Drone Assignment Report

S1871633

TABLE OF CONTENTS

1. SOFTWARE ARCHITECTURE

1.1 INTRODUCTION TO CLASSES

Below are the classes my application makes use of:
1. App 2. ServerData 3. SensorInfo 4. AirQualityData 5. DroneMovement
6. Zones 7. Words 8. WritingToFiles 9. Output

The sole purpose for coming up with these classes is to make sure that each class does one thing at a time. Since I knew that I would be reading from a server and writing to files, I thought it would be a good idea to have a class that does the reading and another that does the writing with its helper methods. Also, I needed classes to deserialize the data from json file into.
As the data for a particular sensor is gotten from two separate places on the server, I created a class that stores the most important information about a sensor in this project so that it is easy to work with the sensors (i.e., accessing information about a sensor). The output of the drone flight path had to be printed out in a specific format, so I had a class for that purpose. I also created a separate class that deals with the confinement area and no fly zones and lastly, the class that controls the drone movement with its helper methods.

- App class: This class executes my application.
- ServerData class: This class reads the data from the server. It makes three different requests to the server (the three folders) to get the data from each and store the data into a variable.
- SensorInfo class: This class contains all the information about a sensor i.e., the point where the sensor is, the battery level, the air quality reading, the what3word location of the sensor.
- AirQualityData: This is the class I deserialize the air quality data json file for each date into so that I can access the elements in the json file.
- DroneMovement: This class has information about that the drone movement and the helper methods needed to move the drone.
- Zones: This class is about the confinement area and everything that has to do with the no fly zone buildings like checking if a path intersects with a no fly zone.
- Words: This is the class I deserialize the details json file for each what3word into so I can access its elements.
- WritingToFiles: Just as the ServerData class does the reading from the server, this class has methods needed to write the drone flight path into a text file and also write to a geojson file.
- Output: This class has methods that deal with how the flight path taken by the drone is printed out.

2. CLASS DOCUMENTATION

2.1 APP CLASS

2.1.a Field

- savedArgs: a private static field which stores all command line arguments in a String array of length 7. The elements in this array can be accessed other classes via a getter.

2.1.b Method

- getArgs: A getter which returns a String array of the command line arguments saved in the savedArgs field. This is a public method because its elements will be needed by other classes.

2.2 ZONES CLASS

2.2.a Fields

- MIN_LNG: A constant of type double which holds the minimum longitude for the confinement area where the drone moves around.
- MAX_LNG: A constant of type double which holds the maximum longitude for the confinement area where the drone moves around.
- MIN_LAT: A constant of type double which holds the minimum latitude for the confinement area given where the drone moves around.
- MAX_LAT: A constant of type double which holds the maximum latitude for the confinement area where the drone moves around.
- buildings: An instance variable that stores the no fly zones/regions read form the server in a List of polygons.

2.2.b Methods

- isPointInConfinementArea
  Parameters: a point (Point p)
  Return type: boolean
  Function: checks if the point is in the confinement area and returns true if that is the case or false otherwise.
  Access modifier: public, because it is needed in another class.

- onLineSegment
  Parameters: three points (Point p, Point q, Point r)
  Return type: boolean
  Function: checks if point p lies on the line segment pr and returns true if that is the case or false otherwise
  Access modifier: private, a helper method for doLinesInteresect method.

- findOrientation
  Parameters: three points (Point p, Point q, Point r)
  Return type: int
  Function: finds the orientation of the three points and returns number (int) i.e., 0 if the three points are collinear, 1 if they are oriented in a clockwise manner and 2 if in a counterclockwise manner.
  Access modifier: private, a helper method for doLinesInteresect method.

- doLinesIntersect
  Parameters: four points (Point p1, Point q1, Point p2, Point q2)
  Return type: boolean
  Function: returns true if line segment p1 q1 intersects line segment p2q2 and false otherwise.
  Access modifier: private, a helper method for doesLineIntersectsPolygon method.
  Refer to Appendix for how this method works.

- doesLineIntersectsPolygon
  Parameters: two points (Point p, Point q) and a polygon (Polygon polygon)
  Return type: boolean
  Function: checks if the line segment pq intersects any of the lines that forms the perimeter of the polygon, returns true if that is the case and false otherwise. It uses doLinesIntersect method to check for the intersection.
  Access modifier: private, a helper method for doesLineIntersectsBuildings method.

- doesLineIntersectsBuildings
  Parameters: two points (Point p, Point q)
  Return type: boolean
  Function: checks if the line segment pq intersects any of the no fly zones stored in the list of polygons called buildings. If that is the case, if returns true and false otherwise.

2.3 AIRQUALITYDATA CLASS

2.3.a Fields

- location: A string instance variable to hold the What3Words location of a sensor. It has a public access modifier.
- battery: A float instance variable to hold the battery level of the sensor. It has a public access modifier.
- reading: A string instance variable to hold the air quality reading. It has a public access modifier.

2.4 WORDS CLASS

2.4.a Fields

- country: A private string instance variable to hold the country of the sensor.
- square: A private Square instance variable.
- nearestPlace: A private string instance variable that holds the place closest to the sensor.
- coordinates: A private LngLat instance variable that holds the coordinates of the sensor.
- words: A string instance variable to hold the What3Words location of a sensor. It has a private access modifier.
- language: A string instance variable that holds the language spoken in the country where sensor is located. It has a private access modifier.
- map: A string instance variable that holds the link to where sensor is located. It has a private access modifier.

2.4.b Method

- getCoordinates:
  Parameters: no parameters
  Return type: LngLat (class which has the longitude and latitude parts of the coordinates)
  Function: a getter for the coordinates of the sensor.
  Access modifier: public

- getWord
  Parameters: no parameters
  Return type: String
  Function: a getter for the what3words of the sensor.
  Access modifier: public

2.4.c Subclasses within Words class

- Square class has two private fields: southwest (type LngLat), northeast (type LngLat).
- LngLat class has two private fields: lng (type double), lat (type double) and two getter methods which are public. getLng (gets the longitude and returns it) and getLat (gets the latitude and returns it).

2.5 SERVERDATA CLASS

2.5.a Fields

- polygons: A public static variable to store the no fly zones in a list of polygons.
- airQualitySensors: A public static variable to hold the air quality data of the sensors in a list.

2.5.b Methods

- getNoFlyZones
  Parameters: no parameters
  Return type: List of polygons (List<Polygon>)
  Function: sends an http request to server to get all the no fly zones in the buildings folder in a form of polygons and put them in a list which is returned.
  Access modifier: private
  This method is static to get memory only once in the class area at the time of class loading.

- getAirQualityListDataList
  Parameter: no parameters
  Return type: List of AirQualityData (List<AirQualityData>)
  Function: sends an http request to server to get all the air quality data for the sensors for a particular date from the maps folder and put them in a list which is returned.
  Access modifier: public, it is needed in another class.
  This method is static to get memory only once in the class area at the time of class loading.

- getLocationDetails
  Parameter: String Location (the what3 Words location of a sensor)
  Return type: an instance of Words class

Function: sends an http request to server to get the details of a particular sensor when given the what3 words location of that sensor from the words folder and returns the details as an instance of the Words class.
Access modifier: public, it is needed in another class.
This method is static to get memory only once in the class area at the time of class loading.

2.6 SENSORINFO CLASS

2.6.a Fields

- point: A private instance variable of type Point to hold the point of the sensor.
- location: A private instance variable of type String to hold the What3 Words location of the sensor.
- battery: A private instance variable of type float to hold the battery level of the sensor.
- reading: A private instance variable of type String to hold the air quality reading.

This class has a constructor, SensorInfo that takes Point point, String location, float battery, String reading and it is used to initialize the object of the class.

2.6.b Methods

- getPoint (public access modifier)
  Parameters: no parameters
  Return type: Point
  Function: gets the Point of a sensor and returns it.

- getLocation (public access modifier)
  Parameters: no parameters
  Return type: String
  Function: gets the location of a sensor and returns it.

- getBattery (public access modifier)
  Parameters: no parameters
  Return type: float
  Function: gets the battery level of a sensor and returns it.

- getReading (public access modifier)
  Parameters: no parameters
  Return type: String
  Function: gets the air quality reading and returns it.

2.7 OUTPUT CLASS

2.7.a Fields

- move: A private instance variable of type int to hold the move number.
- prevLng: A private instance variable of type double to hold the longitude of previous drone position.
- prevLat: A private instance variable of type double to hold the latitude of previous drone position.
- angle: A private instance variable of type int to hold the direction of flight from one point to another.

- curLng: A private instance variable of type double to hold the longitude of current drone position.
- curLat: A private instance variable of type double to hold the latitude of current drone position.
- location: A private instance variable of type String to hold the What3 Words location of the sensor
- battery: A private instance variable of type float to hold the battery level of the sensor.
- reading: A private instance variable of type String to hold the air quality reading.

This class has a constructor which is used to create an instance object and it takes int move, double prevLng, double prevLat, int angle, double curLng, double curLat, String location, float battery and String reading as parameters.

2.7.b Methods

- getBattery (public access modifier)
  Parameters: no parameters
  Return type: float
  Function: gets the battery level of a sensor and returns it.

- getReading (public access modifier)
  Parameters: no parameters
  Return type: String
  Function: gets the air quality reading and returns it.

- getPrevLng (public access modifier)
  Parameters: no parameters
  Return type: double
  Function: gets the longitude of previous drone position and returns it.

- getPrevLat (public access modifier)
  Parameters: no parameters
  Return type: double
  Function: gets the latitude of previous drone position and returns it.

- getCurLng (public access modifier)
  Parameters: no parameters
  Return type: double
  Function: gets the longitude of current drone position and returns it.

- getCurLat (public access modifier)
  Parameters: no parameters
  Return type: double
  Function: gets the latitude of current drone position and returns it.

- getLocation (public access modifier)
  Parameters: no parameters
  Return type: String
  Function: gets the location of a sensor and returns it.

- toString (public access modifier)

Parameter: no parameter
Return type: String
Function: used to print out an object in a string format (in this case, the format needed in our flight path txt file). It overrides Java's toString() method.

2.8 MOVEMENT CLASS

2.8.a Fields

- LINE_DIST: A private final instance variable of type double that holds the distance for a move which is 0.0003.
- sensorInfoList: A private instance variable of type List<SensorInfo> that holds the full details of a sensor i.e., the point of the sensor, the location, the battery level and air-quality reading.
- outputs: A public instance variable to hold the flight path taken.

2.8.b Methods

- calculateDistance
  Parameters: Point p1, Point p2
  Return type: double
  Function: calculates the distance between the two points and return the result.
  Access Modifier: private

- getAngle
  Parameters: Point curPoint, double flightAngle
  Return type: double
  Functions: finds an angle such that a path to be taken does not intersect a no fly zone.
  Access Modifier: private

- findNearestPoint
  Parameters: Point curPoint, List<Point> sensorPoints
  Return type: Point
  Function: finds the nearest sensor from the list of sensors, sensorPoints to the current point, curPoint and returns that point.
  Access Modifier: private

- calculateNextPoint
  Parameters: Point curPoint, double angle
  Return type: Point
  Function: Since each move should have a distance of 0.0003, this method finds the next point from the current point, curPoint using the angle and trigonometry functions (sine and cosine) and returns it.
  Access Modifier: private

- findDirectionOfTravel
  Parameters: Point curPoint, Point nearestSensor
  Return type: double
  Function: finds the angle that gives the shortest distance from next Point (calculated from current Point, curPoint) and nearestSensor.
  Access Modifier: private

- getSensorInfoList
  Parameters: List<AirQualityData>
  Return type: List<SensorInfo>
  Function: gets all the necessary details about all the sensors in the List of air quality Data and puts them in a list which is then returned.
  Access Modifier: private

- getSensors
  Parameters: no parameters
  Return type: List<Point>
  Function: gets all sensor from the List<SensorInfo> sensorInfoList, puts them in a list and returns that list.
  Access Modifier: private

- getDataReadingForPoint
  Parameters: Point p
  Return type: SensorInfo
  Function: given a point p (sensor), it gets all the details of that point and returns it.
  Access Modifier: private

- moveDrone
  Parameters: no parameters
  Return type: List<Output>
  Function: moves the drone, records each move taken and adds to a list which it returned.
  Access Modifier: public, called in the App main method.

2.9 WRITINGTOFILES CLASS

2.9.a Field

- movement: An instance of the movement class (private).

2.9.b Methods

- getColor
  Parameters: String reading, float battery
  Return type: String
  Function: gets the colour in the form of a string of a sensor using the sensor's air quality reading and battery level and returns it.
  Access Modifier: private, a helper method for getGeoJsonFeatureCollection method.

- getMarkerSymbol
  Parameters: String reading, float battery
  Return type: String
  Function: gets the marker symbol of a sensor using the sensor's air quality reading and battery level and returns it.
  Access Modifier: private, a helper method for getGeoJsonFeatureCollection method.

- getVisitedPoints
  Parameters: List<Output> outputs
  Return type: List<Point>

Function: gets all the sensor points that have been visited from outputs (the total flightpath taken by the drone), puts them in a list and returns it.
Access Modifier: private, a helper method for getGeoJsonFeatureCollection method.

- getGeoJsonFeatureCollection
  Parameters: List<Output>outputs
  Return type: FeatureCollection
  Function: creates a feature collection of the total flight path stored in outputs.
  Access Modifier: private, a helper method for writeToGeoJsonFile method.

- writeToGeoJsonFile
  Parameters: no parameters
  Return type: void
  Function: converts a feature collection to json and writes the result to a geojson file.
  Access Modifier: public, used it in App main method.

- writeToOutputFile
  Parameters: no parameters
  Return type: void
  Function: writes the total flight path to a text file.
  Access Modifier: public, used it in App main method.

3. DRONE ALGORITHM

3.1 How my drone algorithm works

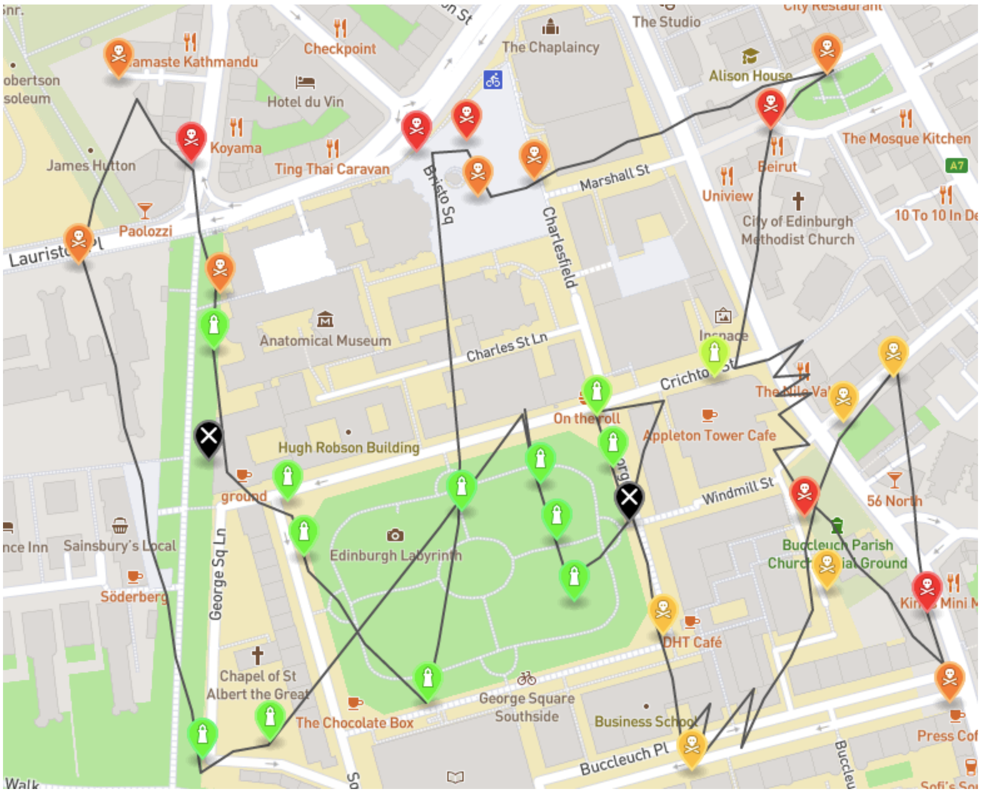
This algorithm uses the idea of shortest path to visit the sensors i.e., the sensor closest to the drone's current position is found among the list of sensors and based on that the best angle among 36 angles (0 - 350) which gives the shortest path to that sensor is taken and used to calculate the drone's next position.

- ❖ Before drone moves, get the list of sensor points for a particular date, start point, set limit of total moves to 150, number of moves to 0, declare an empty list to hold total flight path taken.
- ❖ Set the previous and current positions to the start point.
- ❖ Find nearest sensor, flight angle (direction of travel) to that sensor and next position to move to.
- ❖ While the number of moves is less than the limit of total moves,
  - ➢ check if the next position is in the confinement area
  - ➢ if so, check if the path from drone's current position to that next position intersects any of the no fly zones
  - ➢ if there is no intersection, move to that next position by setting previous position to current position, current position to next position and increase number of moves by 1.
  - ➢ After moving, check if the drone's current position is within 0.0002 of the closest sensor, if so, take air quality reading and record the move and remove that sensor from the list of sensors.
    - ♦ Check if the number of sensors in the list is less than 5, if so, add the start point and check if the drone within 0.0003 of the start point, if that is the case drone stops moving.
  - ➢ if the drone is not within 0.0002 of the closest sensor, record the flight path.

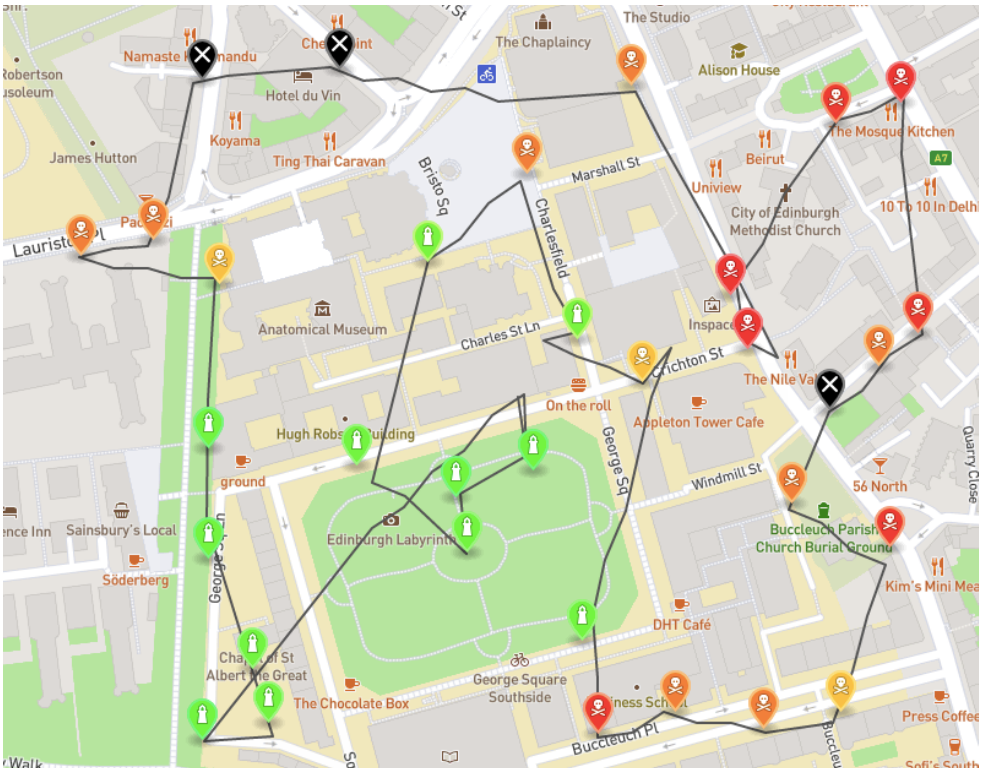
- Set previous position to drone’s current position, find our next closest sensor (which could be the same sensor as before), find the flight angle and next position.
- If the path from the drone’s current position to the next position intersected any of the no fly zones, a different direction of travel which causes no intersection is calculated using the getAngle method and next position is calculated too.
- Suppose the drone’s next position was out of the confinement area, the drone moves back to its previous position and that sensor is removed from the list of sensors.
- Then, check if the list of sensors is empty, if so, add the start point and check if the drone’s current position is within 0.0003 of the start point. If that is the case, the drone stops moving.
- Else, find next nearest sensor to visit, new flight angle (direction of travel) and next position.

  *The drone tries to visit as many sensors as it can.

Map showing drone flight for 27/12/2020



Map showing drone flight for 16/05/2021



4. APPENDIX AND REFERENCES

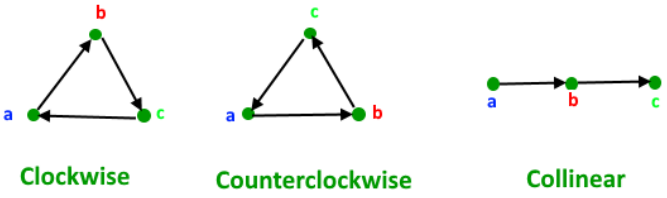
4.1 Appendix

This section contains detailed explanation of how some methods in the Zones class works.

Below are the diagrams showing the orientations in findOrientation method.

(The code used in this project and the explanation was gotten from geeksforgeeks website referenced at the end)

Possible orientations for 3 points a, b, c



Details of how the doLinesIntersect Method works

This method first finds the orientations for the two lines i.e., line segment p1q1 and line segment p2q2 which are listed below using the findOrientation method.

- Orientation between line segment p1q1 and point p2 → orientation1

- Orientation between line segment p1q1 and point q2 → orientation2
- Orientation between line segment p2q2 and point p1 → orientation3
- Orientation between line segment p2q2 and point q1 → orientation4

Cases when doLinesIntersect method return true:

- if orientation1 is not the same as orientation2 and orientation3 is not the same as orientation4. (general case)
- Points p1, q1 and p2 are collinear and p2 lies on segment p1q1.
- Points p1, q1 and q2 are collinear and q2 lies on segment p1q1.
- Points p2, q2 and p1 are collinear and p1 lies on segment p2q2.
- Points p2, q2 and q1 are collinear and q1 lies on segment p2q2.

Diagrams showing the general cases and special cases of intersections of 2 line segments

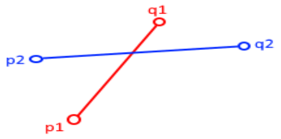
Two segments (p1, q1) and (p2, q2) intersect if and only if one of the following two conditions is verified:

1. *General Case:*

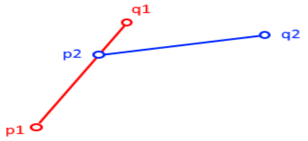
– (p1, q1, p2) and (p1, q1, q2) have different orientations and

– (p2, q2, p1) and (p2, q2, q1) have different orientations.

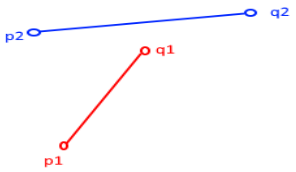
Examples shown below



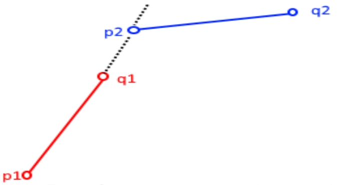
Example : Orientations of (p1,q1,p2) and (p1,q1,q2) are different. Orientations of (p2,q2,p1) and (p2,q2,q1) also differnet.



Example: Orientations of (p1,q1,p2) and (p1,q1,q2) are different. Orientations of (p2,q2,p1) and (p2,q2,q1) also different



Example: Orientations of (p1,q1,p2) and (p1,q1,q2) are different. Orientations of (p2,q2,p1) and (p2,q2,q1) are same



Example: Orientations of (p1,q1,p2) and (p1,q1,q2) are different. Orientations of (p2,q2,p1) and (p2,q2,q1) are same.

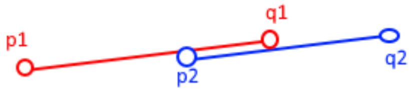
2. *Special Case*

– (p1, q1, p2), (p1, q1, q2), (p2, q2, p1), and (p2, q2, q1) are all collinear and

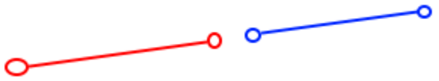
– the x-projections of (p1, q1) and (p2, q2) intersect

– the y-projections of (p1, q1) and (p2, q2) intersect

Examples shown below



Example: All points are collinear. The x-projections pf (p1,q1) and (p2,q2) intersect. The y-projection of (p1,q1) and(p2,q2) also intersect



Example: All points are collinear .The x-projections of (p1,q1) and (p2,q2) not intersect. The y-projection of (p1,q1) and (p2,q2) do not intersect

4.2 REFERENCES

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

2. Check if two line segments intersect
https://www.geeksforgeeks.org/check-if-two-given-line-segments-intersect/

3. Drawing of maps
http://geojson.io/

4. Writing to a file in Java
https://www.javatpoint.com/java-filewriter-class/
https://stackoverflow.com/questions/2885173/how-do-i-create-a-file-and-write-to-it-in-java/