# Informatics Large Practical Report

# Table of Contents

# 1. Software Architecture Description

The classes present in this package are:

- App
    - This class works as a controller for the framework. It parses the arguments, connects to the network to get the GeoJson, starts the drone's search and logs the resulting GeoJson and txt files. It does not handle any data except for data required for logging. App is the entry to the program and handles running the general framework. It was included so there is a level of abstraction from the search algorithms.
- Logging
    - To encapsulate logging, this singleton class contains static methods and a static variable which are required to write the geojson and txt files.
    - Logging writes to the txt file as the program runs because this is slightly more efficient than storing the string and writing it at the end. It also becomes more efficient if the number of moves per game grows.
- Map
    - Map is a singleton class as the game should only be played over one map. It contains a static list of stations from the online map for the specified date. It makes sense to store the stations in the map separately to the other classes so an (essentially) global list of the charging stations can be used and updated.
- Direction
    - This is an Enum class containing all the possible directions in which the drone can move. It is used to calculate the drones next position and move it throughout the map.
- ChargingStation
    - This class was included so that the features from the geojson could be used easily, with the power and coins at each station updated here. It also controls the charging of the drone instead of having this functionality in the drone class because the charging station charges the drone when it is within range of the station.
- Position
    - This class tracks the location of the drone on the map. It was included as opposed to using the Point class from the mapbox package since it contains more information. It keeps track of a hScore (used for the stateful search) and could be extended so each Position is considered a node. It also tracks the Direction taken to reach this Position. The last two are only available within the package since they are only used for the search strategy.
- Drone
    - Drone is an abstract super class. It contains all the variables that a drone tracks (power, coins, moves etc.) and contains methods for moving and starting of the drone's search. It has an abstract method searchStrategy() so that each subclass of Drone can implement its own search. For example, StatelessDrone implements a search with limited lookahead and no memory, but only different to the StatefulDrone in this method. Drone is abstract because it cannot function without a search strategy.

- StatelessDrone
    - This is the subclass of Drone, and implements its search strategy with limited lookahead and no memory. Even though it stores the path it takes (for logging purposes), it does not remember visiting a ChargingStation.
- StatefulDrone
    - This is another subclass of Drone, which implements a smarter search strategy with no restrictions on memory or lookahead. It implements a greedy best first search (see more in Section 3).

# 2. Class Documentation

NB: Getters and setters are not mentioned in the method summaries.

## 2.1 App

App acts as a controller for the package. It deals with the input, runs the drone's search and calls the logging for the geojson file. It also controls the networking for the drone, getting a list of features from the server and passing it to the Map class.
App prints the power and coins the drone has by the end of the game and the total power and coins available from positive stations in the map for the specified day.

### 2.1.1 Method Summary

| |
|---|
| static main<br>- Runs the main outline of the drone running<br>- **Parameters:**<br>  - String day, month, year - the date for the map requested<br>  - double latitude - the latitude for the initial drone position<br>  - double longitude - the longitude for the initial drone position<br>  - int seed - the seed for the random generator<br>  - String state - the drone state required |
| static getFeatures<br>- Returns the list of features for the map for the specified date<br>- **Parameters:**<br>  - String day, month, year - the date for the map requested<br>- **Returns:**<br>  - List<Feature> - a list of features for the specified date |
| private static formUrlString<br>- Returns a formatted string with the URL for the map for the server<br>- **Parameters:**<br>  - String day, month, year - the date for the map requested<br>- **Returns:**<br>  - String - a String in the correct format to act as a URL for the server |

| |
|---|
| private static getNetwork<br>- Connects to the server. Returns a String of GeoJson from the map for the day specified in the url<br>- **Parameters:**<br>  - String url - the URL for the server to connect to<br>- **Returns:**<br>  - String - the GeoJson in a string form<br>- **Throws:**<br>  - IOException if it cannot connect to the network |
| private static stringToFeatures<br>- Returns a list of features from the input GeoJson string<br>- **Parameters:**<br>  - String gson - GeoJson in a string<br>- **Returns:**<br>  - List<Feature> - a list of features |

## 2.2 ChargingStation

ChargingStation is used to track the state of each charging station in the map.
Note that the ChargingStation does not have a setter for position, as it will not move once it has been instantiated.

### 2.2.1 Field Summary

| |
|---|
| Position position - used to keep track of the ChargingStation's location |
| float coins - used to keep track of the ChargingStation's coins |
| float power - used to keep track of the ChargingStation's power |
| boolean visited - used to check whether the ChargingStation has been charged from |
| String id - used to identify unique charging stations |
| public static final double chargeRange - the radius of the range for the charging station |

### 2.2.2 Method Summary

| |
|---|
| charge<br>- Charges the drone for being within range of the station. It updates the drone's power and coins. If the drone reaches 0 power/coins before the station can fully charge the drone, the station keeps the remaining power/coins.<br>- **Parameters:**<br>  - Drone drone - the drone that is being charged for being in the range<br>- **Returns:**<br>  - void |

## 2.3 Direction

Direction is an enumerated class containing all the directions in which the drone can move. It can only move in these 16 directions. Enum values:

| | | | |
|---|---|---|---|
| N - North | NNE - North North East | NE - North East | ENE - East North East |
| E - East | ESE - East South East | SE - South East | SSE - South South East |
| S - South | SSW - South South West | SW - South West | WSW - West South West |
| W - West | WNW - West North West | NW - North West | NNW - North North West |

## 2.4 Drone

The drone is an abstract super class. It implements many of the methods that would be used by any subclass but requires the subclasses to implement a specific search strategy for the drone.

### 2.4.1 Field Summary

| |
|---|
| Position position - used to keep track of the Drone's location |
| float coins - used to keep track of the Drone's coins |
| float power - used to keep track of the Drone's power |
| int moves - used to keep track of the number of moves the Drone has used |
| Random rnd - used as a random generator for random moves |
| final double moveRange - the drone's change in distance for each move |
| final int maxMoves - the maximum moves a drone can make before finishing the game |
| final double powerPerMove - the power each move takes |

### 2.4.2 Method Summary

| |
|---|
| endConditions<br>- Checks whether the game end conditions have been reached (maxMoves or not enough power for another move reached).<br>- **Returns:**<br>  - boolean - if game end conditions have been reached or not |
| move<br>- Checks whether the drone can move. If it can: charges the drone for one move, increments the number of moves, sets its new Position and adds this to the history of positions. |

- **Parameters:**
  - Direction dir - the direction in which to move
- **Returns:**
  - boolean - whether a move was successful (true) or not

inRangeOfStation
- Checks whether the drone is in range of a charging station, and if so charges from the closest station.
- **Returns:**
  - boolean - whether a charging station was in range or not

initSearchStrategy
- Initialises the Drone's search and the Logger's BufferedWriter, so the drone can write the logging txt file. Once the search is done, it closes the BufferedWriter.
- **Parameters:**
  - String day, month, year - the date for the map requested
  - String state - either stateless or stateful for the type of drone
- **Returns:**
  - void

abstract searchStrategy
- The actual search strategy for the drone

## 2.5 Logging

Logging contains static classes for creating txt and geojson files to view the output of the drone. It stores a BufferedWriter to write to the txt file through the program.

### 2.5.1 Field Summary

protected static BufferedWriter bw - used to write to the txt file

### 2.5.2 Method Summary

static logToGJson
- Creates the GeoJson file by adding a LineString to the original map and writing this to a new file with the format state-day-month-year.geojson
- **Parameters:**
  - List<Feature> featureList - the original map's list of features
  - List<Position> dronePathTrace - the drone's list of positions
  - String day, month, year - the date for the map requested
  - String state - the state for the map
- **Returns:**
  - void
- **Throws:**
  - IOException if it cannot write to the file

static setWriter
- Sets the BufferedWriter for the class to the correct file location and to the correct file name (formatted state-day-month-year.txt)
- **Parameters:**
  - String day, month, year - the date for the map requested
  - String state - the state for the map
- **Returns:**
  - void
- **Throws:**
  - IOException if the BufferedWriter cannot be set

static logToTxt
- Writes a comma separated line to the txt file using the BufferedWriter in the format initialLatitude,initalLongitude,directionMoved,newLatitude,newLongitude,coins,power
- **Parameters:**
  - Position curPos - the position of the drone before it moved
  - Position nextPos - the position of the drone after it moved
  - Direction dir - the direction the drone chooses to move
  - double coins - the coins the drone has after the move
  - double power - the power the drone has after the move
- **Returns:**
  - Void
- **Throws**
  - IOException if the file cannot be written to.

## 2.6 Map

Map stores the list of ChargingStation objects in the map for a specified day. It also contains some static functions for retrieving information about charging stations in the map and other information.

### 2.6.1 Field Summary

List<ChargingStation> stations - a list of the charging stations in a specified map.

### 2.6.2 Method Summary

static setStations
- Sets the stations from the list of features from a map.

- **Parameters:**
  - List<Feature> featureList - the map's list of features
- **Returns:**
  - Void

static nearestFeature
- Finds the closest ChargingStation to the given position and returns it

| |
|---|
| - **Parameters:**<br>  - List<ChargingStation> stations - the list of stations to search through<br>  - Position pos - the position the station must be closest to<br>- **Returns:**<br>  - ChargingStation - the station closest to the given position |
| static inRange<br>- Checks whether two Positions are in range of each other. The range is the ChargingStation range<br>- **Parameters:**<br>  - Position p - one of the positions to check<br>  - Position q - the other position to check<br>- **Returns:**<br>  - boolean - whether the two positions are in range or not |

## 2.7 Position

Position keeps track of a latitude and longitude with some other fields which are used to calculate a new Position if moving from this position in a particular direction.

### 2.7.1 Field Summary

| |
|---|
| double latitude - the latitude of the Position |
| double longitude - the longitude of the Position |
| protected double hScore - can be used to store a heuristic score for this Position instance |
| Protected Direction dirToGetHere - can be used to indicate the direction from one Position to get to this Position |

The rest of the fields in Position are used to calculate the next Position if moving in a given Direction. The documentation for this can be found in the coursework specification.

### 2.7.2 Method Summary

| |
|---|
| nextPosition<br>- Calculates the latitude and longitude of the next Position if moving in a given direction from this Position<br>- **Parameters:**<br>  - Direction dir - the direction in which to move<br>- **Returns:**<br>  - Position - the new location if moving in this direction |
| inPlayArea() |

- Checks whether the current Position instance is in the boundary of the game
- **Returns:**
    - boolean - whether or not the Position is in the boundary

pythDistanceFrom
- Gives the Pythagorean (Euclidean) distance between two Positions.
- **Parameters:**
    - Position q - the position to calculate the distance
- **Returns:**
    double - the distance between this Position and the given Position.

## 2.8 StatefulDrone

This implements the abstract drone class and implements a search strategy for this method. The search strategy is a greedy best first search and more detail about this can be found in Section 3.

### 2.8.1 Method Summary

searchStrategy
- Calculates a list of good and bad stations from the Map and performs the main outline of the search.
- **Returns:**
    - void

private makeMoves
- Given a list of Directions, moves the drone in each Direction in turn.
- **Parameters:**
    - List<Direction> moves - the list of Directions the drone should move in
- **Returns:**
    - boolean - whether or not all of the moves were successful

private greedySearch
- Applies a greedy best first search to reach the given goal (within 50 moves) from the current Position, avoiding the bad charging stations
- **Parameters:**
    - ChargingStation goal - the target the drone wants to reach
    - List<ChargingStation> badStations - a list of stations to avoid in the search
- **Returns:**
    - List<Direction> - a list of Directions to move in to reach the goal from the current position.

private addAllNeighbors
- Finds all the valid (in the play area and not in range of a bad station) Positions one move away. It also applies a hScore to each Position, indicating how close it is to the goal.
- **Parameters:**
    - List<Position> open - a list of Positions already being considered
    - List<Position> closed - a list of Positions already considered
    - Position curPos - the position to get all the neighbors for

- ChargingStation goal - the target to reach
- **Returns:**
  - List<Position> - a list of Positions one move away with their hScores calculated

private checkPosInList
- Checks whether a position with a specific location is in a list, only considering its latitude and longitude.
- **Parameters:**
  - List<Position> list - the list to check
  - Position pos - the position to find in the list
- **Return:**
  - Boolean based on whether the position is in the list or not

private avoidanceStrategy
- Moves the drone to only avoid the bad stations until its 250 moves are finished or the drone runs out of power.
- **Parameters:**
  - List<ChargingStation> badStations - a list of charging stations to avoid
- **Returns:**
  - void

## 2.9 StatelessDrone

StatelessDrone implements the abstract drone class and implements a search strategy with no memory and limited lookahead (one move ahead only).

### 2.9.1 Method Summary

searchStrategy
- Finds the charging stations one move away from the drone's current position, sorts them by how good they are (if any) and picks the move that will gain the most coins/power or lose the least coins/power. If no stations are nearby, the drone moves randomly in a valid (in the play area) direction using the superclass's rnd field. Continues this recursively until the drone runs out of power or moves.
- **Returns:**
  - void

# 3. Stateful Drone Strategy

The stateful drone first separates all the good stations and bad stations. The good stations are those with positive power and coins, and the bad stations are the others. Since it has no limit memory, it can use all of these to find the charging station closest to it, unlike the stateless drone.

It then calculates a path to the nearest charging station using a greedy best first search strategy. A list of points to consider (open) and points already considered (closed) are set up. The drone then adds the neighbors of each the current point to the open list. The neighboring points cannot already be in the open list, or the closed list. The closed list ensures that the drone does not go back to a previously seen position and repeatedly search the same few positions to reach one goal (for example if it is stuck around a wall of bad stations). The stateless drone sometimes does this as it has no memory. This only matters for each target, and not overall as the drone may need to visit the same position twice to reach two different charging stations.

For the stateful search, the open list is sorted according to the hScore (heuristic score), which is the distance of the neighboring point from the goal. The Position closest to the goal is chosen as the drone's next move, following the greedy best first search paradigm. The process is repeated until the target is reached, or until a set number of moves (50 in this case) is reached.

If a path can be found, the stateful drone follows it to the target and repeats with the next charging station. If no path can be found, the station is added to a holding list (holdingStations) and it will revisit this list of stations once it has visited all the other good stations it can visit. The stateless drone does not need to do this since it has limited lookahead, so it's moves are random and it does not fixate on one target. The stateful drone does fixate on a target and can get stuck if there is no path from the current position (or no path at all), so a limit on the number of steps it can look ahead is set.

The search process is continued for the holding list as well. The drone may be unable to visit a few stations (though this only happens rarely), so it won't always get all the coins in a map. This is because the greedy search sometimes does not work well with finding a longer path around an obstacle. If this happens from two positions, then the drone does not visit it and instead moves to the avoidance strategy.

The stateless drone's avoidance strategy must constantly be performed (avoid bad stations at each step), but it can end up getting stuck in an area with only bad stations and be unable to move outside of this. The stateful drone is more targeted, so it avoids the bad stations to get to a goal, and once it has reached all the stations it can, it moves to just avoid bad stations. This works better since the stateful drone first targets all the good stations, and then focuses on finishing the game. The stateless drone will never know if it has visited all the good stations since it has no memory.

The next figures show the stateful and stateless searches for the map on 05/07/2019 with random seed 5678. The stateful drone has a much more targeted path for both it's search for positive stations and in its avoidance.

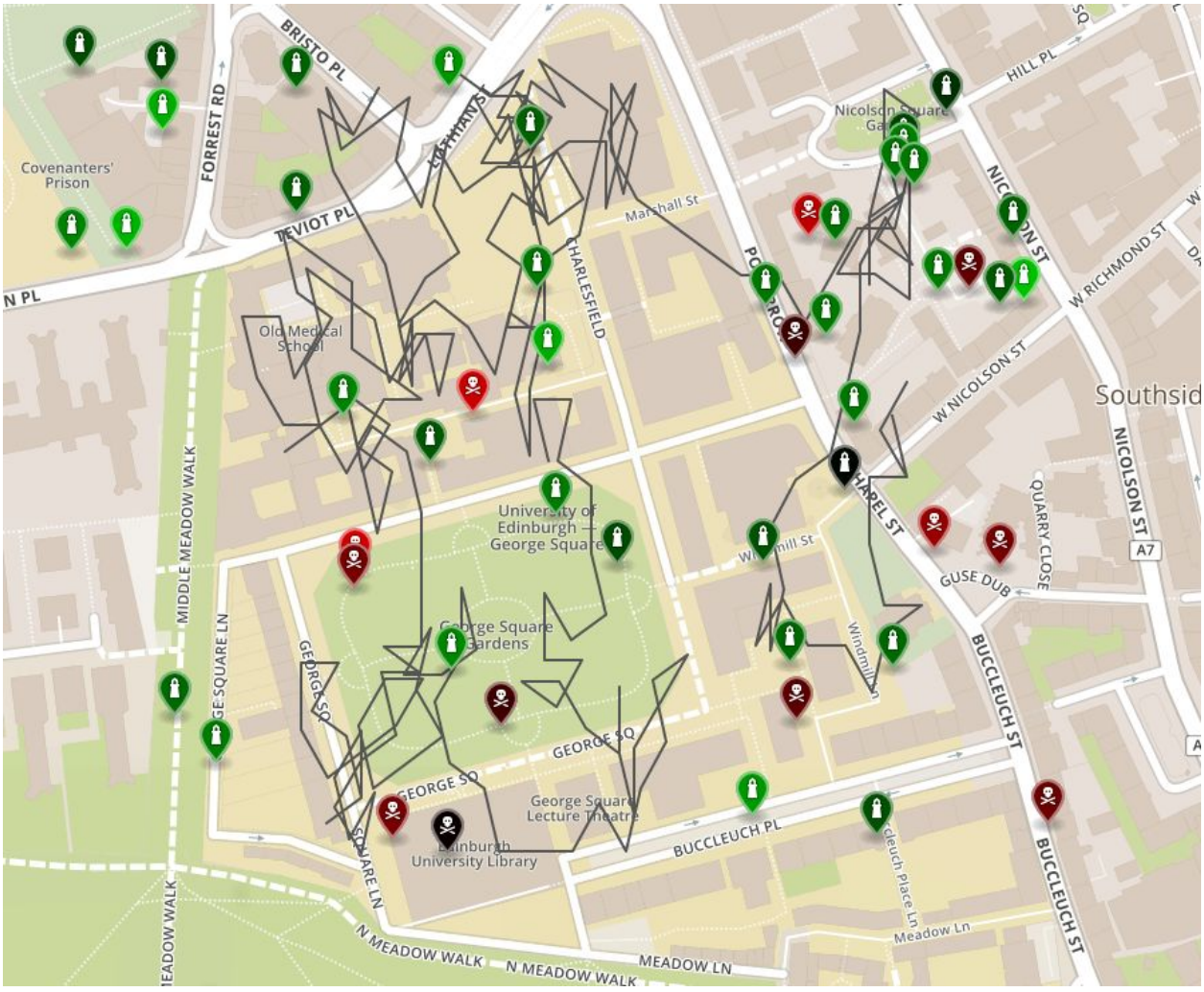


Figure 1: A stateless search for 05/07/2019 with seed 5678. The search does reasonably well but takes a while to move around the bad stations near the library and to move towards the positive stations.

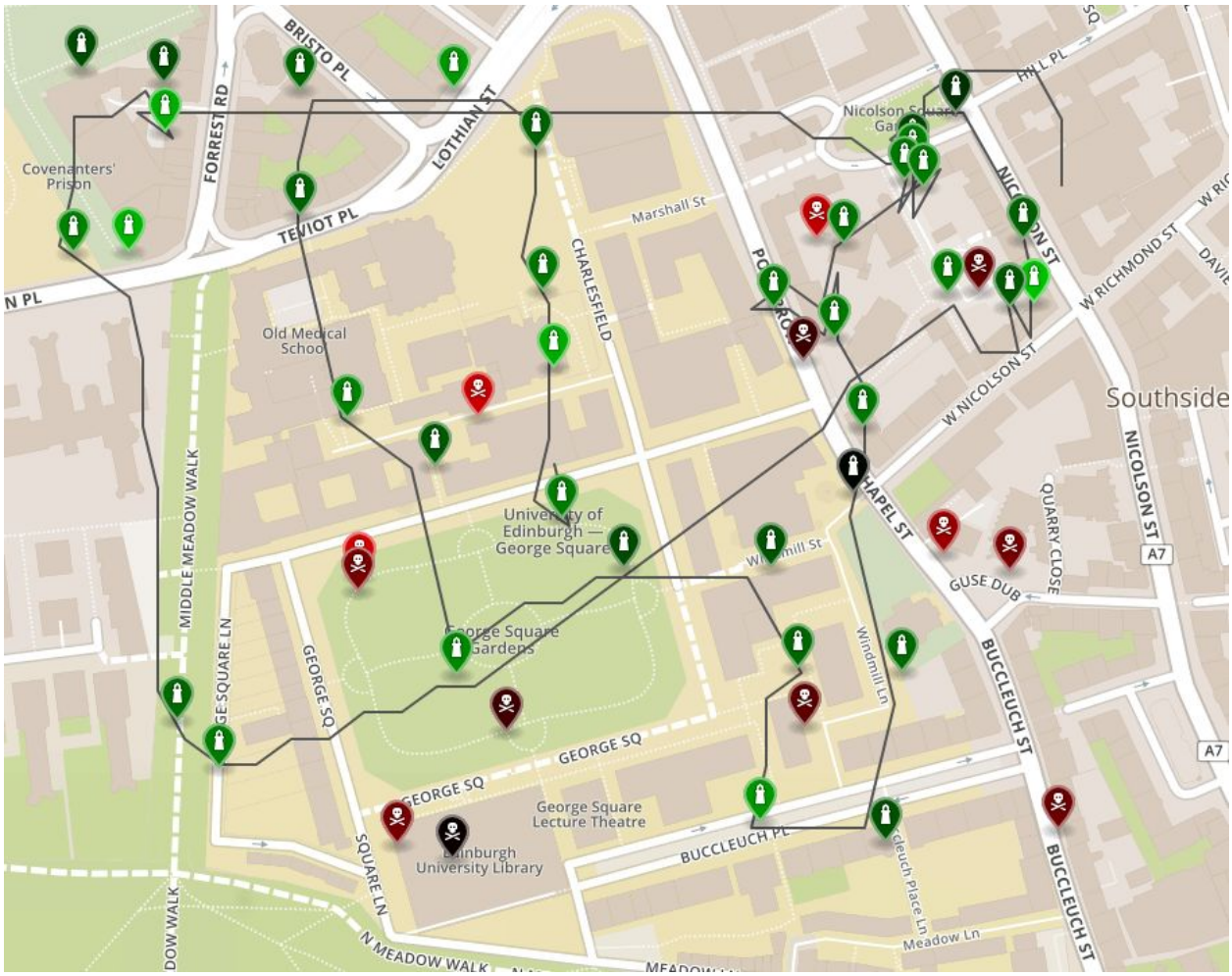


Figure 2: A stateful search for the date 05/07/2019 with seed 5678. The search is much clearer and targets a specific positive charging station.