threshold so that the drone does not surpass the threshold moves of 1500 including going back to Appleton Tower. The hovering is also counted as move for the drone.

Avoiding the No Fly Zones

There are four no fly zones[12], each a polygon in shape. When the algorithm calculates the next path for the drone, then it is checked if the drone path passes through the no fly zone or not. If the drone path is outside the no fly zone area, it directly traverses to the next

[12] As per the Coursework specification.

location but if the drone path does pass through the no fly zone, then the drone uses the landmarks present on map to avoid the no fly zones.

There is one other situation where the drone might pass through the no fly zone. The drone moves in straight lines of 0.00015 degrees length towards the next location. While making these movements towards the specified location the angle between the drone and the location it is moving towards keeps changing(slightly). This can give rise to a situation where the drone might just graze through or just pass through the no fly zone which needs to be avoided(as shown in Figure 1(left)). For this, we have a check block in the algorithm(*droneMovement* method) where if the drone passes through the no fly zone it is shifted by 10 degrees until it moves out of the no fly zone(as in Figure 1(right)).

This is not the most efficient way to approach this problem but respecting the Coursework specification and the timeframe, I tried my best to focus on ensuring that the entire specification was met as efficiently as possible.

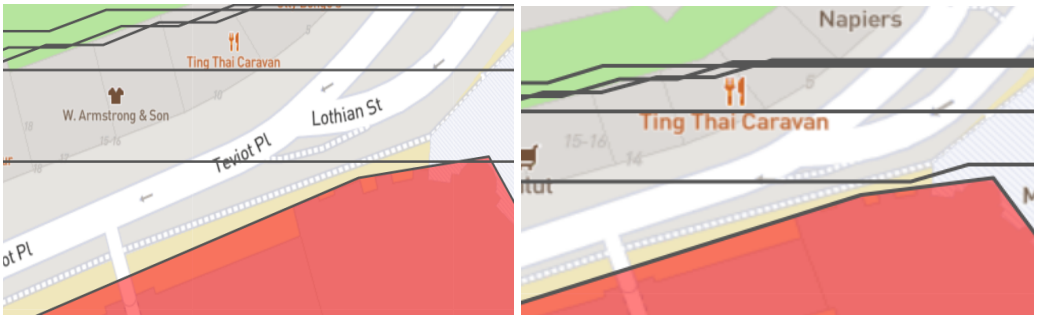

Figure 1 : An image of drone path from 09-09-2022 without the shift angle(left) and with the shift angle(right)

This algorithm can be developed further to make it more efficient. The shift angle implementation in the program is done by brute force by just shifting the path by 10 degree angles. This can be made better by checking for the anticlockwise and clockwise directions so the drone has to shift the least from the original path to avoid the no fly zone. Also the approximation of 1400 for the drone should be made more robust so that the drone can deliver more orders before going back to Appleton Tower.

Travelling via Landmark

The two landmarks[13] are point locations on the map to help avoid the no fly zone. The program is generalised for any number of landmarks. The function of Landmarks has already been defined in the basic structure of algorithm.

Reflection

The project has been eventfully challenging and it was great learning curve. My algorithm is relatively efficient, but could further be enhanced to counter the few discrepancies as discussed above. Different factors like limited time and making the code modular has made implementing this project a fulfilling challenge.

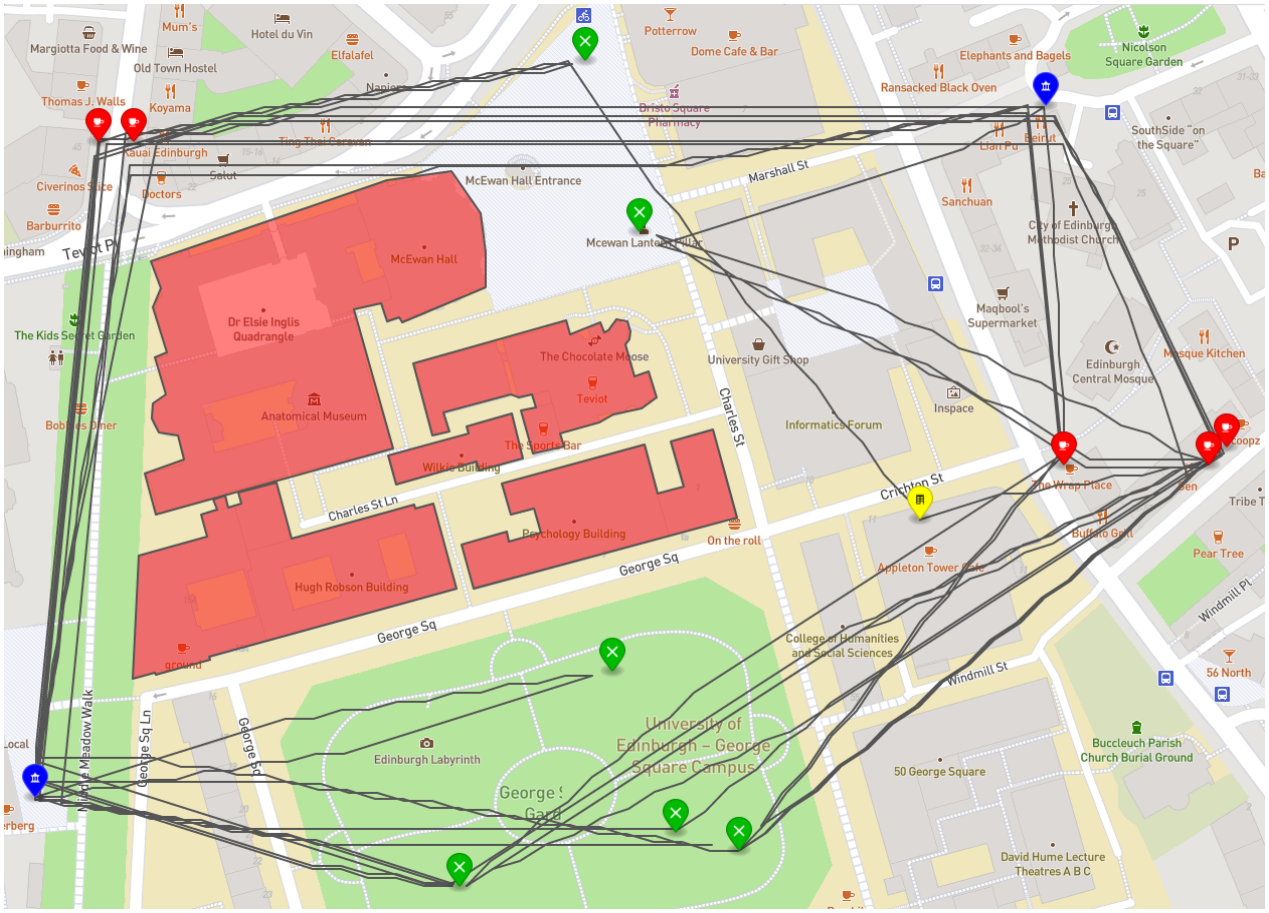

Figure 2 : A sample output map for the date 07-12-2022 with the starting position of Appleton Tower.

[13] As per the Coursework specification

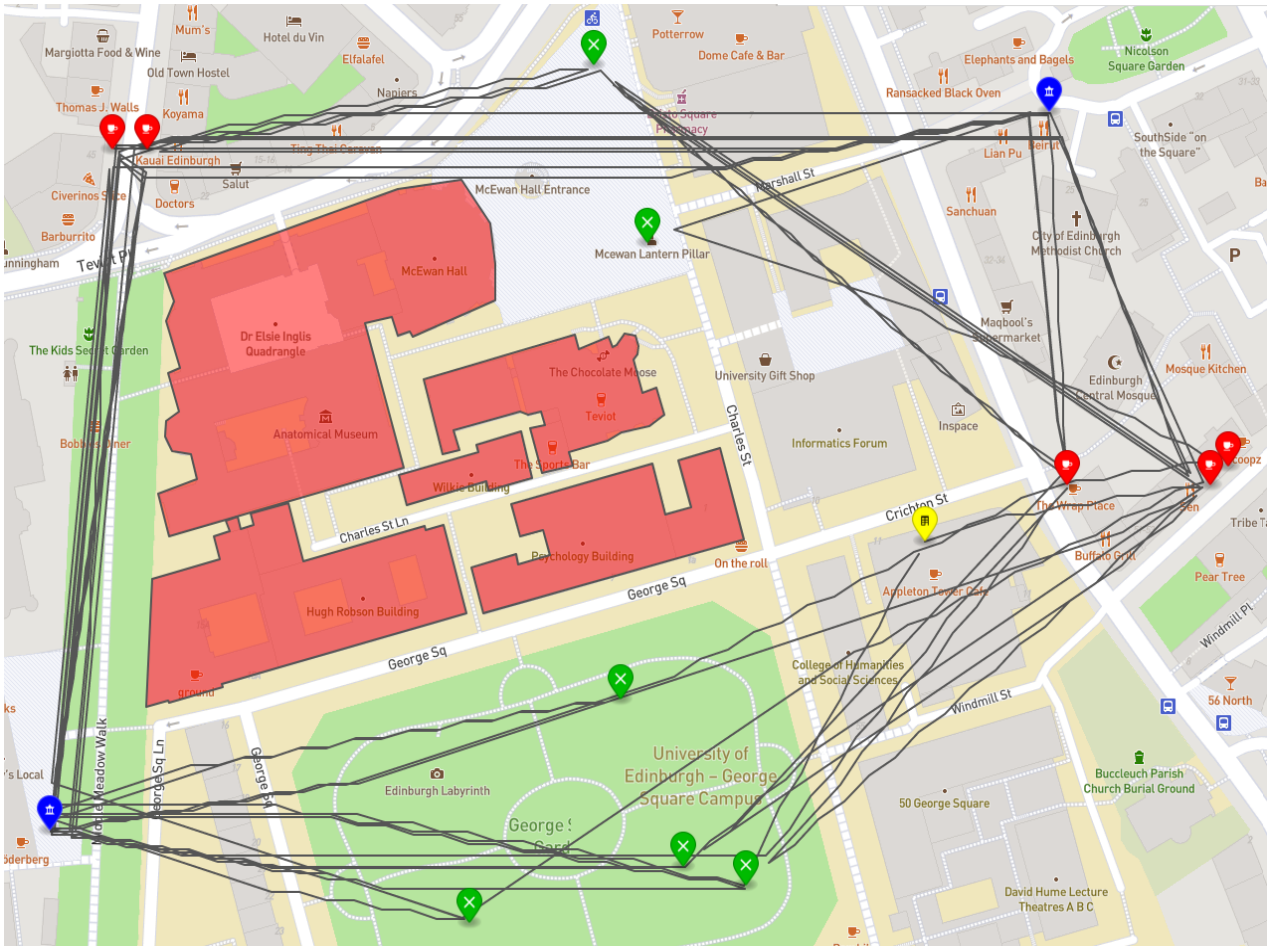

Figure 3 : A sample output map for the date 15-02-2023 with the starting position of Appleton Tower.

References:

1. Informatics Large Practical *Stephen Gilmore*, 2021
2. Travelling Salesman Problem
   https://en.wikipedia.org/wiki/Travelling_salesman_problem
3. GeeksforGeeks for checking if a point lied inside no fly zone algorithm
   https://www.geeksforgeeks.org/how-to-check-if-a-given-point-lies-inside-a-polygon/