AQ maps

s1858833

December 3, 2020

Contents

Class Hierarchy

Classes

- java.lang.Object
    - java.awt.geom.Point2D
        - java.awt.geom.Point2D.Double
            - uk.ac.ed.inf.aqmaps.geometry.Coords (in 3.1, page 31)
    - org.jgrapht.alg.tour.HamiltonianCycleAlgorithmBase
        - org.jgrapht.alg.tour.TwoOptHeuristicTSP
            - uk.ac.ed.inf.aqmaps.flightplanning.EnhancedTwoOptTSP (in 2.1, page 19)
    - org.jgrapht.graph.AbstractGraph
        - org.jgrapht.graph.AbstractBaseGraph
            - org.jgrapht.graph.SimpleGraph
                - org.jgrapht.graph.SimpleWeightedGraph
                    - uk.ac.ed.inf.aqmaps.flightplanning.SensorGraph (in 2.6, page 27)
                    - uk.ac.ed.inf.aqmaps.noflyzone.ObstacleGraph (in 6.1, page 50)
    - uk.ac.ed.inf.aqmaps.App (in 1.1, page 8)
    - uk.ac.ed.inf.aqmaps.Drone (in 1.2, page 9)
    - uk.ac.ed.inf.aqmaps.Move (in 1.3, page 10)
    - uk.ac.ed.inf.aqmaps.Results (in 1.4, page 12)
    - uk.ac.ed.inf.aqmaps.Sensor (in 1.5, page 13)
    - uk.ac.ed.inf.aqmaps.SensorMarkerFactory (in 1.6, page 15)
    - uk.ac.ed.inf.aqmaps.Settings (in 1.7, page 16)
    - uk.ac.ed.inf.aqmaps.W3W (in 1.8, page 17)
    - uk.ac.ed.inf.aqmaps.deserializers.CoordsDeserializer (in 5.1, page 46)
    - uk.ac.ed.inf.aqmaps.deserializers.SensorDeserializer (in 5.2, page 47)
    - uk.ac.ed.inf.aqmaps.deserializers.W3WDeserializer (in 5.3, page 48)
    - uk.ac.ed.inf.aqmaps.flightplanning.FlightCacheKey (in 2.2, page 22)
    - uk.ac.ed.inf.aqmaps.flightplanning.FlightCacheValue (in 2.3, page 23)
    - uk.ac.ed.inf.aqmaps.flightplanning.FlightPlan (in 2.4, page 24)
    - uk.ac.ed.inf.aqmaps.flightplanning.FlightPlanner (in 2.5, page 25)
    - uk.ac.ed.inf.aqmaps.flightplanning.WaypointNavigation (in 2.7, page 29)
    - uk.ac.ed.inf.aqmaps.geometry.Polygon (in 3.2, page 35)
    - uk.ac.ed.inf.aqmaps.io.FileOutputController (in 4.4, page 41)
    - uk.ac.ed.inf.aqmaps.io.ServerInputController (in 4.5, page 42)
    - uk.ac.ed.inf.aqmaps.io.WebServer (in 4.6, page 44)
    - uk.ac.ed.inf.aqmaps.noflyzone.ObstaclePathfinder (in 6.2, page 52)

- uk.ac.ed.inf.aqmaps.noflyzone.Obstacles (in 6.3, page 54)

Interfaces

- uk.ac.ed.inf.aqmaps.io.InputController (in 4.1, page 38)
- uk.ac.ed.inf.aqmaps.io.OutputController (in 4.2, page 39)
- uk.ac.ed.inf.aqmaps.io.Server (in 4.3, page 40)

Chapter 1

Package uk.ac.ed.inf.aqmaps

Package Contents *Page*

1.1 Class App

Flies a drone around Edinburgh to collect air quality data from sensors and create a map. This code follows the Google Java Style Guide at https://google.github.io/styleguide/javaguide.html

1.1.1 Declaration

```
public class App
 extends java.lang.Object
```

1.1.2 Constructor summary

App()

1.1.3 Method summary

main(String[]) Main method

1.1.4 Constructors

- **App**

```
public App()
```

1.1.5 Methods

- **main**

```
public static void main(java.lang.String[] args)
```

  - **Description**
    Main method
  - **Parameters**
    - `args` – a list of arguments in the form: day month year latitude longitude randomSeed portNumber [timeLimit]

1.2 Class Drone

Represents the drone. Performs route planning, than follows that plan to collect sensor data.

1.2.1 Declaration

```
public class Drone
 extends java.lang.Object
```

1.2.2 Field summary

input
output
settings

1.2.3 Constructor summary

Drone(Settings, InputController, OutputController)

1.2.4 Method summary

flyRoute(List, Results) Fly the drone along the route, collecting sensor data and adding it to the results.
planRoute() Plan the route that the drone will follow.
start() Start the drone and perform route planning and data collection for the given settings.

1.2.5 Fields

- `private final Settings` **settings**
- `private final io.InputController` **input**
- `private final io.OutputController` **output**

1.2.6 Constructors

- **Drone**

```
public Drone(Settings settings,io.InputController input,io.
    OutputController output)
```

  - **Parameters**
    - `settings` – the current Settings
    - `input` – the InputController which handles data input
    - `output` – the OutputController which handles data output

1.2.7 Methods

- **flyRoute**

```
private void flyRoute(java.util.List flightPlan,Results results)
```

  - **Description**
    Fly the drone along the route, collecting sensor data and adding it to the results.
  - **Parameters**
    - `flightPlan` – the flight plan for the drone to follow
    - `results` – the Results object to record the sensor data in

- **planRoute**

```
private java.util.List planRoute()
```

  - **Description**
    Plan the route that the drone will follow.

  - **Returns** – a list of Moves specifying the route

- **start**

  ```
  public void start()
  ```

  - **Description**
    Start the drone and perform route planning and data collection for the given settings.

1.3 Class Move

A class representing a single move to be made by the drone.

1.3.1 Declaration

```
public class Move
 extends java.lang.Object
```

1.3.2 Field summary

after
before
direction
sensorW3W

1.3.3 Constructor summary

Move(Coords, Coords, int, W3W)

1.3.4 Method summary

getAfter()
getBefore()
getDirection()
getSensorW3W()
toString()

1.3.5 Fields

- `private final geometry.Coords` **before**
- `private final geometry.Coords` **after**
- `private final int` **direction**
- `private final W3W` **sensorW3W**

1.3.6 Constructors

- **Move**

  ```
  public Move(geometry.Coords before,geometry.Coords after,int
      direction,W3W sensorW3W)
  ```

  - **Parameters**
    - `before` – the position of the drone before the move
    - `after` – the position of the drone after the move
    - `direction` – the direction of the move in degrees, from 0 to 350 anticlockwise starting from east
    - `sensorW3W` – the location of the sensor visited by the drone at the end of this move, or null if no sensor is visited

1.3.7 Methods

- **getAfter**

  ```
  public geometry.Coords getAfter()
  ```

  - **Returns** – the position of the drone after making the move

- **getBefore**

  ```
  public geometry.Coords getBefore()
  ```

  - **Returns** – the position of the drone before making the move

- **getDirection**

  ```
  public int getDirection()
  ```

  - **Returns** – the direction of move in degrees

- **getSensorW3W**

  ```
  public W3W getSensorW3W()
  ```

  - **Returns** – the W3W of the sensor that this move reaches, or null if it does not reach a sensor

- **toString**

  ```
  public java.lang.String toString()
  ```

1.4 Class Results

Holds and processes the calculated flightpath and collected sensor data

1.4.1 Declaration

```
public class Results
 extends java.lang.Object
```

1.4.2 Field summary

flightpath The planned flightpath of the drone as a list of Moves
sensorsVisited A map from W3W sensor locations to their corresponding Sensor data.
sensorW3Ws A list of the W3W of the sensors that are planned to be visited.

1.4.3 Constructor summary

Results(List) Constructor

1.4.4 Method summary

createFlightpathLineString() Creates a GeoJSON LineString which shows the path that shows the path taken by the drone.
createSensorMarkers() Creates Features containing Points which mark the position and reading/status of the sensors.
getFlightpathString() Gets a flightpath String of the following format: 1,[startLng],[startLat],[angle],[endLng],[endLat],[sensor w3w or null]“n 2,[startLng],[startLat],[angle],[endLng],[endLat],[sensor w3w or null]“n ...
getMapGeoJSON() Creates a GeoJSON string of a map which displays the flightpath of the drone and markers displaying the readings or status of the sensors.
recordFlightpath(List) Adds a calculated flight to the results
recordSensorReading(Sensor) Add a sensor with its readings to the results

1.4.5 Fields

- `private final java.util.Map` **sensorsVisited**
  - A map from W3W sensor locations to their corresponding Sensor data.
- `private final java.util.List` **sensorW3Ws**
  - A list of the W3W of the sensors that are planned to be visited.
- `private java.util.List` **flightpath**
  - The planned flightpath of the drone as a list of Moves

1.4.6 Constructors

- **Results**

  ```
  public Results(java.util.List sensorW3Ws)
  ```

  - **Description**
    Constructor
  - **Parameters**
    * `sensorW3Ws` – a list of sensor locations as W3W that the drone is visiting

1.4.7 Methods

- **createFlightpathLineString**

  ```
  private com.mapbox.geojson.Feature createFlightpathLineString()
  ```

  - **Description**
    Creates a GeoJSON LineString which shows the path that shows the path taken by the drone.
  - **Returns** – a Feature containing the LineString

- **createSensorMarkers**

  ```
  private java.util.List createSensorMarkers()
  ```

  - **Description**
    Creates Features containing Points which mark the position and reading/status of the sensors.
  - **Returns** – a List of Features

- **getFlightpathString**

  ```
  public java.lang.String getFlightpathString()
  ```

  - **Description**
    Gets a flightpath String of the following format: 1,[startLng],[startLat],[angle],[endLng],[endLat],[sensor w3w or null]“n 2,[startLng],[startLat],[angle],[endLng],[endLat],[sensor w3w or null]“n ...
  - **Returns** – the flightpath String

- **getMapGeoJSON**

```
public java.lang.String getMapGeoJSON()
```

- **Description**
  Creates a GeoJSON string of a map which displays the flightpath of the drone and markers displaying the readings or status of the sensors.
- **Returns** – a String of the GeoJSON

- **recordFlightpath**

```
public void recordFlightpath(java.util.List flightpath)
```

- **Description**
  Adds a calculated flight to the results
- **Parameters**
  - `flightpath` – a list of Moves representing the flightpath

- **recordSensorReading**

```
public void recordSensorReading(Sensor sensor)
```

- **Description**
  Add a sensor with its readings to the results
- **Parameters**
  - `sensor` – the Sensor

1.5 Class Sensor

A sensor with a battery level and reading.

1.5.1 Declaration

```
public class Sensor
 extends java.lang.Object
```

1.5.2 Field summary

battery
location
reading

1.5.3 Constructor summary

Sensor(W3W, float, String) Constructor

1.5.4 Method summary

getBattery()
getLocation()
getReading()

1.5.5 Fields

- `private final W3W` **location**
- `private final float` **battery**
- `private final java.lang.String` **reading**

1.5.6 Constructors

- **Sensor**

```
public Sensor(W3W w3wLocation, float battery, java.lang.String
    reading)
```

  - **Description**
    Constructor
  - **Parameters**
    * `w3wLocation` – the W3W location of the sensor
    * `battery` – the current battery level of the sensor, as a percentage
    * `reading` – the reading of the sensor as a String

1.5.7 Methods

- **getBattery**

```
public float getBattery()
```

  - **Returns** – the battery level of this sensor as a percentage

- **getLocation**

```
public W3W getLocation()
```

  - **Returns** – the W3W location of this sensor

- **getReading**

```
public java.lang.String getReading()
```

- **Returns** – the reading of the sensor, as a String. If the battery level is 10% or greater this should contain a float value, but if it is less than 10% the reading cannot be trusted and may be incorrect, null or NaN.

1.6 Class SensorMarkerFactory

A factory which constructs sensor markers which displays the location, status and reading of a sensor. This factory does not produce hypothetical SensorMarker instances, but Feature instances instead, since Feature cannot be subclassed due to its lack of a public constructor.

1.6.1 Declaration

```
public class SensorMarkerFactory
 extends java.lang.Object
```

1.6.2 Constructor summary

SensorMarkerFactory()

1.6.3 Method summary

createPoint(W3W, String) Create a Feature containing a Point marker with a position, colour, and no marker symbol.
createPoint(W3W, String, String) Create a Feature with a position, colour, and marker symbol.
getMarkerSymbol(double) Gets the marker symbol string for the given pollution level
getRgbString(double) Converts a pollution level to a corresponding RGB string.
getSensorMarker(W3W, Sensor) Creates a marker located at the position of this sensor.

1.6.4 Constructors

- **SensorMarkerFactory**

```
public SensorMarkerFactory()
```

1.6.5 Methods

- **createPoint**

```
private com.mapbox.geojson.Feature createPoint(W3W w3w, java.lang
    .String rgbString)
```

- **Description**
  Create a Feature containing a Point marker with a position, colour, and no marker symbol.
- **Parameters**
  - `w3w` – the W3W location of the sensor
  - `rgbString` – the RGB colour string of the point
- **Returns** – a Feature containing a Point

• **createPoint**

```
private com.mapbox.geojson.Feature createPoint(W3W w3w,java.lang
    .String rgbString ,java.lang.String markerSymbol)
```

- **Description**
  Create a Feature with a position, colour, and marker symbol.
- **Parameters**
  - `w3w` – the W3W location of the sensor
  - `rgbString` – the RGB colour string of the point
  - `markerSymbol` – the marker symbol string
- **Returns** – a Feature containing a Point

• **getMarkerSymbol**

```
private java.lang.String getMarkerSymbol(double pollutionLevel)
```

- **Description**
  Gets the marker symbol string for the given pollution level
- **Parameters**
  - `pollutionLevel` – the pollution level as a double between 0 and 256
- **Returns** – a String describing the marker symbol

• **getRgbString**

```
private java.lang.String getRgbString(double pollutionLevel)
```

- **Description**
  Converts a pollution level to a corresponding RGB string.
- **Parameters**
  - `pollutionLevel` – the pollution level as a double between 0 and 256
- **Returns** – the rgb string #xxxxxx of the colour representing the pollution level

- **getSensorMarker**

  ```
  public com.mapbox.geojson.Feature getSensorMarker(W3W w3w, Sensor
      sensor)
  ```

  - **Description**
    Creates a marker located at the position of this sensor. If a sensor reading was taken successfully the marker is coloured and assigned a symbol based on the reading, and if it has low battery or was not visited, assigns different symbols.
  - **Parameters**
    * `w3w` – the location of the sensor as a W3W
    * `sensor` – the Sensor containing the sensor data, or null if the sensor was not visited
  - **Returns** – a Feature containing a Point and various attributes describing the marker

1.7 Class Settings

Holds the settings derived from the command line arguments.

1.7.1 Declaration

```
public class Settings
 extends java.lang.Object
```

1.7.2 Field summary

day
DEFAULT_TIME_LIMIT_SECONDS The default value in seconds of the time limit on the flight planning algorithm, to be used when a time limit has not been received as an argument.
maxRunTime
month
port
randomSeed
startCoords
year

1.7.3 Constructor summary

Settings(String[])

1.7.4 Method summary

getDay()
getMaxRunTime()
getMonth()
getPort()
getRandomSeed()
getStartCoords()
getYear()

1.7.5 Fields

- `private static final double` **DEFAULT_TIME_LIMIT_SECONDS**
  - The default value in seconds of the time limit on the flight planning algorithm, to be used when a time limit has not been received as an argument. Increasing this has highly diminishing returns. See `FlightPlanner` (in 2.5, page 25). Default set to 0 for random repeatability.
- `private final int` **day**
- `private final int` **month**
- `private final int` **year**
- `private final geometry.Coords` **startCoords**
- `private final int` **randomSeed**
- `private final int` **port**
- `private final double` **maxRunTime**

1.7.6 Constructors

- **Settings**

  ```
  public Settings(java.lang.String[] args)
  ```

  - **Parameters**
    - `args` – the input command line args

1.7.7 Methods

- **getDay**

  ```
  public int getDay()
  ```

  - **Returns** – the day to generate the map for

- **getMaxRunTime**

  ```
  public double getMaxRunTime()
  ```

  - **Returns** – the maximum run time of the flight planner in seconds

- **getMonth**

  ```
  public int getMonth()
  ```

  - **Returns** – the month to generate the map for

- **getPort**

  ```
  public int getPort()
  ```

  - **Returns** – the port number of the server

- **getRandomSeed**

  ```
  public int getRandomSeed()
  ```

  - **Returns** – the random seed to use in the algorithms

- **getStartCoords**

  ```
  public geometry.Coords getStartCoords()
  ```

  - **Returns** – the starting coordinates of the drone

- **getYear**

  ```
  public int getYear()
  ```

  - **Returns** – the year to generate the map for

1.8 Class W3W

Holds what3words coordinate and word information.

1.8.1 Declaration

```
public class W3W
 extends java.lang.Object
```

1.8.2 Field summary

coordinates The coordinates of the centre of the W3W square
words

1.8.3 Constructor summary

W3W(Coords, String)

1.8.4 Method summary

getCoordinates()
getWords()

1.8.5 Fields

- `private final geometry.Coords` **coordinates**
  - The coordinates of the centre of the W3W square
- `private final java.lang.String` **words**

1.8.6 Constructors

- **W3W**

  ```
  public W3W(geometry.Coords coordinates ,java.lang.String words)
  ```

  - **Parameters**
    * `coordinates` – the coordinates of the centre of the W3W square
    * `words` – the 3 words

1.8.7 Methods

- **getCoordinates**

  ```
  public geometry.Coords getCoordinates()
  ```

  - **Returns** – the coordinates of the centre of the W3W square
- **getWords**

```
public java.lang.String getWords()
```

- **Returns** – words the 3 words

Chapter 2

Package uk.ac.ed.inf.aqmaps.flightplanning

Package Contents *Page*

2.1 Class EnhancedTwoOptTSP

A modified version of TwoOptHeuristicTSP from JGraphT which is used to compute tours which visit all sensors and returns to the starting point. Source code of TwoOptHeuristicTSP can be found here (GitHub).

JGraphT main website, GitHub source, accessed 30/11/2020

The following JavaDoc is unchanged from the the original:

The 2-opt heuristic algorithm for the TSP problem.

The travelling salesman problem (TSP) asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?".

This is an implementation of the 2-opt improvement heuristic algorithm. The algorithm generates *passes* initial tours and then iteratively improves the tours until a local minimum is reached. In each iteration it applies the best possible 2-opt move which means to find the best pair of edges $(i,i+1)$ and $(j,j+1)$ such that replacing them with (i,j) and $(i+1,j+1)$ minimizes the tour length. The default initial tours use RandomTour, however an alternative algorithm can be provided to create the initial tour. Initial tours generated using NearestNeighborHeuristicTSP give good results and performance.

See wikipedia for more details.

This implementation can also be used in order to try to improve an existing tour. See method `improveTour(GraphPath)` }.

2.1.1 Declaration

```
public class EnhancedTwoOptTSP
 extends org.jgrapht.alg.tour.TwoOptHeuristicTSP
```

2.1.2 Field summary

flightPlanner The FlightPlanner for computing tour weights
graph A graph representation of the sensors
index
n
revIndex
start The start and end position of the drone

2.1.3 Constructor summary

EnhancedTwoOptTSP(int, int, Coords, FlightPlanner) Constructor

2.1.4 Method summary

applyMove(int[], int[], int, int) This code is part of the original library's improve(), but it is extracted into a method to avoid duplication as this class uses it in more than one place.
getDirectLength(List) (New method, not in the library).
getTour(Graph) Computes a tour by first using JGraphT's TwoOptHeuristicTSP (the superclass of this) to find a short tour using the edge weights in the provided graph, which are straight line (obstacle avoiding) distance measures.
getTourAsList(int[]) New method, not in the library.
improve(int[]) A modified version of the improve() method in the library.
improveTour(GraphPath) (Code unchanged from library code other than type parameters)

init(Graph) (Reduced version of library code as we aren't using dist[][])
pathToTour(GraphPath) (Code and Javadoc unchanged from library other than type parameters)
tourToPath(int[]) (Code and Javadoc unchanged from library other than type parameters)

2.1.5 Fields

- `private final uk.ac.ed.inf.aqmaps.geometry.Coords` **start**
  - The start and end position of the drone
- `private final FlightPlanner` **flightPlanner**
  - The FlightPlanner for computing tour weights
- `private org.jgrapht.Graph` **graph**
  - A graph representation of the sensors
- `private int` **n**
- `private java.util.Map` **index**
- `private java.util.Map` **revIndex**

2.1.6 Constructors

- **EnhancedTwoOptTSP**

```
public EnhancedTwoOptTSP(int passes, int seed, uk.ac.ed.inf.aqmaps
    .geometry.Coords start, FlightPlanner flightPlanner)
```

  - **Description**
    Constructor
  - **Parameters**
    - `passes` – how many initial random tours to check when running 2-opt
    - `seed` – the random seed
    - `start` – the start position of the drone
    - `flightPlanner` – the FlightPlanner to use for the second 2-opt pass to compute tour weights as the number of moves needed by the drone

2.1.7 Methods

- **applyMove**

```
private void applyMove(int[] tour, int[] newTour, int i, int j)
```

  - **Description**
    This code is part of the original library's improve(), but it is extracted into a method to avoid duplication as this class uses it in more than one place.
  - **Parameters**
    * `tour` – the tour to apply the move to
    * `newTour` – the new tour with the move applied
    * `i` – the first swap point
    * `j` – the second swap point

- **getDirectLength**

```
private double getDirectLength(java.util.List tourList)
```

  - **Description**
    (New method, not in the library). Computes the length of tour using the direct real distance between the vertices.
  - **Parameters**
    * `tourList` – the tour as a list of Coords
  - **Returns** – the length of the tour in degrees, using euclidean distance (not using the flight planner)

- **getTour**

```
public org.jgrapht.GraphPath getTour(org.jgrapht.Graph graph)
```

  - **Description**
    Computes a tour by first using JGraphT's TwoOptHeuristicTSP (the superclass of this) to find a short tour using the edge weights in the provided graph, which are straight line (obstacle avoiding) distance measures. Then, it runs a second pass of 2-opt to further improve upon the tour by instead using a FlightPlanner to generate the actual drone moves along the tour and using the number of moves as the weight of a tour.
  - **Parameters**
    * `graph` – the input sensor graph containing the start location and the sensors, and edge weights of the shortest path between two points which avoids obstacles.
  - **Returns** – the tour as a GraphPath

- **getTourAsList**

```
private java.util.List getTourAsList(int[] tour)
```

- **Description**

  New method, not in the library. Converts a tour as an array to the corresponding list of Coords.
- **Parameters**
  * `tour` – a tour as an int[]
- **Returns** – a List of Coords

• **improve**

```
private int[] improve(int[] tour)
```

- **Description**

  A modified version of the improve() method in the library. The main difference is that this version does not have a distance matrix due to the fact that the distance between a pair of sensors depends heavily on the rest of the tour that it is in. Instead of calculating the distance change with

  `double change = dist[ci][cj] + dist[ci1][cj1] - dist[ci][ci1] - dist[cj][cj1];`

  we instead use the FlightPlanner to plan a route along the tour, and compare its length to the unmodified tour. Since the tours being improved by this algorithm have already been through 2-opt with direct distance measures, drastic changes to the tour are unlikely to improve it, and are also computationally expensive since they are longer and may collide with more buildings and so on. To avoid this, only tours which are close in distance to the original will be tried.

  Original JGraphT JavaDoc:

  Improve the tour using the 2-opt heuristic. In each iteration it applies the best possible 2-opt move which means to find the best pair of edges $(i,i+1)$ and $(j,j+1)$ such that replacing them with (i,j) and $(i+1,j+1)$ minimizes the tour length.

  The returned array instance might or might not be the input array.
- **Parameters**
  * `tour` – the input tour
- **Returns** – a possibly improved tour

• **improveTour**

```
public org.jgrapht.GraphPath improveTour(org.jgrapht.GraphPath
    graphPath)
```

- **Description**

  (Code unchanged from library code other than type parameters)

  Try to improve a tour by running the 2-opt heuristic using the FlightPlanner to measure the length of tours.

  - **Parameters**
    - `graphPath` – a tour
  - **Returns** – a possibly improved tour

- **init**

  ```
  private void init(org.jgrapht.Graph graph)
  ```

  - **Description**
    (Reduced version of library code as we aren't using dist[][])
    Initialize graph and mapping to integer vertices.
  - **Parameters**
    - `graph` – the input graph

- **pathToTour**

  ```
  private int[] pathToTour(org.jgrapht.GraphPath path)
  ```

  - **Description**
    (Code and Javadoc unchanged from library other than type parameters)
    Transform from a path representation to an array representation.
  - **Parameters**
    - `path` – graph path
  - **Returns** – an array containing the index of the vertices of the tour

- **tourToPath**

  ```
  private org.jgrapht.GraphPath tourToPath(int[] tour)
  ```

  - **Description**
    (Code and Javadoc unchanged from library other than type parameters)
    Transform from an array representation to a graph path.
  - **Parameters**
    - `tour` – an array containing the index of the vertices of the tour
  - **Returns** – a graph path

2.1.8 Members inherited from class TwoOptHeuristicTSP

`org.jgrapht.alg.tour.TwoOptHeuristicTSP`

- `private int` **createInitialTour**`()`
- `private` **dist**
- `public GraphPath` **getTour**`(org.jgrapht.Graph` **arg0**`)`
- `private` **graph**
- `private int` **improve**`(int[]` **arg0**`)`
- `public GraphPath` **improveTour**`(org.jgrapht.GraphPath` **arg0**`)`
- `private` **index**
- `private void` **init**`(org.jgrapht.Graph` **arg0**`)`
- `private final` **initializer**
- `private final` **minCostImprovement**
- `private` **n**
- `private final` **passes**
- `private int` **pathToTour**`(org.jgrapht.GraphPath` **arg0**`)`
- `private` **revIndex**
- `private GraphPath` **tourToPath**`(int[]` **arg0**`)`

2.1.9 Members inherited from class HamiltonianCycleAlgorithmBase

`org.jgrapht.alg.tour.HamiltonianCycleAlgorithmBase`

- `protected void` **checkGraph**`(org.jgrapht.Graph` **arg0**`)`
- `protected GraphPath` **closedVertexListToTour**`(java.util.List` **arg0**`, org.jgrapht.Graph` **arg1**`)`
- `protected GraphPath` **edgeSetToTour**`(java.util.Set` **arg0**`, org.jgrapht.Graph` **arg1**`)`
- `protected GraphPath` **getSingletonTour**`(org.jgrapht.Graph` **arg0**`)`
- `protected void` **requireNotEmpty**`(org.jgrapht.Graph` **arg0**`)`
- `protected GraphPath` **vertexListToTour**`(java.util.List` **arg0**`, org.jgrapht.Graph` **arg1**`)`

2.2 Class FlightCacheKey

A class which holds data about the input values of the flight planning algorithm from a position to a sensor, potentially with a next sensor. This is for use in the cache, so stores hashed value directly in order to save memory and time calculating extra hashes.

2.2.1 Declaration

```
public class FlightCacheKey
 extends java.lang.Object
```

2.2.2 Field summary

currentHash
hashcode A hash of the parameters
nextHash
startHash

2.2.3 Constructor summary

FlightCacheKey(Coords, Coords, Coords) Constructor

2.2.4 Method summary

equals(Object) Needed to work with a HashMap, automatically generated by IntelliJ.

hashCode() Since this class stores the hashcode directly, we do not do any computation and just return it.

2.2.5 Fields

- `private final int` **hashcode**
  - A hash of the parameters
- `private final int` **startHash**
- `private final int` **currentHash**
- `private final int` **nextHash**

2.2.6 Constructors

- **FlightCacheKey**

```
public FlightCacheKey(uk.ac.ed.inf.aqmaps.geometry.Coords
    startPosition ,uk.ac.ed.inf.aqmaps.geometry.Coords
    currentTarget ,uk.ac.ed.inf.aqmaps.geometry.Coords nextTarget)
```

  - **Description**
    Constructor
  - **Parameters**
    - `startPosition` – the start position of the drone.
    - `currentTarget` – the current target
    - `nextTarget` – the next target if there is one, or null otherwise

2.2.7 Methods

- **equals**

```
public boolean equals(java.lang.Object o)
```

  - **Description**
    Needed to work with a HashMap, automatically generated by IntelliJ.

- **hashCode**

  ```
  public int hashCode()
  ```

  - **Description**
    Since this class stores the hashcode directly, we do not do any computation and just return it.

2.3 Class FlightCacheValue

A class which holds data about the output values of the flight planning algorithm from a position to a sensor, potentially with a next sensor. This does not hold the actual tour, and is only used for the size of the tour, as it would use a lot of memory.

2.3.1 Declaration

```
public class FlightCacheValue
 extends java.lang.Object
```

2.3.2 Field summary

endPosition
length

2.3.3 Constructor summary

FlightCacheValue(int, Coords) Constructor

2.3.4 Method summary

getEndPosition()
getLength()

2.3.5 Fields

- `private final int` **length**
- `private final uk.ac.ed.inf.aqmaps.geometry.Coords` **endPosition**

2.3.6 Constructors

- **FlightCacheValue**

  ```
  public FlightCacheValue(int length,uk.ac.ed.inf.aqmaps.geometry.
      Coords endPosition)
  ```

  - **Description**
    Constructor
  - **Parameters**
    * `length` – the number of moves in this flight path section
    * `endPosition` – the ending position of the drone in this flight path section

2.3.7 Methods

- **getEndPosition**

  ```
  public uk.ac.ed.inf.aqmaps.geometry.Coords getEndPosition()
  ```

  - **Returns** – the ending position of the drone in this flight path section

- **getLength**

  ```
  public int getLength()
  ```

  - **Returns** – the number of moves in this flight path section

2.4 Class FlightPlan

A wrapper class for a flight plan as a list of moves, and the random seed that was used to generate it. This is needed so that results can be sorted by seed before finding the minimum, allowing for consistent operation even when concurrency is used.

2.4.1 Declaration

```
public class FlightPlan
 extends java.lang.Object
```

2.4.2 Field summary

moves
seed

2.4.3 Constructor summary

FlightPlan(int, List)

2.4.4 Method summary

getMoves()
getMovesWithLimit() Get the list of moves in the flight plan, limited to a maximum of 150 moves.
getSeed()
toString()

2.4.5 Fields

- `private final int` **seed**
- `private final java.util.List` **moves**

2.4.6 Constructors

- **FlightPlan**

  ```
  public FlightPlan(int seed,java.util.List moves)
  ```

  - **Parameters**
    * `seed` – the random seed that his flight plan used
    * `moves` – the list of move which makes up the flight plan

2.4.7 Methods

- **getMoves**

  ```
  public java.util.List getMoves()
  ```

- **getMovesWithLimit**

  ```
  public java.util.List getMovesWithLimit()
  ```

  - **Description**
    Get the list of moves in the flight plan, limited to a maximum of 150 moves. In testing with the current possible input values, this never came close actually limiting the number of moves.
  - **Returns** – a list of moves of length = 150

- **getSeed**

  ```
  public int getSeed()
  ```

- **toString**

  ```
  public java.lang.String toString()
  ```

2.5 Class FlightPlanner

Handles the creation of a flight plan for the drone. Uses JGraphT's TwoOptHeuristicTSP algorithm as part of process, which was the best performing of JGraphT's Hamiltonian Cycle algorithms, however this could be changed easily.

2.5.1 Declaration

```
public class FlightPlanner
 extends java.lang.Object
```

2.5.2 Field summary

atomicSeedCounter Holds the next random seed to be used when creating the next plan.
cache Caches the number of moves and end position of navigating from a point to a target, with a particular following target.
CORNER_CUT_RADIUS_FRACTION See `cutCorner(Coords, Coords, Coords)` This value performed the best in testing.
ITERATIONS When the time limit is off, this is the number of times to run the algorithm before picking the shortest.
MAX_ITERATIONS When the time limit is on, this is the maximum possible number of iterations.
obstacles
sensorCoordsW3WMap A map from coordinates to the W3W of a sensor at that location.
startTime The System.nanoTime() at which we started running flight planning algorithms.
timeLimitNanos The approximate maximum run time for flight planning in nanoseconds (to work with System.nanoTime()).
timeLimitOn The time limit can be turned off to run for a specified `ITERATIONS` number of iterations to produce consistent output.
timerStarted
TWO_OPT_PASSES The number of initial tours to try when running 2-opt.

2.5.3 Constructor summary

FlightPlanner(Obstacles, List, int, double) Construct a flight planner with the given time limit in seconds.

2.5.4 Method summary

computeFlightLength(List) Computes the length of a flight plan which follows the given sensor coordinate tour.
constructFlightAlongTour(List) Create a flight plan for the drone along the given sensor tour.

createBestFlightPlan(Coords) Create a flight plan for the drone which visits all sensors and returns to the start.
createPlan(Coords, SensorGraph) Creates a flight plan for the drone which visits all sensors and returns to the start.
cutCorner(Coords, Coords, Coords) Attempts to cut the corner by moving from the target a distance of WaypointNavigation#SENSOR_RANGE * `CORNER_CUT_RADIUS_FRACTION` in a direction between the current position and the next target.

2.5.5 Fields

- `private static final int` **TWO_OPT_PASSES**
  - The number of initial tours to try when running 2-opt. Increasing this does not necessarily reduce average path length, and may increase it, since it will introduce less variability in what goes into the flight-plan mode 2-opt. We do not want the tours given to the improver to all be similarly optimal, since the two tour weight measures are different. There is a trade off to be had between this constant and the number of times the algorithm runs

- `private static final int` **ITERATIONS**
  - When the time limit is off, this is the number of times to run the algorithm before picking the shortest. Can be changed to trade off for speed and efficacy. Increasing it has diminishing returns.

- `private static final int` **MAX_ITERATIONS**
  - When the time limit is on, this is the maximum possible number of iterations. If this many are run before the time is up, it stops and does not wait for the timer. The current value is high enough for this to never happen.

- `private static final double` **CORNER_CUT_RADIUS_FRACTION**
  - See `cutCorner(Coords, Coords, Coords)` This value performed the best in testing.

- `private final uk.ac.ed.inf.aqmaps.noflyzone.Obstacles` **obstacles**

- `private final java.util.Map` **sensorCoordsW3WMap**
  - A map from coordinates to the W3W of a sensor at that location.

- `private final java.util.Map` **cache**
  - Caches the number of moves and end position of navigating from a point to a target, with a particular following target. Used in `computeFlightLength(List)` . This does not cache the actual moves that would be needed to use the cache in `constructFlightAlongTour(List)` since the memory use would be too high and almost all of the moves stored would not be used since we only need to construct the tour once. Using a cache in testing resulted in a speedup of 60-70%. Since `computeFlightLength(List)` will be run in parallel, we use a ConcurrentHashMap to prevent blocking, which was about 60% faster than a Hashtable in testing (on 6 cores/12 threads).

- `private final java.util.concurrent.atomic.AtomicInteger` **atomicSeed-Counter**
  - Holds the next random seed to be used when creating the next plan. This is used so when the time limit is being used to cut off execution, it will have run using the first seeds in the planned sequence (seed, seed+1, seed+2, ...). The alternative is to create this sequence in advance and then run through it with map(), however when executed in parallel they would not be executed starting from the front, and the seeds that were chosen would end up being essentially random.

    Using this allows to say with confidence that if the algorithm produced a flight plan in n iterations, that flight plan used a random seed between the command line input seed and (seed+n), and you would be able to generate the same flight plan again.
- `private final boolean` **timeLimitOn**
  - The time limit can be turned off to run for a specified `ITERATIONS` number of iterations to produce consistent output.
- `private final java.util.concurrent.atomic.AtomicBoolean` **timerStarted**
- `private long` **timeLimitNanos**
  - The approximate maximum run time for flight planning in nanoseconds (to work with System.nanoTime()). The algorithm will repeat as many times as possible with different random seeds within this time frame, up to a maximum of MAX_ITERATIONS and then the best flight path will be chosen. High values of this have highly diminishing returns.
- `private double` **startTime**
  - The System.nanoTime() at which we started running flight planning algorithms.

2.5.6 Constructors

- **FlightPlanner**

```
public FlightPlanner(uk.ac.ed.inf.aqmaps.noflyzone.Obstacles
    obstacles,java.util.List sensorW3Ws,int randomSeed,double
    timeLimit)
```

  - **Description**

    Construct a flight planner with the given time limit in seconds. If the time limit is not greater than 0, turns it off and uses a maximum number of iterations instead.
  - **Parameters**
    - `obstacles` – the Obstacles containing the no-fly zones
    - `sensorW3Ws` – the W3W locations of the sensors
    - `randomSeed` – the initial random seed to use
    - `timeLimit` – the time limit for the algorithm in seconds. If it is equal to 0 then disables the time limit and runs for a fixed number of iterations.

2.5.7 Methods

- **computeFlightLength**

  ```
  public int computeFlightLength(java.util.List tour)
  ```

  - **Description**
    Computes the length of a flight plan which follows the given sensor coordinate tour.
  - **Parameters**
    - `tour` – a list of Coords specifying the order to visit the sensors
  - **Returns** – the number of moves in the flight plan

- **constructFlightAlongTour**

  ```
  private java.util.List constructFlightAlongTour(java.util.List
      tour)
  ```

  - **Description**
    Create a flight plan for the drone along the given sensor tour.
  - **Parameters**
    - `tour` – a list of Coords specifying the order to visit the sensors
  - **Returns** – a list of Moves representing the flight plan

- **createBestFlightPlan**

  ```
  public java.util.List createBestFlightPlan(uk.ac.ed.inf.aqmaps.
      geometry.Coords startPosition)
  ```

  - **Description**
    Create a flight plan for the drone which visits all sensors and returns to the start. Runs the algorithm a large number of times with different random seeds, in parallel, and chooses the shortest.
  - **Parameters**
    - `startPosition` – the starting position of the drone
  - **Returns** – a list of Moves representing the flight plan

- **createPlan**

  ```
  private FlightPlan createPlan(uk.ac.ed.inf.aqmaps.geometry.
      Coords startPosition, SensorGraph sensorGraph)
  ```

- **Description**
  Creates a flight plan for the drone which visits all sensors and returns to the start. First uses an two stage 2-opt heuristic (see `EnhancedTwoOptTSP` (in 2.1, page 19)) to generate a tour, then constructs a flight plan for the drone along the route.
- **Parameters**
  * `startPosition` – the starting position of the drone
  * `sensorGraph` – the graph containing all of the sensors and distances
- **Returns** – a list of Moves representing the flight plan

• **cutCorner**

```
private uk.ac.ed.inf.aqmaps.geometry.Coords cutCorner(uk.ac.ed.
    inf.aqmaps.geometry.Coords currPos,uk.ac.ed.inf.aqmaps.
    geometry.Coords target,uk.ac.ed.inf.aqmaps.geometry.Coords
    nextTarget)
```

- **Description**
  Attempts to cut the corner by moving from the target a distance of WaypointNavigation#SENSOR_RANGE * `CORNER_CUT_RADIUS_FRACTION` in a direction between the current position and the next target. It tries 3 directions: the bisector between the directions, and the 2 recursive bisectors (3 equally spaced directions). Picks the target which minimises the new path that goes through the new point, and does not collide with an obstacle. If none of the new targets are an improvement, outputs the original target.
- **Parameters**
  * `currPos` – the position of the drone before it will move towards the target
  * `target` – the location of the target
  * `nextTarget` – the location of the target after the current target
- **Returns** – a new target, which has potentially been moved in order to cut the corner

2.6 Class SensorGraph

A graph of the sensors and their distances to each other, taking into account obstacle evasion.

2.6.1 Declaration

```
public class SensorGraph
 extends org.jgrapht.graph.SimpleWeightedGraph
```

2.6.2 Constructor summary

SensorGraph(Coords, Collection, Obstacles) Private Constructor

2.6.3 Method summary

createWithStartLocation(Coords, Collection, Obstacles) Creates a complete weighted graph with the points of all of the sensors and the starting position.

2.6.4 Constructors

- **SensorGraph**

```
private SensorGraph(uk.ac.ed.inf.aqmaps.geometry.Coords
    startPosition,java.util.Collection sensorCoords,uk.ac.ed.inf.
    aqmaps.noflyzone.Obstacles obstacles)
```

  - **Description**
    Private Constructor
  - **Parameters**
    * `startPosition` – the starting position of the drone
    * `sensorCoords` – a Collection of the Coords of the sensors to be visited
    * `obstacles` – the Obstacles that need to be avoided

2.6.5 Methods

- **createWithStartLocation**

```
public static SensorGraph createWithStartLocation(uk.ac.ed.inf.
    aqmaps.geometry.Coords startPosition,java.util.Collection
    sensorCoords,uk.ac.ed.inf.aqmaps.noflyzone.Obstacles
    obstacles)
```

  - **Description**
    Creates a complete weighted graph with the points of all of the sensors and the starting position. The edge weights are the shortest distance between the points, avoiding obstacles if necessary.
  - **Parameters**
    * `startPosition` – the starting position of the drone
    * `sensorCoords` – a Collection of the Coords of the sensors to be visited
    * `obstacles` – the Obstacles that need to be avoided
  - **Returns** – a SensorGraph

2.6.6 Members inherited from class SimpleWeightedGraph

`org.jgrapht.graph.SimpleWeightedGraph`

- `public static GraphBuilder` **createBuilder**`(java.lang.Class` **arg0**`)`
- `public static GraphBuilder` **createBuilder**`(java.util.function.Supplier` **arg0**`)`
- `private static final` **serialVersionUID**

2.6.7 Members inherited from class SimpleGraph

`org.jgrapht.graph.SimpleGraph`

- `public static GraphBuilder` **createBuilder**`(java.lang.Class` **arg0**`)`
- `public static GraphBuilder` **createBuilder**`(java.util.function.Supplier` **arg0**`)`
- `private static final` **serialVersionUID**

2.6.8 Members inherited from class AbstractBaseGraph

`org.jgrapht.graph.AbstractBaseGraph`

- `public Object` **addEdge**`(java.lang.Object` **arg0**`, java.lang.Object` **arg1**`)`
- `public boolean` **addEdge**`(java.lang.Object` **arg0**`, java.lang.Object` **arg1**`, java.lang.Object` **arg2**`)`
- `public Object` **addVertex**`()`
- `public boolean` **addVertex**`(java.lang.Object` **arg0**`)`
- `public Object` **clone**`()`
- `public boolean` **containsEdge**`(java.lang.Object` **arg0**`)`
- `public boolean` **containsVertex**`(java.lang.Object` **arg0**`)`
- `public int` **degreeOf**`(java.lang.Object` **arg0**`)`
- `public Set` **edgeSet**`()`
- `public Set` **edgesOf**`(java.lang.Object` **arg0**`)`
- `private` **edgeSupplier**
- `public Set` **getAllEdges**`(java.lang.Object` **arg0**`, java.lang.Object` **arg1**`)`
- `public Object` **getEdge**`(java.lang.Object` **arg0**`, java.lang.Object` **arg1**`)`
- `public Object` **getEdgeSource**`(java.lang.Object` **arg0**`)`
- `public Supplier` **getEdgeSupplier**`()`
- `public Object` **getEdgeTarget**`(java.lang.Object` **arg0**`)`
- `public double` **getEdgeWeight**`(java.lang.Object` **arg0**`)`
- `public GraphType` **getType**`()`
- `public Supplier` **getVertexSupplier**`()`
- `private static final` **GRAPH_SPECIFICS_MUST_NOT_BE_NULL**
- `private static final` **GRAPH_SPECIFICS_STRATEGY_REQUIRED**
- `private` **graphSpecificsStrategy**
- `public Set` **incomingEdgesOf**`(java.lang.Object` **arg0**`)`
- `public int` **inDegreeOf**`(java.lang.Object` **arg0**`)`
- `private` **intrusiveEdgesSpecifics**
- `private static final` **INVALID_VERTEX_SUPPLIER_DOES_NOT_RETURN_UNIQUE_VERTICI**
- `private static final` **LOOPS_NOT_ALLOWED**
- `private static final` **MIXED_GRAPH_NOT_SUPPORTED**
- `public int` **outDegreeOf**`(java.lang.Object` **arg0**`)`
- `public Set` **outgoingEdgesOf**`(java.lang.Object` **arg0**`)`
- `public boolean` **removeEdge**`(java.lang.Object` **arg0**`)`
- `public Object` **removeEdge**`(java.lang.Object` **arg0**`, java.lang.Object` **arg1**`)`
- `public boolean` **removeVertex**`(java.lang.Object` **arg0**`)`
- `private static final` **serialVersionUID**
- `public void` **setEdgeSupplier**`(java.util.function.Supplier` **arg0**`)`
- `public void` **setEdgeWeight**`(java.lang.Object` **arg0**`, double` **arg1**`)`
- `public void` **setVertexSupplier**`(java.util.function.Supplier` **arg0**`)`
- `private` **specifics**
- `private static final` **THE_GRAPH_CONTAINS_NO_EDGE_SUPPLIER**
- `private static final` **THE_GRAPH_CONTAINS_NO_VERTEX_SUPPLIER**
- `private` **type**
- `private transient` **unmodifiableVertexSet**
- `public Set` **vertexSet**`()`
- `private` **vertexSupplier**

2.6.9 Members inherited from class AbstractGraph

`org.jgrapht.graph.AbstractGraph`

- `protected boolean` **assertVertexExist**`(java.lang.Object` **arg0**`)`
- `public boolean` **containsEdge**`(java.lang.Object` **arg0**`, java.lang.Object` **arg1**`)`
- `public boolean` **equals**`(java.lang.Object` **arg0**`)`
- `public int` **hashCode**`()`
- `public boolean` **removeAllEdges**`(java.util.Collection` **arg0**`)`
- `protected boolean` **removeAllEdges**`(java.lang.Object[]` **arg0**`)`
- `public Set` **removeAllEdges**`(java.lang.Object` **arg0**`, java.lang.Object` **arg1**`)`
- `public boolean` **removeAllVertices**`(java.util.Collection` **arg0**`)`
- `public String` **toString**`()`
- `protected String` **toStringFromSets**`(java.util.Collection` **arg0**`, java.util.Collection` **arg1**`, boolean` **arg2**`)`

2.7 Class WaypointNavigation

A class which handles the navigation of the drone along a series of waypoints to a target. This is the core of the drone control algorithm that plans the movement of the drone itself including all rules about move lengths and directions.

2.7.1 Declaration

```
public class WaypointNavigation
 extends java.lang.Object
```

2.7.2 Field summary

countIterations Count the number if times that navigateToLocation is called
END_POSITION_RANGE The drone must be within this distance of the end position at the end of the flight
MOVE_LENGTH The distance the drone travels in one move.
obstacles
OFFSETS The offsets from the direct direction to try when looking for a direction to move in.
SENSOR_RANGE The drone must be within this distance of a sensor to be able to read it.
targetLocation The target location, such as a sensor or end position
targetSensorW3W If the target is a sensor holds its W3W location, otherwise null
visitedSet Keeps track of all of the points we have visited so far so we can avoid looping back on ourselves.
waypoints A list of Coords waypoints for the drone to follow on its way to the target.

2.7.3 Constructor summary

WaypointNavigation(Obstacles)

2.7.4 Method summary

inRangeOfTarget(Coords) Determines whether or not the position is in range of the target.
navigateAlongWaypoints(Coords, int, int) Recursively finds moves which navigate from waypoint to waypoint, until the target location is reached.
navigateToLocation(Coords, List, W3W) Find a sequence of moves that navigates the drone from the current location along the waypoints to the target.
predictMaxMoveLength(Coords, Coords) Predict the maximum number of moves to reach the specified waypoint.

2.7.5 Fields

- `public static final double` **MOVE_LENGTH**
  - The distance the drone travels in one move.
- `public static final double` **SENSOR_RANGE**
  - The drone must be within this distance of a sensor to be able to read it.
- `public static final double` **END_POSITION_RANGE**
  - The drone must be within this distance of the end position at the end of the flight
- `private final int[]` **OFFSETS**
  - The offsets from the direct direction to try when looking for a direction to move in. Starts at 0 and works outwards. This is easier and faster to hardcode than to generate everytime we run navigation.
- `private final uk.ac.ed.inf.aqmaps.noflyzone.Obstacles` **obstacles**
- `private final java.util.Set` **visitedSet**
  - Keeps track of all of the points we have visited so far so we can avoid looping back on ourselves.
- `private java.util.List` **waypoints**
  - A list of Coords waypoints for the drone to follow on its way to the target.
- `private uk.ac.ed.inf.aqmaps.geometry.Coords` **targetLocation**
  - The target location, such as a sensor or end position
- `private uk.ac.ed.inf.aqmaps.W3W` **targetSensorW3W**
  - If the target is a sensor holds its W3W location, otherwise null
- `private int` **countIterations**
  - Count the number if times that navigateToLocation is called

2.7.6 Constructors

- **WaypointNavigation**

  ```
  public WaypointNavigation(uk.ac.ed.inf.aqmaps.noflyzone.
      Obstacles obstacles)
  ```

  - **Parameters**
    - `obstacles` – the obstacles for collision checking

2.7.7 Methods

- **inRangeOfTarget**

  ```
  private boolean inRangeOfTarget(uk.ac.ed.inf.aqmaps.geometry.
      Coords position)
  ```

  - **Description**
    Determines whether or not the position is in range of the target. The range that determines this depends on whether the target is a sensor or the start/end position of the drone.
  - **Parameters**
    - `position` – the position
  - **Returns** – true if it is in range of the target sensor or end position, false otherwise

- **navigateAlongWaypoints**

  ```
  private java.util.List navigateAlongWaypoints(uk.ac.ed.inf.
      aqmaps.geometry.Coords currentPosition ,int currWaypoint ,int
      movesTilTimeout)
  ```

  - **Description**
    Recursively finds moves which navigate from waypoint to waypoint, until the target location is reached.
  - **Parameters**
    - `currentPosition` – the current position of the drone
    - `currWaypoint` – the current waypoint number
    - `movesTilTimeout` – the maximum number of moves to the next waypoint until the current branch of the search is ended
  - **Returns** – a list of moves which take the drone from the current location to the target

- **navigateToLocation**

```
public java.util.List navigateToLocation(uk.ac.ed.inf.aqmaps.
    geometry.Coords startingPosition,java.util.List waypoints,uk.
    ac.ed.inf.aqmaps.W3W targetSensorW3W)
```

- **Description**

  Find a sequence of moves that navigates the drone from the current location along the waypoints to the target.
- **Parameters**
  * `startingPosition` – the starting position of the drone
  * `waypoints` – a list of Coords waypoints for the drone to follow on its way to the target.
  * `targetSensorW3W` – the W3W of the target sensor, or null if the target is not a sensor.
- **Returns** – a list of Moves that navigate the drone from the starting position to in range of the target

• **predictMaxMoveLength**

```
private int predictMaxMoveLength(uk.ac.ed.inf.aqmaps.geometry.
    Coords startPos,uk.ac.ed.inf.aqmaps.geometry.Coords target)
```

- **Description**

  Predict the maximum number of moves to reach the specified waypoint. The formula is ceiling(distance / MOVE_LENGTH) + 2.
- **Parameters**
  * `startPos` – the starting position of the move
  * `target` – the target waypoint of the move
- **Returns** – the estimated maximum number of moves that it will take to reach the target

Chapter 3

Package uk.ac.ed.inf.aqmaps.geometry

Package Contents *Page*

3.1 Class Coords

Holds a longitude and latitude pair, using a Point2D. This class uses euclidean geometry and its calculations do **not** match with real life.

3.1.1 Declaration

```
public class Coords
 extends java.awt.geom.Point2D.Double
```

3.1.2 Constructor summary

Coords(double, double)

3.1.3 Method summary

bisectorDirection(Coords, Coords) Let this point be P.
buildFromGeojsonPoint(Point) Convert a mapbox point into a Coords
directionTo(Coords) Let this point be P.
getPointOnBisector(Coords, Coords, double) Let this point be P.

getPositionAfterMoveDegrees(double, double) Creates a new Coords which is the result of moving from the current location at the specified angle for the specified length.
getPositionAfterMoveRadians(double, double) Creates a new Coords which is the result of moving from the current location at the specified angle for the specified length.
roundedDirection10Degrees(Coords, int) Let this point be P.
toString() Used for printing Coords for debugging

3.1.4 Constructors

- **Coords**

  ```
  public Coords(double lng, double lat)
  ```

  - **Parameters**
    - `lng` – longitude
    - `lat` – latitude

3.1.5 Methods

- **bisectorDirection**

  ```
  public double bisectorDirection(Coords A, Coords B)
  ```

  - **Description**
    Let this point be P. Calculate the direction of the acute bisector between the lines PA and PB.
  - **Parameters**
    - `A` – point A
    - `B` – point B
  - **Returns** – the direction of the bisector in radians

- **buildFromGeojsonPoint**

  ```
  public static Coords buildFromGeojsonPoint(com.mapbox.geojson.
      Point p)
  ```

  - **Description**
    Convert a mapbox point into a Coords
  - **Parameters**
    - `p` – the mapbox Point
  - **Returns** – an equivalent Coords

- **directionTo**

  ```
  public double directionTo(Coords A)
  ```

  - **Description**
    Let this point be P. Calculates the direction or angle of the line PA with respect to the horizontal, where east is 0, north is pi/2, south is -pi/2, west is pi
  - **Parameters**
    * `A` – point A
  - **Returns** – the direction in radians

- **getPointOnBisector**

  ```
  public Coords getPointOnBisector(Coords A,Coords B,double
      distance)
  ```

  - **Description**
    Let this point be P. Creates a point a specified distance away in the direction of the acute bisector between the lines PA and PB.
  - **Parameters**
    * `A` – point A
    * `B` – point B
    * `distance` – the distance the new point should be away from this point
  - **Returns** – the new point

- **getPositionAfterMoveDegrees**

  ```
  public Coords getPositionAfterMoveDegrees(double degrees ,double
      length)
  ```

  - **Description**
    Creates a new Coords which is the result of moving from the current location at the specified angle for the specified length. Angle in degrees version.
  - **Parameters**
    * `degrees` – the direction of the move as an angle in degrees
    * `length` – the length of the move
  - **Returns** – a Coords containing the calculated point

- **getPositionAfterMoveRadians**

  ```
  public Coords getPositionAfterMoveRadians(double radians ,double
      length)
  ```

- **Description**
  Creates a new Coords which is the result of moving from the current location at the specified angle for the specified length. Angle in radians version.
- **Parameters**
  * `radians` – the direction of the move as an angle in radians
  * `length` – the length of the move
- **Returns** – a Coords containing the calculated point

- **roundedDirection10Degrees**

```
public int roundedDirection10Degrees(Coords A, int offset)
```

- **Description**
  Let this point be P. Calculates the direction of the line PA, rounded to the nearest 10 degrees, offset by an amount, and expressed in the range [0,350].
- **Parameters**
  * `A` – point A
  * `offset` – the offset
- **Returns** – the direction in degrees

- **toString**

```
public java.lang.String toString()
```

- **Description**
  Used for printing Coords for debugging

3.1.6 Members inherited from class Point2D.Double

`java.awt.geom.Point2D.Double`

- `public double` **getX**`()`
- `public double` **getY**`()`
- `private static final` **serialVersionUID**
- `public void` **setLocation**`(double` **arg0**`, double` **arg1**`)`
- `public String` **toString**`()`
- `public` **x**
- `public` **y**

3.1.7 Members inherited from class Point2D

`java.awt.geom.Point2D`

- `public Object` **clone**`()`
- `public double` **distance**`(double` **arg0**`, double` **arg1**`)`
- `public static double` **distance**`(double` **arg0**`, double` **arg1**`, double` **arg2**`, double` **arg3**`)`
- `public double` **distance**`(Point2D` **arg0**`)`
- `public double` **distanceSq**`(double` **arg0**`, double` **arg1**`)`
- `public static double` **distanceSq**`(double` **arg0**`, double` **arg1**`, double` **arg2**`, double` **arg3**`)`
- `public double` **distanceSq**`(Point2D` **arg0**`)`
- `public boolean` **equals**`(java.lang.Object` **arg0**`)`
- `public abstract double` **getX**`()`
- `public abstract double` **getY**`()`
- `public int` **hashCode**`()`
- `public abstract void` **setLocation**`(double` **arg0**`, double` **arg1**`)`
- `public void` **setLocation**`(Point2D` **arg0**`)`

3.2 Class Polygon

Holds a polygon as a list of the Coords that make up the vertices, in order

3.2.1 Declaration

```
public class Polygon
 extends java.lang.Object
```

3.2.2 Field summary

boundingBox A rectangular bounding box which contains the polygon
OUTLINE_MARGIN The margin to use when generating a polygon which outlines another, see `generateOutlinePoints()`
path Holds a path around the polygon and is used for checking inside-ness and generating bounding boxes.
points A list of the vertices of the polygon
segments All of the line segments that make up the edges

3.2.3 Constructor summary

Polygon(List) Initialize a Polygon from a list of Coords points

3.2.4 Method summary

buildFromFeature(Feature) Create a Polygon from a GeoJSON Polygon
contains(Coords) Checks whether a given point is containing within this polygon.
createBoundingBox() Creates a bounding box that contains the rectangular bounds of the polygon.
createSegments() Creates a list of the segments between adjacent points in the polygon.
generateOutlinePoints() Generates the points of a new polygon which contains the original by a very small margin.
generatePath2D() Creates a Path2D of the points in this polygon, so we can use its getBounds2D() and contains() methods.
getPoints()
getSegments() This method is currently only used in a test, but it is kept to test whether the segments have been created properly.
lineCollision(Coords, Coords) Determines whether the line segment between the start and end points collides with the polygon.

3.2.5 Fields

- `public static final double` **OUTLINE_MARGIN**
  - The margin to use when generating a polygon which outlines another, see `generateOutlinePoints()`
- `private final java.util.List` **points**
  - A list of the vertices of the polygon
- `private final java.awt.geom.Path2D` **path**
  - Holds a path around the polygon and is used for checking inside-ness and generating bounding boxes.
- `private final java.util.List` **segments**
  - All of the line segments that make up the edges
- `private final java.awt.geom.Rectangle2D` **boundingBox**
  - A rectangular bounding box which contains the polygon

3.2.6 Constructors

- **Polygon**

```
private Polygon(java.util.List points)
```

  - **Description**
    Initialize a Polygon from a list of Coords points
  - **Parameters**
    - `points` – a list of Coords

3.2.7 Methods

- **buildFromFeature**

  ```
  public static Polygon buildFromFeature(com.mapbox.geojson.
      Feature feature)
  ```

  - **Description**
    Create a Polygon from a GeoJSON Polygon
  - **Parameters**
    * `feature` – a GeoJSON Feature containing a Polygon
  - **Returns** – the converted Polygon

- **contains**

  ```
  public boolean contains(Coords p)
  ```

  - **Description**
    Checks whether a given point is containing within this polygon.
  - **Parameters**
    * `p` – the point
  - **Returns** – true if the polygon contains the point, false otherwise

- **createBoundingBox**

  ```
  private java.awt.geom.Rectangle2D createBoundingBox()
  ```

  - **Description**
    Creates a bounding box that contains the rectangular bounds of the polygon.
  - **Returns** – the bounding box as a Rectangle2D

- **createSegments**

  ```
  private java.util.List createSegments()
  ```

  - **Description**
    Creates a list of the segments between adjacent points in the polygon.
  - **Returns** – a list of Segments as Line2D

- **generateOutlinePoints**

  ```
  public java.util.List generateOutlinePoints()
  ```

  - **Description**
    Generates the points of a new polygon which contains the original by a very small margin. It generates points a distance of 1.0e-14 from each point in the original Polygon in the direction of the bisecting angle between the two adjacent sides, or the opposite direction if that point is inside the polygon. The resulting polygon will be larger than the original by a margin of 1.0e-14 on all sides.
  - **Returns** – the outlining Polygon

- **generatePath2D**

  ```
  private java.awt.geom.Path2D generatePath2D()
  ```

  - **Description**
    Creates a Path2D of the points in this polygon, so we can use its getBounds2D() and contains() methods.
  - **Returns** – a Path2D containing the vertices of the polygon

- **getPoints**

  ```
  public java.util.List getPoints()
  ```

  - **Returns** – the points which make up the vertices of this polygon

- **getSegments**

  ```
  public java.util.List getSegments()
  ```

  - **Description**
    This method is currently only used in a test, but it is kept to test whether the segments have been created properly.
  - **Returns** – the segments which make up the edges of the polygon

- **lineCollision**

  ```
  public boolean lineCollision(Coords start ,Coords end)
  ```

  - **Description**
    Determines whether the line segment between the start and end points collides with the polygon.
  - **Parameters**
    * `start` – the coordinates of the start point
    * `end` – the coordinates of the end point
  - **Returns** – true if the segment collides with an obstacle, false otherwise

Chapter 4

Package uk.ac.ed.inf.aqmaps.io

Package Contents *Page*

4.1 Interface InputController

Handles interaction with input from all remote information sources and devices. Gets information on sensor locations, no-fly zones, W3W locations, and sensor readings. Implementations of the interface can gather the information from any source, such as a simple web server for testing purposes, or from a full system where data is also read from real sensors.

4.1.1 Declaration

```
public interface InputController
```

4.1.2 All known subinterfaces

ServerInputController (in 4.5, page 42)

4.1.3 All classes known to implement interface

ServerInputController (in 4.5, page 42)

4.1.4 Method summary

getNoFlyZones() Gets information about no-fly zones from a remote source
getSensorW3Ws() Gets the list of sensors that need to be visited from a remote source
readSensor(W3W) Reads information from the sensor at the provided W3W location

4.1.5 Methods

- **getNoFlyZones**

```
java.util.List getNoFlyZones()
```

  - **Description**
    Gets information about no-fly zones from a remote source
  - **Returns** – a FeatureCollection containing the locations of the no-fly zones

- **getSensorW3Ws**

```
java.util.List getSensorW3Ws()
```

  - **Description**
    Gets the list of sensors that need to be visited from a remote source
  - **Returns** – a list of W3W locations of the sensors

- **readSensor**

```
uk.ac.ed.inf.aqmaps.Sensor readSensor(uk.ac.ed.inf.aqmaps.W3W
    location)
```

  - **Description**
    Reads information from the sensor at the provided W3W location
  - **Parameters**
    * `location` – the location of the sensor as a W3W class
  - **Returns** – a Sensor object representing the current status of the sensor

4.2 Interface OutputController

Handles interaction with all output locations. Implementations may output to any source, such as to a file or to a server.

4.2.1 Declaration

```
public interface OutputController
```

4.2.2 All known subinterfaces

FileOutputController (in 4.4, page 41)

4.2.3 All classes known to implement interface

FileOutputController (in 4.4, page 41)

4.2.4 Method summary

outputFlightpath(String) Outputs the flightpath planned by the drone
outputMapGeoJSON(String) Outputs the GeoJSON map containing the flightpath and the sensor readings collected by the drone

4.2.5 Methods

- **outputFlightpath**

  ```
  void outputFlightpath(java.lang.String flightpathText)
  ```

  - **Description**
    Outputs the flightpath planned by the drone
  - **Parameters**
    - `flightpathText` – the String containing the flightpath data

- **outputMapGeoJSON**

  ```
  void outputMapGeoJSON(java.lang.String json)
  ```

  - **Description**
    Outputs the GeoJSON map containing the flightpath and the sensor readings collected by the drone
  - **Parameters**
    - `json` – the GeoJSON String

4.3 Interface Server

Handles requesting data from a server.

4.3.1 Declaration

```
public interface Server
```

4.3.2 All known subinterfaces

WebServer (in 4.6, page 44)

4.3.3 All classes known to implement interface

WebServer (in 4.6, page 44)

4.3.4 Method summary

requestData(String) Request the data that is located at the given URL.

4.3.5 Methods

- **requestData**

  ```
  java.lang.String requestData(java.lang.String url)
  ```

  - **Description**
    Request the data that is located at the given URL. Will cause a fatal error if it cannot connect to the server, or if the requested file is not found.
  - **Parameters**
    * `url` – the URL of the file to request
  - **Returns** – the requested data as a String

4.4 Class FileOutputController

Outputs to the current directory of the filesystem

4.4.1 Declaration

```
public class FileOutputController
 extends java.lang.Object implements OutputController
```

4.4.2 Field summary

settings

4.4.3 Constructor summary

FileOutputController(Settings)

4.4.4 Method summary

outputFlightpath(String)
outputMapGeoJSON(String)

4.4.5 Fields

- `private final uk.ac.ed.inf.aqmaps.Settings` **settings**

4.4.6 Constructors

- **FileOutputController**

```
public FileOutputController(uk.ac.ed.inf.aqmaps.Settings
    settings)
```

  - **Parameters**
    - `settings` – a Settings holding the current input arguments

4.4.7 Methods

- **outputFlightpath**

```
void outputFlightpath(java.lang.String flightpathText)
```

  - **Description copied from OutputController (in 4.2, page 39)**
    Outputs the flightpath planned by the drone
  - **Parameters**
    - `flightpathText` – the String containing the flightpath data

- **outputMapGeoJSON**

```
void outputMapGeoJSON(java.lang.String json)
```

  - **Description copied from OutputController (in 4.2, page 39)**
    Outputs the GeoJSON map containing the flightpath and the sensor readings collected by the drone
  - **Parameters**
    - `json` – the GeoJSON String

4.5 Class ServerInputController

Implements the Remote interface using a connection to a simple web server.

4.5.1 Declaration

```
public class ServerInputController
 extends java.lang.Object implements InputController
```

4.5.2 Field summary

noFlyZones
sensorMap
sensorW3Ws Note: we need the sensor W3W as a list instead of using keySet() on the map since the set has an undetermined ordering (specifically, not determined by the random seed), which changes the initial random tours generated by the 2-opt algorithm and produces different results with the same random seed.
server Represents the server which we will get data from
serverUrl The base URL of the server, such as http://localhost:80

4.5.3 Constructor summary

ServerInputController(Server, int, int, int, int) Create a new ServerInputController instance with the given Server, date, and port number
ServerInputController(Settings) Create a new ServerInputController instance with the given settings

4.5.4 Method summary

convertToSensor(SensorDeserializer) Converts a SensorDeserializer to a Sensor by getting the coordinates of its W3W location from the server.
getNoFlyZones()
getSensorW3Ws()
loadData(int, int, int) Loads the no-fly zones and sensor data from the server into the class fields.
readSensor(W3W)

4.5.5 Fields

- `private final java.util.HashMap` **sensorMap**
- `private final java.util.List` **sensorW3Ws**
  - Note: we need the sensor W3W as a list instead of using keySet() on the map since the set has an undetermined ordering (specifically, not determined by the random seed), which changes the initial random tours generated by the 2-opt algorithm and produces different results with the same random seed.
- `private final Server` **server**
  - Represents the server which we will get data from
- `private final java.lang.String` **serverUrl**

  - The base URL of the server, such as http://localhost:80

- `private java.util.List` **noFlyZones**

4.5.6 Constructors

- **ServerInputController**

```
public ServerInputController(Server server,int day,int month,int
    year,int port)
```

  - **Description**
    Create a new ServerInputController instance with the given Server, date, and port number
  - **Parameters**
    - `server` – a Server
    - `day` – the day
    - `month` – the month
    - `year` – the year
    - `port` – the port of the server

- **ServerInputController**

```
public ServerInputController(uk.ac.ed.inf.aqmaps.Settings
   settings)
```

  - **Description**
    Create a new ServerInputController instance with the given settings
  - **Parameters**
    - `settings` – the Settings object containing the current settings

4.5.7 Methods

- **convertToSensor**

```
private uk.ac.ed.inf.aqmaps.Sensor convertToSensor(uk.ac.ed.inf.
   aqmaps.deserializers.SensorDeserializer sensorDeserializer)
```

  - **Description**
    Converts a SensorDeserializer to a Sensor by getting the coordinates of its W3W location from the server.
  - **Parameters**
    - `sensorDeserializer` – the sensorDeserializer to convert

  - **Returns** – an equivalent Sensor containing its coordinates

- **getNoFlyZones**

```
java.util.List getNoFlyZones()
```

  - **Description copied from InputController (in 4.1, page 38)**
    Gets information about no-fly zones from a remote source
  - **Returns** – a FeatureCollection containing the locations of the no-fly zones

- **getSensorW3Ws**

```
java.util.List getSensorW3Ws()
```

  - **Description copied from InputController (in 4.1, page 38)**
    Gets the list of sensors that need to be visited from a remote source
  - **Returns** – a list of W3W locations of the sensors

- **loadData**

```
private void loadData(int day,int month,int year)
```

  - **Description**
    Loads the no-fly zones and sensor data from the server into the class fields.
  - **Parameters**
    * `day` – the day
    * `month` – the month
    * `year` – the year

- **readSensor**

```
uk.ac.ed.inf.aqmaps.Sensor readSensor(uk.ac.ed.inf.aqmaps.W3W
    location)
```

  - **Description copied from InputController (in 4.1, page 38)**
    Reads information from the sensor at the provided W3W location
  - **Parameters**
    * `location` – the location of the sensor as a W3W class
  - **Returns** – a Sensor object representing the current status of the sensor

4.6 Class WebServer

Handles requesting data from an HTTP server.

4.6.1 Declaration

```
public class WebServer
 extends java.lang.Object implements Server
```

4.6.2 Field summary

client

4.6.3 Constructor summary

WebServer()

4.6.4 Method summary

requestData(String)

4.6.5 Fields

- `private final java.net.http.HttpClient` **client**

4.6.6 Constructors

- **WebServer**

  ```
  public WebServer()
  ```

4.6.7 Methods

- **requestData**

  ```
  java.lang.String requestData(java.lang.String url)
  ```

  – **Description copied from Server (in 4.3, page 40)**
    Request the data that is located at the given URL. Will cause a fatal error if it cannot connect to the server, or if the requested file is not found.
  – **Parameters**
    * `url` – the URL of the file to request
  – **Returns** – the requested data as a String

Chapter 5

Package uk.ac.ed.inf.aqmaps.deserializers

Package Contents *Page*

5.1 Class CoordsDeserializer

Used for deserialization of Coords, this is needed since the field names in Coords inherit from Point2D so @SerializedName can't be used to rename lng and lat to x and y

5.1.1 Declaration

```
public class CoordsDeserializer
 extends java.lang.Object
```

5.1.2 Field summary

lat
lng

5.1.3 Constructor summary

CoordsDeserializer()

5.1.4 Method summary

getCoords()

5.1.5 Fields

- `private double` **lng**
- `private double` **lat**

5.1.6 Constructors

- **CoordsDeserializer**

```
public CoordsDeserializer()
```

5.1.7 Methods

- **getCoords**

```
public uk.ac.ed.inf.aqmaps.geometry.Coords getCoords()
```

  - **Returns** – a Coords object

5.2 Class SensorDeserializer

Used only for deserializing the sensor information from JSON. The real Sensor class stores the location as W3W instead of a String.

5.2.1 Declaration

```
public class SensorDeserializer
 extends java.lang.Object
```

5.2.2 Field summary

battery
location
reading

5.2.3 Constructor summary

SensorDeserializer()

5.2.4 Method summary

getBattery()
getLocation()
getReading()

5.2.5 Fields

- `private java.lang.String` **location**
- `private float` **battery**
- `private java.lang.String` **reading**

5.2.6 Constructors

- **SensorDeserializer**

```
public SensorDeserializer()
```

5.2.7 Methods

- **getBattery**

```
public float getBattery()
```

  - **Returns** – the battery level of this sensor as a percentage

- **getLocation**

```
public java.lang.String getLocation()
```

  - **Returns** – the location of the sensor as a W3W string

- **getReading**

```
public java.lang.String getReading()
```

  - **Returns** – the reading of the sensor, as a String

5.3 Class W3WDeserializer

Used for deserialization of W3W, `CoordsDeserializer` (in 5.1, page 46) for why this is necessary.

5.3.1 Declaration

```
public class W3WDeserializer
 extends java.lang.Object
```

5.3.2 Field summary

coordinates
words

5.3.3 Constructor summary

W3WDeserializer()

5.3.4 Method summary

getW3W()

5.3.5 Fields

- `private CoordsDeserializer` **coordinates**
- `private java.lang.String` **words**

5.3.6 Constructors

- **W3WDeserializer**

  ```
  public W3WDeserializer()
  ```

5.3.7 Methods

- **getW3W**

  ```
  public uk.ac.ed.inf.aqmaps.W3W getW3W()
  ```

  – **Returns** – a W3W object

Chapter 6

Package uk.ac.ed.inf.aqmaps.noflyzone

6.1 Class ObstacleGraph

A graph of the vertices of the obstacles and edges between them if they have line of sight.

6.1.1 Declaration

```
public class ObstacleGraph
 extends org.jgrapht.graph.SimpleWeightedGraph
```

6.1.2 Constructor summary

ObstacleGraph(Obstacles, List) Constructor

6.1.3 Method summary

prepareGraph(List, Obstacles) Prepare a weighted graph containing all points which form an outline around the polygons as vertices, and edges connecting them if they have line of sight, which have a weight equal to the distance between them.

6.1.4 Constructors

- **ObstacleGraph**

  ```
  private ObstacleGraph(Obstacles obstacles ,java.util.List
      outlinePoints)
  ```

  - **Description**
    Constructor
  - **Parameters**
    - `outlinePoints` – the points which form the outline of the obstacle polygons
    - `obstacles` – the obstacles to prepare the graph for

6.1.5 Methods

- **prepareGraph**

  ```
  public static ObstacleGraph prepareGraph(java.util.List
      outlinePoints ,Obstacles obstacles)
  ```

  - **Description**
    Prepare a weighted graph containing all points which form an outline around the polygons as vertices, and edges connecting them if they have line of sight, which have a weight equal to the distance between them. The graph uses outline polygons since if it used the original polygons, their points would occupy the same location and any line emerging from the corner of an obstacle would be considered to be colliding with it. See see `generateOutlinePoints()` .
  - **Parameters**
    - `outlinePoints` – the points which form the outline of the obstacle polygons
    - `obstacles` – the obstacles to prepare the graph for
  - **Returns** – a graph representation of the obstacles

6.1.6 Members inherited from class SimpleWeightedGraph

`org.jgrapht.graph.SimpleWeightedGraph`

- `public static GraphBuilder` **createBuilder**`(java.lang.Class` **arg0**`)`
- `public static GraphBuilder` **createBuilder**`(java.util.function.Supplier` **arg0**`)`
- `private static final` **serialVersionUID**

6.1.7 Members inherited from class SimpleGraph

`org.jgrapht.graph.SimpleGraph`

- `public static GraphBuilder` **createBuilder**`(java.lang.Class` **arg0**`)`
- `public static GraphBuilder` **createBuilder**`(java.util.function.Supplier` **arg0**`)`
- `private static final` **serialVersionUID**

6.1.8 Members inherited from class AbstractBaseGraph

`org.jgrapht.graph.AbstractBaseGraph`

- `public Object` **addEdge**`(java.lang.Object` **arg0**`, java.lang.Object` **arg1**`)`
- `public boolean` **addEdge**`(java.lang.Object` **arg0**`, java.lang.Object` **arg1**`, java.lang.Object` **arg2**`)`
- `public Object` **addVertex**`()`
- `public boolean` **addVertex**`(java.lang.Object` **arg0**`)`
- `public Object` **clone**`()`
- `public boolean` **containsEdge**`(java.lang.Object` **arg0**`)`
- `public boolean` **containsVertex**`(java.lang.Object` **arg0**`)`
- `public int` **degreeOf**`(java.lang.Object` **arg0**`)`
- `public Set` **edgeSet**`()`
- `public Set` **edgesOf**`(java.lang.Object` **arg0**`)`
- `private` **edgeSupplier**
- `public Set` **getAllEdges**`(java.lang.Object` **arg0**`, java.lang.Object` **arg1**`)`
- `public Object` **getEdge**`(java.lang.Object` **arg0**`, java.lang.Object` **arg1**`)`
- `public Object` **getEdgeSource**`(java.lang.Object` **arg0**`)`
- `public Supplier` **getEdgeSupplier**`()`
- `public Object` **getEdgeTarget**`(java.lang.Object` **arg0**`)`
- `public double` **getEdgeWeight**`(java.lang.Object` **arg0**`)`
- `public GraphType` **getType**`()`
- `public Supplier` **getVertexSupplier**`()`
- `private static final` **GRAPH_SPECIFICS_MUST_NOT_BE_NULL**
- `private static final` **GRAPH_SPECIFICS_STRATEGY_REQUIRED**
- `private` **graphSpecificsStrategy**
- `public Set` **incomingEdgesOf**`(java.lang.Object` **arg0**`)`
- `public int` **inDegreeOf**`(java.lang.Object` **arg0**`)`
- `private` **intrusiveEdgesSpecifics**
- `private static final` **INVALID_VERTEX_SUPPLIER_DOES_NOT_RETURN_UNIQUE_VERTICI**
- `private static final` **LOOPS_NOT_ALLOWED**
- `private static final` **MIXED_GRAPH_NOT_SUPPORTED**
- `public int` **outDegreeOf**`(java.lang.Object` **arg0**`)`
- `public Set` **outgoingEdgesOf**`(java.lang.Object` **arg0**`)`
- `public boolean` **removeEdge**`(java.lang.Object` **arg0**`)`
- `public Object` **removeEdge**`(java.lang.Object` **arg0**`, java.lang.Object` **arg1**`)`
- `public boolean` **removeVertex**`(java.lang.Object` **arg0**`)`
- `private static final` **serialVersionUID**
- `public void` **setEdgeSupplier**`(java.util.function.Supplier` **arg0**`)`
- `public void` **setEdgeWeight**`(java.lang.Object` **arg0**`, double` **arg1**`)`
- `public void` **setVertexSupplier**`(java.util.function.Supplier` **arg0**`)`
- `private` **specifics**
- `private static final` **THE_GRAPH_CONTAINS_NO_EDGE_SUPPLIER**
- `private static final` **THE_GRAPH_CONTAINS_NO_VERTEX_SUPPLIER**
- `private` **type**
- `private transient` **unmodifiableVertexSet**
- `public Set` **vertexSet**`()`
- `private` **vertexSupplier**

6.1.9 Members inherited from class AbstractGraph

`org.jgrapht.graph.AbstractGraph`

- `protected boolean` **assertVertexExist**`(java.lang.Object` **arg0**`)`
- `public boolean` **containsEdge**`(java.lang.Object` **arg0**`, java.lang.Object` **arg1**`)`
- `public boolean` **equals**`(java.lang.Object` **arg0**`)`
- `public int` **hashCode**`()`
- `public boolean` **removeAllEdges**`(java.util.Collection` **arg0**`)`
- `protected boolean` **removeAllEdges**`(java.lang.Object[]` **arg0**`)`
- `public Set` **removeAllEdges**`(java.lang.Object` **arg0**`, java.lang.Object` **arg1**`)`
- `public boolean` **removeAllVertices**`(java.util.Collection` **arg0**`)`
- `public String` **toString**`()`
- `protected String` **toStringFromSets**`(java.util.Collection` **arg0**`, java.util.Collection` **arg1**`, boolean` **arg2**`)`

6.2 Class ObstaclePathfinder

Handles obstacle evasion. Uses Obstacles and an ObstacleGraph to find paths between points which do not collides with any obstacles.

6.2.1 Declaration

```
public class ObstaclePathfinder
 extends java.lang.Object
```

6.2.2 Field summary

graph A weighted graph containing all points which form an outline around the polygons as vertices, and edges connecting them if they have line of sight, which have a weight equal to the distance between them.

obstacles

6.2.3 Constructor summary

ObstaclePathfinder(ObstacleGraph, Obstacles) Construct an Obstacle evader with the given graph and obstacles.

6.2.4 Method summary

addEdgeIfHasLineOfSight(Coords, Coords) Adds an edge between points A and B if they have line of sight to each other, or in other words if the line between them does not collide with an obstacle.

getPathBetweenPoints(Coords, Coords) Find the shortest path between the start and end points, navigating around obstacles if necessary.

getShortestPath(Coords, Coords) Computes a GraphPath containing the shortest path from the start to the end coordinates, navigating around obstacles if required.

getShortestPathLength(Coords, Coords) Find the length of the shortest path between the start and end points, navigating around obstacles if necessary.

6.2.5 Fields

- `private final ObstacleGraph` **graph**
  - A weighted graph containing all points which form an outline around the polygons as vertices, and edges connecting them if they have line of sight, which have a weight equal to the distance between them.
- `private final Obstacles` **obstacles**

6.2.6 Constructors

- **ObstaclePathfinder**

```
public ObstaclePathfinder(ObstacleGraph graph,Obstacles
    obstacles)
```

  - **Description**
    Construct an Obstacle evader with the given graph and obstacles.
  - **Parameters**
    - `graph` – a graph of the obstacles
    - `obstacles` – the Obstacles

6.2.7 Methods

- **addEdgeIfHasLineOfSight**

```
private void addEdgeIfHasLineOfSight(uk.ac.ed.inf.aqmaps.
    geometry.Coords A,uk.ac.ed.inf.aqmaps.geometry.Coords B)
```

  - **Description**
    Adds an edge between points A and B if they have line of sight to each other, or in other words if the line between them does not collide with an obstacle. Sets the edge weight to the distance between the points.
  - **Parameters**
    - `A` – point A
    - `B` – point B

- **getPathBetweenPoints**

```
public java.util.List getPathBetweenPoints(uk.ac.ed.inf.aqmaps.
    geometry.Coords start,uk.ac.ed.inf.aqmaps.geometry.Coords end
    )
```

  - **Description**
    Find the shortest path between the start and end points, navigating around obstacles if necessary.
  - **Parameters**
    * `start` – the starting point
    * `end` – the ending point
  - **Returns** – a list of points specifying the route

- **getShortestPath**

```
private org.jgrapht.GraphPath getShortestPath(uk.ac.ed.inf.
    aqmaps.geometry.Coords start,uk.ac.ed.inf.aqmaps.geometry.
    Coords end)
```

  - **Description**
    Computes a GraphPath containing the shortest path from the start to the end coordinates, navigating around obstacles if required.
  - **Parameters**
    * `start` – the start point
    * `end` – the end point
  - **Returns** – a GraphPath

- **getShortestPathLength**

```
public double getShortestPathLength(uk.ac.ed.inf.aqmaps.geometry
    .Coords start,uk.ac.ed.inf.aqmaps.geometry.Coords end)
```

  - **Description**
    Find the length of the shortest path between the start and end points, navigating around obstacles if necessary. Euclidean distance is used as the length measure.
  - **Parameters**
    * `start` – the starting point
    * `end` – the ending point
  - **Returns** – the length of the path in degrees

6.3 Class Obstacles

Holds information about the obstacles or no-fly zones that the drone must avoid.

6.3.1 Declaration

```
public class Obstacles
 extends java.lang.Object
```

6.3.2 Field summary

BOTTOM_RIGHT A Point representing the southeast corner of the confinement area.
graph A weighted graph containing all points which form an outline around the polygons as vertices, and edges connecting them if they have line of sight, which have a weight equal to the distance between them.
polygons A list of the Polygon representations of the obstacles
TOP_LEFT A Point representing the northwest corner of the confinement area.

6.3.3 Constructor summary

Obstacles(List) Constructs Obstacles out of a list of Polygons specifying their locations

6.3.4 Method summary

getObstaclePathfinder() Gets an ObstaclePathfinder using these Obstacles.
isInConfinement(Coords) Determines whether or not a point is inside the confinement area
lineCollision(Coords, Coords) Determines whether the line segment between the start and end points collides with a obstacle.
pointCollides(Coords) Determine whether the given point is inside an obstacle, or outside the confinement area.

6.3.5 Fields

- `public static final uk.ac.ed.inf.aqmaps.geometry.Coords` **TOP_LEFT**
  - A Point representing the northwest corner of the confinement area.
- `public static final uk.ac.ed.inf.aqmaps.geometry.Coords` **BOTTOM_RIGHT**
  - A Point representing the southeast corner of the confinement area.
- `private final ObstacleGraph` **graph**
  - A weighted graph containing all points which form an outline around the polygons as vertices, and edges connecting them if they have line of sight, which have a weight equal to the distance between them.
- `private final java.util.List` **polygons**
  - A list of the Polygon representations of the obstacles

6.3.6 Constructors

- **Obstacles**

  ```
  public Obstacles(java.util.List polygons)
  ```

  - **Description**
    Constructs Obstacles out of a list of Polygons specifying their locations
  - **Parameters**
    * `polygons` – the Polygons which make up the obstacles

6.3.7 Methods

- **getObstaclePathfinder**

  ```
  public ObstaclePathfinder getObstaclePathfinder()
  ```

  - **Description**
    Gets an ObstaclePathfinder using these Obstacles. The ObstaclePathfinder uses a clone of the obstacle graph, allowing it to be used concurrently with other ObstaclePathfinder.
  - **Returns** – an ObstaclePathfinder instance with these obstacles

- **isInConfinement**

  ```
  public boolean isInConfinement(uk.ac.ed.inf.aqmaps.geometry.
      Coords point)
  ```

  - **Description**
    Determines whether or not a point is inside the confinement area
  - **Parameters**
    * `point` – the point to examine
  - **Returns** – true if the point is inside the confinement area, false otherwise

- **lineCollision**

  ```
  public boolean lineCollision(uk.ac.ed.inf.aqmaps.geometry.Coords
      start ,uk.ac.ed.inf.aqmaps.geometry.Coords end)
  ```

  - **Description**
    Determines whether the line segment between the start and end points collides with a obstacle.

- **Parameters**
  - `start` – the coordinates of the start point
  - `end` – the coordinates of the end point
- **Returns** – true if the segment collides with an obstacle, false otherwise

- **pointCollides**

```
public boolean pointCollides(uk.ac.ed.inf.aqmaps.geometry.Coords
    coords)
```

  - **Description**
    Determine whether the given point is inside an obstacle, or outside the confinement area. This is currently only used in testing to generate random starting points.
  - **Parameters**
    - `coords` – the point
  - **Returns** – true if there is a collision, false otherwise