Informatics Large Practical

Stephen Gilmore and Paul Jackson
School of Informatics, University of Edinburgh

Document version 1.0.2
First issued on: September 22, 2021
First revised on: September 23, 2021
Date of this revision: September 28, 2021

About

The Informatics Large Practical is a 20 point Level 9 course which is available for Year 3 undergraduate students on Informatics degrees. It is not available to visiting undergraduate students or students in Year 4 or Year 5 of their undergraduate studies. It is not available to postgraduate students. Year 4, Year 5 and postgraduate students have other practical courses which are provided for them.

Scope

The Informatics Large Practical is an individual practical exercise which consists of one large design and implementation project, with two coursework submissions. Coursework 1 involves creating a new project and implementing some fundamental components of the project. Coursework 2 is the implementation of the entire project together with a report on the implementation.

Courseworks	Deadline	Out of	Weight
Coursework 1	16:00 on Friday 15th October 2021	25	25%
Coursework 2	16:00 on Friday 3rd December 2021	75	75%

Please note that the two courseworks are not equally weighted. There is no exam paper for the Informatics Large Practical so to calculate your final mark out of 100 for the practical just add together your marks for the courseworks.

Introduction

You have been asked to help the School of Informatics investigate an idea for a service which should allow the students in the School to make the most of their lunch hour where they take a much-needed break from lectures, studying, labs, and practicals like this one! The School of Informatics is considering creating a service where students can place a lunch order for sandwiches and drinks and have these delivered to them while they take a break in Bristo Square or George Square. Lunches will be delivered by an autonomous airborne drone which will travel to sandwich shops in the University Central Area to collect the lunch items in a lunch order, and then fly to a chosen location to deliver these to the student who placed the order. It is hoped that this service will allow students to make the most of their lunch break because they will not need to queue at the shops and collect their lunch in person. In addition, the service could be helpful to new students who have just joined the School and have not yet got their bearings and do not know the sandwich shops near the Central Area.

—◇—

The idea for a drone-based sandwich delivery service has not been universally popular. Some of the schools in the University Central Area think that the potential for error in deliveries is too great and they do not want to deal with problems such as drones crashing into their building, or getting stranded on the roof of the building, or dropping sandwiches and drinks onto their staff and students. It is expected that some of these schools will define a "no-fly zone" consisting of their buildings. The drone will therefore have to plan its routes so that it does not fly over buildings in the no-fly zone.

—◇—

Ordering the lunches is relatively straightforward. The School of Informatics can easily create an online system to take student lunch orders in the morning and add these to a database of orders to be delivered at lunchtime. What is less clear is whether or not it is feasible for the drone to fulfil these orders, given that (i) the service is expected to be popular with a lot of lunch orders being placed each day, (ii) only one drone is available for making the deliveries, (iii) the drone cannot carry more than one order at a time (to avoid delivering the wrong order to the wrong person), (iv) the drone must avoid buildings in the no-fly zone, and (v) the drone can only fly for a limited time before its battery will run out and it will need to be recharged. Recharging is a slow process which means that the drone will no longer be in service that lunchtime.

—◇—

Your task is to devise and implement an algorithm to control the flight of the drone as it makes its deliveries while respecting the constraints on drone movement specified in this document. You will be provided with some synthetic test data representing typical lunch orders and other data about the service such as the details of the sandwich shops which are participating in the scheme, the menus for these shops, and the location of the drop-off points where deliveries can be made. This information will come in the form of a database (for the order information) and a website (for the rest of the information). It is important to stress that the information in the test data which you will be given only represents the current best guess at what the elements of the drone service will be when it is operational, and the service in practice might use different shops or it might deliver to different drop-off points. For this reason, your solution must be *data-driven*. That is, it must read the information from the database and the website and particular shops or particular drop-off points or other details must not be hardcoded in your application, except where it is explicitly stated in this document that it is acceptable to do so.

—◇—

Your implementation of the drone control software should be considered to be a prototype only in the sense that you should think that it is being created with the intention of passing it on to a team of software developers and student volunteers in the School of Informatics who will maintain and develop it in the months and years ahead when the drone lunch delivery is operational. For this reason, the clarity and readability of your code is important; you need to produce code which can be read and understood by others.

Latitudes and longitudes

In this practical we will be using latitudes and longitudes to identify locations on the map (such as shops and drop-off points).

- ▷ Longitude is the measurement east or west of the prime meridian.
- ▷ Latitude is the measurement of distance north or south of the Equator.

(The above are National Geographic definitions.) Latitudes and longitudes are measured in *degrees*, so we stay with this unit of measurement throughout all our calculations. Even when we are calculating the *distance* between two points we express this in degrees rather than metres or kilometres to avoid unnecessary conversions between one unit of measurement and another. As a convenient simplification in this practical, locations expressed using latitude and longitude are treated as though they were points on a plane, not points on the surface of a sphere. This simplification allows us to use Pythagorean distance as the measure of the distance between points. That is, the distance between (x_1, y_1) and (x_2, y_2) is just

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

— ◇ —

In general, it will not be possible to manoeuvre the drone to a specified location exactly. Being *close to* the location will be sufficient, where ℓ_1 is *close to* ℓ_2 if the distance between ℓ_1 and ℓ_2 is strictly less than the *distance tolerance* of 0.00015 degrees.

— ◇ —

When we write a location as a pair of coordinates in this document we will use the convention (*longitude, latitude*) because the language which we use for rendering maps puts longitude first and latitude second. In this project, longitudes will always be negative (~ -3) and latitudes will always be positive ($\sim +56$) because Edinburgh is located at (approximately) longitude 3 degrees West and latitude 56 degrees North.

The movement of the drone

The flight of the drone is subject to the following stipulations:

- the drone can make at most 1500 moves before it runs out of battery;
- the moves are of two types, the drone can either *fly* or *hover*—the drone can change its latitude and longitude when it flies, but not when it is hovering i.e. when it makes a hover move—flying and hovering use the same amount of energy;
- every move when flying is a straight line of length 0.00015 degrees[1];
- the drone *cannot fly in an arbitrary direction*: it can only be sent in a direction which is a multiple of ten degrees where we use the convention that 0 means go East, 90 means go North, 180 means go West, and 270 means go South, with the other multiples of ten between 0 and 350 representing the obvious directions between these four major compass directions[2];
- when the drone is hovering, we use the obvious junk value of −999 for the angle, to indicate that the angle does not play a role in determining the next latitude and longitude of the drone;
- the drone *must hover for one move* when collecting a lunch order from a shop, or delivering a lunch order to a customer;
- the drone is launched each day from the top of the Appleton Tower at location (−3.186874, 55.944494) and should return *close to* this location when the day's deliveries are complete.

[1] Because of unavoidable rounding errors in calculations with double-precision numbers these moves may be fractionally more of fractionally less than 0.00015 degrees. Differences of $\pm 10^{-12}$ degrees are acceptable. Double-precision numbers must be used to represent quantities measured in degrees because of the need for accuracy in specifying locations.

[2] Negative angles are not allowed when specifying the direction of flight, and angles greater than 350 are not allowed. The convention that we use for angles simplifies the calculations of drone positions.

The Drone Confinement Area

All locations which need to be visited have a latitude which lies between 55.942617 and 55.946233. They also have a longitude which lies between −3.184319 and −3.192473. There is no reason for the drone to be outside this area, so these coordinates define the *drone confinement area* as illustrated in Figure 1. The drone must at all times remain *strictly inside* the confinement area. To be clear, if the drone is at location (−3.192473, 55.946233) then it is *outside* the confinement area, and is judged to be malfunctioning.

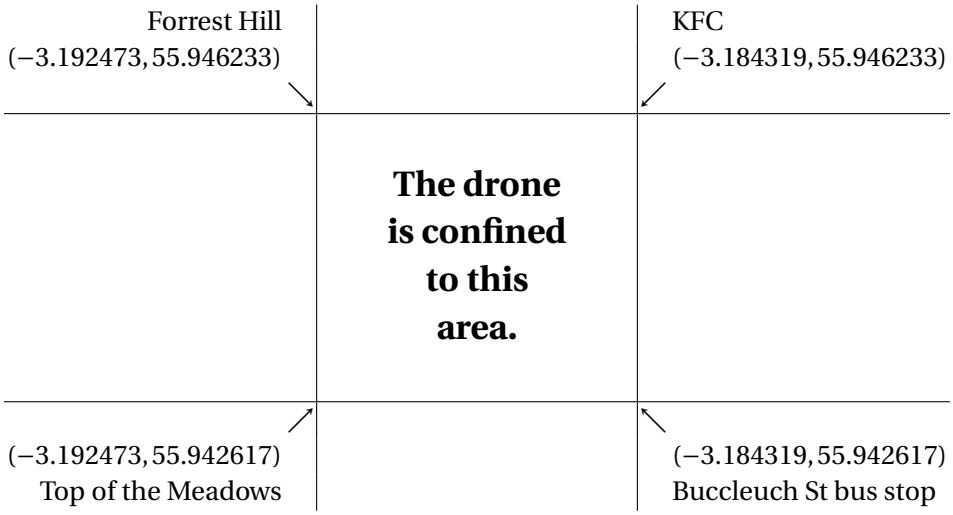

Figure 1: The drone confinement area

Use of *WhatThreeWords* addresses

One issue which the architects of the drone sandwich delivery service had to solve was how to direct customers of the system to their desired location to collect their lunch order. Customers can choose from a fixed list of options where they want their order delivered at the time of placing their order but the delivery locations are virtual in nature and are not marked with a pole or flag or other signpost. It would be possible to give the customer a location in the form of a (longitude, latitude) pair such as (−3.18933, 55.943389) but past experience with handling numerical values on previous projects has shown that it is very easy to transpose digits when keying location information. In our use-case, this would send customers to the wrong location to collect their delivery, which would be highly undesirable. It would be preferable not to work with raw numerical values for locations in order to make the service easier for customers to use.

—◇—

The *What3Words* encoding is a novel location addressing system which is used to guide users to locations where there is no street address available[3]. Instead of using a raw numerical (longitude, latitude) pair such as (−3.18933, 55.943389), users can use a What3Words address made up of three words instead, such as `less.change.atomic`. The use of simple words in the name makes these addresses a lot easier to type than (longitude, latitude) pairs. There is never more than one collection point in any What3Words tile so the WhatThreeWords address can be used as the location of the collection point and the associated WhatThreeWords mobile phone app can be used to guide the user to the right location. It is hoped that the use of What3Words addresses for this service will reduce the number of times that a customer goes to the wrong place to collect their lunch delivery.

[3] The What3Words system maps the earth's surface using 3m × 3m tiles, each with a unique three word address. More information is available at `https://what3words.com/`.

The web server content

The information which the service needs to function is stored in a database and on a web server. Information which is transient (relevant only for a single day, such as lunch orders) will be stored in the database. Information which has greater longevity (changed only one a month or even less frequently, such as the details of the shops participating in the service) will be stored on the web server. In order to help with the development of the software for the drone, the architects on the drone delivery service have made available the web content for a web server with test data which is organised in the way that the data will organised when the service is operational. The web server has three top-level folders, `buildings`, `words`, and `menus`.

The `buildings` folder: As we know, the drone must always remain within the drone confinement area and if it ventures outside that area it is considered to be malfunctioning. Within the confinement area are five regions where the drone is not allowed to fly; a drone entering these regions will also be considered to be malfunctioning. These five regions are known as the *no-fly zones* for the drone. The details of the no-fly zones are given in the file `buildings/no-fly-zones.geojson`. This file is in GeoJSON format, which is a standard way of encoding geographic data structures and map information. For more details on GeoJSON visit `https://geojson.org`. To render GeoJSON maps, visit `https://geojson.io`.

In the test data there are five no-fly zones where the drone is not allowed to be. These cover the McEwan Hall complex; the Teviot building; the Wilkie Building; the Psychology and Neuroscience buildings; and the Chrystal Macmillan and Hugh Robson buildings. The drone cannot move into any of the no-fly zones with a move, and it cannot *pass through* any no-fly zones in moving from one point to the next.

Also present in the `buildings` folder is a file of test data called `landmarks.geojson`. This identifies landmarks which the drone is allowed to fly close to when navigating. These do not serve the functions of collecting or delivering food; they are intended to be used when planning routes for the drone to fly around the no-fly zones.

The no-fly zones and the landmarks are illustrated in Figure 2, together with the drone confinement area showing the limit of the drone's position.

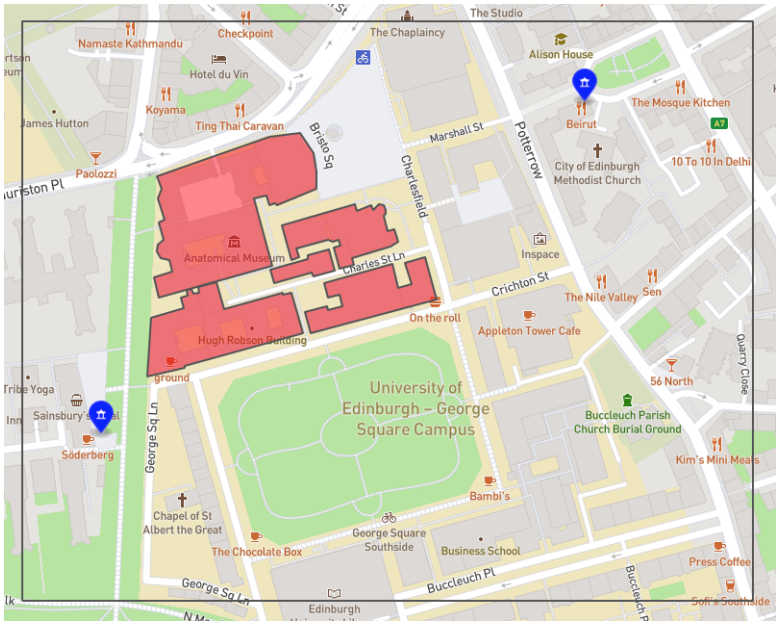

Figure 2: A GeoJSON map rendered by the website http://geojson.io/. The map shows the drone confinement area (the outer grey rectangle), the five no-fly zones (the semi-transparent red polygons), and the two landmarks (the blue teardrop-shaped markers).

The `words` folder: The `words` folder contains JSON (JavaScript Object Notation) format files which give the What3Words information corresponding to a What3Words address. The details for a What3Words address "`first.second.third`" will be stored in the file `words/first/second/third/details.json`. The most important field in this record for our purposes is the `coordinates` field which allows us to associate a What3Words address with a specific longitude (`"lng"`) and longitude (`"lat"`).

The What3Words file corresponding to the address `"less.change.atomic"` is shown in Figure 3. The important `coordinates` field is highlighted in bold.

```
{
  "country": "GB",
  "square": {
    "southwest": {
      "lng": -3.189355,
      "lat": 55.943375
    },
    "northeast": {
      "lng": -3.189306,
      "lat": 55.943402
    }
  },
  "nearestPlace": "Edinburgh",
  "coordinates": {
    "lng": -3.18933,
    "lat": 55.943389
  },
  "words": "less.change.atomic",
  "language": "en",
  "map": "https://w3w.co/less.change.atomic"
}
```

Figure 3: The content of the file `words/less/change/atomic/details.json`

The `menus` folder: The `menus` folder contains the JSON file `menus.json`. This file contains the information about the shops which are participating in the drone lunch delivery service.

The JSON structure of this file is the following:

- Overall, the file contains a JSON list. Each item in this list is a JSON record giving information about a shop.
- The JSON record for a shop has three fields:
  - `name` (a string): the name of the shop;
  - `location` (a string): the location of the shop as a WhatThreeWords address; and
  - `menu` (a list): the items on sale in the shop, together with their price in pence.
- The JSON record for an item on sale has two fields:
  - `item` (a string): the item being sold;
  - `pence` (an integer): the price in pence of this item[4].

An extract from the file `menus/menus.json` appears in Figure 4.

[4] Prices are expressed as integers instead of floats to avoid rounding errors when calculating the total price of an order.

```
[
  {
    "name" : "Rudis",
    "location" : "sketch.spill.puzzle",
    "menu" :
      [
        { "item" : "Ham and mozzarella Italian roll", "pence" : 230 },
        { "item" : "Salami and Swiss Italian roll", "pence" : 230 },
        { "item" : "Cambozola and tomato Italian roll", "pence" : 230 },
        { "item" : "Goat's cheese salad Italian roll", "pence" : 230 },
        ...
      ]
  },
  ...
  {
    "name" : "Bing Tea",
    "location" : "looks.clouds.daring",
    "menu" :
      [
        { "item" : "Flaming tiger latte", "pence" : 460 },
        { "item" : "Dirty matcha latte", "pence" : 460 },
        { "item" : "Strawberry matcha latte", "pence" : 460 },
        { "item" : "Fresh taro latte", "pence" : 460 },
        ...
      ]
   }
]
```


Figure 4: An extract from the file `menus/menus.json`

The makeup of a lunch order

We haven't said anything so far about the nature of a lunch order so let's discuss that now. As you might imagine, there is a limit on the weight that the drone can lift. The maximum number of items in an order has been fixed so that the drone will always be able to lift the order, even if it consists of the heaviest items which can be ordered by the drone service. There are other constraints also, as listed below.

1. An order can have a minimum of one item, and a maximum of four.
2. An order can be made up of items from no more than two shops.
3. Every order is subject to a fixed delivery charge, which is 50p.

The shops which participate in the service are notified when the drone will arrive, they bag up the items in the order, and add the bag to the drone when it is hovering close to the location of the shop[5]. We will not be concerned here with the system which sends notifications to the shops; the architects of the drone service already have a system in place for this.

```
/**
 * Fetch and setup one order from web and
database
 * @param orderNo order number fetch from
database
 * @param deliverTo word3word delivery location
fetch from database
 * @throws OrderValidationException if the database could
not be accessed
 */
```

The structure of the database

The information about each day's orders is stored in an Apache Derby database[6]. This is a relational database which can be accessed by SQL queries. The database has a simple design and all of the necessary information can be accessed using basic SQL queries and update commands. Examples of these will be given in the course lecture on Derby.

[5] We imagine that the items are placed in a basket that is hanging down from the drone so that the drone is always hovering some safe height above the user's head.

[6] Information on the Apache Derby project is available at `https://db.apache.org/derby`. Instructions for running the database server are in Appendix C beginning on page 26 of this document.

The information on lunch orders has been written to a database table called `orders`. This table has four columns:

- `orderNo` — an eight-character hexadecimal string giving the unique order number for this order;
- `deliveryDate` — a `java.sql.Date` object specifying the date on which the order is to be delivered;
- `customer` — an eight-character string with the matriculation number of the student who placed the order; and
- `deliverTo` — a variable-length string of at most 18 characters giving the WhatThreeWords address of the delivery location.

The `orders` table has been created using the following SQL command.

```
create table orders(orderNo char(8),
                    deliveryDate date,
                    customer char(8),
                    deliverTo varchar(18))
```

A sample of data from the `orders` table is shown below.

orderNo	deliveryDate	customer	deliverTo
"987526aa"	*2022-04-11*	"s2271919"	"surely.native.foal"
⋮	⋮	⋮	⋮
"d7d0821c"	*2022-05-14*	"s2238543"	"truck.hits.early"
⋮	⋮	⋮	⋮

—◇—

The database also contains a second table called `orderDetails`. This contains the details of which menu items are in an order. This table has two columns:

- `orderNo` — an order number linking this information back to an order in the `orders` table; and
- `item` — a variable-length string of at most 58 with the name of a menu item in the order.[7]

The `orderDetails` table has been created using the following SQL command.

```
create table orderDetails(orderNo char(8),
                          item varchar(58))
```

A sample of data from the `orderDetails` table is shown below.

orderNo	item
"987526aa"	"Cambozola and tomato Italian roll"
"987526aa"	"Goat's cheese salad Italian roll"
⋮	⋮
"d7d0821c"	"Feta, olives, Greek yoghurt and tomato French country roll"
"d7d0821c"	"Hummus and salad Italian roll"
"d7d0821c"	"Strawberry matcha latte"
⋮	⋮

From this we learn that order number `987526aa` was for two items, and order number `d7d0821c` was for three items.

[7] You could argue that the database would be more efficient to access if it contained numeric item codes instead of strings with item names, and that would be true, but the architects of the system have chosen to use strings instead because they believe that it will make the application code easier to debug by removing a layer of indirection mapping codes to items.

The synthetic data in the database which you are given to test your application covers a two-year period beginning on 01/01/2022 and ending on 31/12/2023. The synthetic data uses the idea that the students who use the service will be new entrants to the university so their matriculation numbers are all of the form "s22*nnnnn*" or "s23*nnnnn*" where each character *n* is a digit between 0 and 9.

Files to be used during testing

In addition to the web server content and the database which are to be accessed when your application is running, you will also be given some GeoJSON files which have been prepared for you to use when you are *testing* your application. These will be come from a file called `testing.zip` on the course website.

—◇—

When the drone delivery service is operational the web server content and the database will be kept up-to-date but the GeoJSON files prepared for testing purposes will not; they only relate to the synthetic data that you are given to test your application, not the lunch orders received each day when the service is operational. This means that your application should not read these GeoJSON files, you should only load them on the http://geojson.io/ website when checking the flightpath that you have generated for the airborne drone.

—◇—

Unlike, for example, the GeoJSON file of the no-fly zones, the content of the database is not held in a graphical format so the purpose of the GeoJSON test files is to allow us to produce a visualisation of the information in the database, to help with understanding the important locations which are in the synthetic data. One file, `testing/all.geojson`, includes representations of all of the locations of interest in the drone confinement area. This is shown in Figure 5. This file will make a convenient background when rendering your drone flightpath.

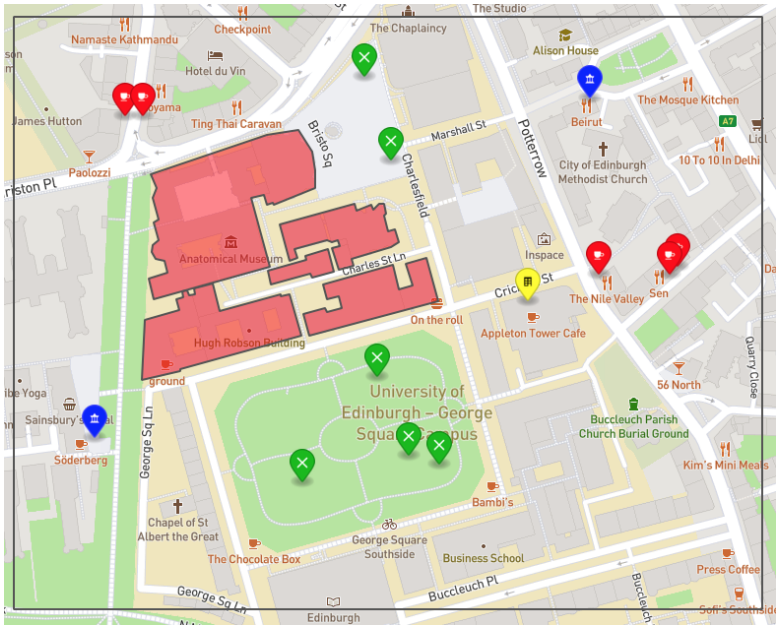

Figure 5: The contents of the file `testing/all.geojson` rendered by the website http://geojson.io/. This file contains all of the features that we have seen in Figure 2 plus the initial location of the drone (the yellow placemarker, on top of Appleton Tower), the five shops which are participating in the scheme according to the website content `menus/menus.json` (the red placemarkers, with a coffee cup symbol), and the six delivery locations according to the synthetic data in the database (the green placemarkers with a white X symbol).

The runtime of your code

In operation, the drone service will take lunch orders each day in the morning until 11:59. Your application will then be run to plan the flightpath of the drone with the intention of launching the drone to begin to deliver the service for that day at 12:00. To achieve this, your application to plan and plot the flightpath of the drone should aim to have a runtime of *60 seconds or less.* You need to bear this restricted runtime in mind when designing the algorithm that you will use to generate the flightpath of the drone.

—◇—

Runtimes which are much longer than 60 seconds, for example an average of 10 or 20 minutes, would obviously be problematical because they are delaying the launch of the drone considerably. Delays in the delivery of their lunch are sure to be unpopular with hungry students so an algorithm with a runtime of 10 minutes would be quite unsatisfactory.

—◇—

When the drone delivery service is operational the *über JAR* of your application[8] will be deployed on a server running Ubuntu 20.04 (focal) DICE. The precise machine to host the service has not been purchased yet but its specifications will be similar to or better than the machine `student.compute.inf.ed.ac.uk`. A good way to check whether your application will be able to deliver the required level of performance in deployment would be to time the runtime of your *über JAR* on the machine `student.compute.inf.ed.ac.uk` when it is lightly-loaded[9].

Judging the viability of the service

It is accepted that the service may not be able to deliver every order every day depending on the number of orders placed. An important metric to be used in determining the viability of the drone delivery service will be the *sampled average percentage monetary value* delivered by the service. You should have this metric in mind when developing your algorithm for controlling the drone.

—◇—

The percentage monetary value delivered each day is calculated as the total monetary value of deliveries made divided by the total monetary value of orders placed. The fixed delivery charge of 50p per order is included in both totals.

—◇—

The sampled average percentage monetary value is calculated by taking a random sample of days (say 7, 12, 24, or 31 days) and computing the average of the percentage monetary value delivered on each of those days. A small sample of days will give an approximate idea of the viability of the service; larger sample sizes will give a more accurate idea.

[8] The über JAR is the relocatable compiled version of your code packaged together with all of the libraries that it depends on.
[9] For example, when the `who` command lists fewer than ten users using the machine.

The implementation task

As the main part of this practical exercise, you are to develop a Java application which when given a date calculates a flightpath for the drone which delivers the lunch orders placed for that date as best it can before it returns close to its initial starting location. As previously stated, the number of moves made by the drone[10] should be 1500 or fewer.

—◇—

Your application should record the drone's behaviour by creating two database tables to record information on the day's deliveries. The first table records the deliveries made by the drone and the second table records the flightpath of the drone move-by-move. In addition to this you should produce a GeoJSON map which provides a visual representation of the flightpath of the drone.

—◇—

Assuming that your project is named `ilp` then when compiled with the Maven build system your Java application will produce an *über* JAR file in the `target` folder of your project named `ilp-1.0-SNAPSHOT.jar`. If you run this JAR file with the command

```
java -jar target/ilp-1.0-SNAPSHOT.jar  15 09 2022  80 1527
```

it should read the lunch orders for the date 15/09/2022 from the database, connecting at port 1527. It should read the menus from the website, connecting to the web server at port 80. If the database server was instead located at port 9751 and the web server was located at port 8080 then the command would instead be

```
java -jar target/ilp-1.0-SNAPSHOT.jar  15 09 2022  8080 9751
```

Your application may write any diagnostic messages that it likes to the standard output stream provided that the total size of these messages does not exceed 1Mb.

—◇—

The results computed by your algorithm to control the drone will be written to two database tables and a GeoJSON file. You should assume that the database is an Apache Derby database named `derbyDB`.

The database table `deliveries` You should create a database table named `deliveries` using the following SQL command.

```
create table deliveries(orderNo char(8),
                        deliveredTo varchar(19),
                        costInPence int)
```

Please ensure that you use exactly the column names `orderNo`, `deliveredTo`, and `costInPence` for this table. If a table of this name already exists in the `derbyDB` database then it should be deleted (dropped) before this SQL `create table` command is executed. The `deliveries` table which you create should have an entry for every lunch delivery which your drone makes. This entry should consist of:

- `orderNo` — the eight-character hexadecimal string assigned to this order in the `orders` table;
- `deliveredTo` — the WhatThreeWords address of the delivery location; and
- `costInPence` — the total cost of the order, including the standard 50p delivery charge.

[10] Refer to Page 3 for the definition of a move.

The database table `flightpath` You should create a database table named `flightpath` using the following SQL command.

```
create table flightpath(orderNo char(8),
                        fromLongitude double,
                        fromLatitude double,
                        angle integer,
                        toLongitude double,
                        toLatitude double)
```

Please ensure that you use exactly the column names `orderNo`, `fromLongitude`, `fromLatitude`, `angle`, `toLongitude`, and `toLatitude` for this table. If a table of this name already exists in the `derbyDB` database then it should be deleted (dropped) before the SQL `create table` command above is executed. The `flightpath` table which you create should provide a detailed record of every move made by the drone while making the day's lunch deliveries. The entries in this table should consist of:

- `orderNo` — the eight-character order number for the lunch order which the drone is currently collecting or delivering[11];
- `fromLongitude` — the longitude of the drone at the start of this move;
- `fromLatitude` — the latitude of the drone at the start of this move;
- `angle` — the angle of travel of the drone in this move[12];
- `toLongitude` — the longitude of the drone at the end of this move; and
- `toLatitude` — the latitude of the drone at the end of this move.

The output file `drone-`*`DD-MM-YYYY`*`.geojson` This file is generated in the current working directory when the application is run. It is a text file in GeoJSON format[13]. It should contain a FeatureCollection which consists of exactly one Feature. That Feature must be of type LineString. The LineString contains a list of coordinates which illustrate the flightpath of the drone. These coordinates should be approximately equal to the corresponding longitudes and latitudes stored in the `flightpath` table of the database. They will not be exactly equal because by default GeoJSON documents use single-precision floating-point values for longitudes and latitudes whereas the values stored in the database are double-precision. This means that we will take the values in the database to be the definitive flightpath of the drone with the values in the GeoJSON file providing a reasonable approximation for the purposes of visualisation.

—◇—

When it is rendered by the website `http://geojson.io` along with the contents of the testing file `testing/all.geojson`, your `drone-`*`DD-MM-YYYY`*`.geojson` file should produce a visualisation similar to that in Figure 6 with a grey line showing the flightpath of the drone. The shape of this line will depend on the date being plotted and on the drone control algorithm which you devise.

—◇—

Note: Please use hyphens, not underscores, in the name of this file and use only lowercase letters. A filename of `drone-15-09-2022.geojson` is acceptable; a filename like `Drone_15-09-2022.GEOJSON` is not. Please note that your filename must use two digits for the day and the month; a filename of `drone-15-9-2022.geojson` is not acceptable because it does not match the pattern for the filename of `drone-`*`DD-MM-YYYY`*`.geojson`.

[11] The contents of this field are not important when the drone is making the flight back to the top of the Appleton Tower when all of the day's orders have been delivered.

[12] Refer to Page 3 of this document for the allowable values which this field can take.

[13] Please see `http://geojson.org` for details of the GeoJSON format.

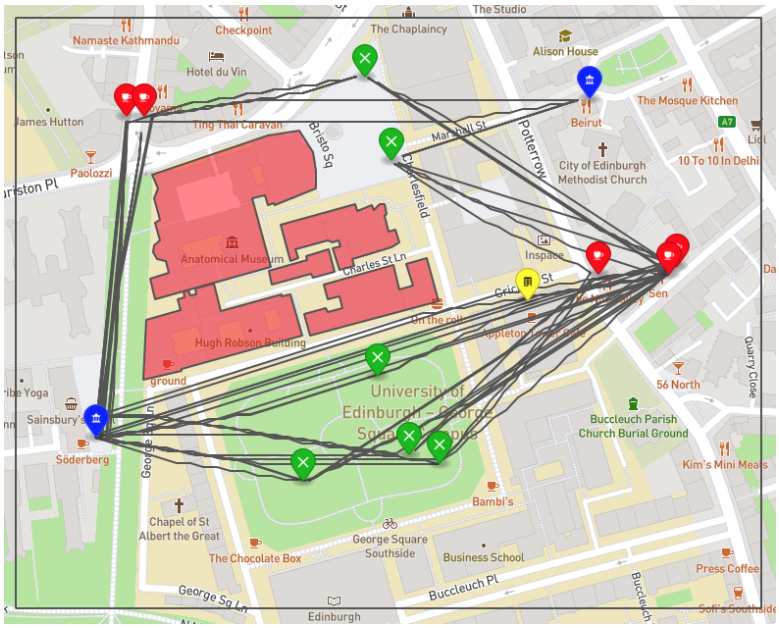

Figure 6: The contents of the file `testing/all.geojson` (from Figure 5) overlaid with an output GeoJSON file with the drone flightpath, rendered together by the website http://geojson.io/. The grey squiggly line connecting the initial location (the yellow marker), the shops (the red markers), the delivery locations (the green markers), the landmarks (the blue markers), and the final location (the initial location again) is a graphical representation of the flightpath of the drone which your algorithm will generate. Note that the drone never leaves the confinement area (the outer rectangle) and it never enters the no-fly zones (the semi-transparent red polygons over the buildings in the centre).

Development environment

We will provide support for the use of IntelliJ IDEA Community Edition as your development environment for this project. You may already be familiar with IntelliJ IDEA from the *Informatics 1 – Object-Oriented Programming* course. IntelliJ IDEA CE is available for download from `https://www.jetbrains.com/idea/`. It is the free edition of the IntelliJ IDEA platform and supports the programming language and the build system which we use (Java and Maven respectively). Downloads of IntelliJ IDEA are available for Windows, macOS, and Linux. IntelliJ IDEA is pre-installed on DICE Ubuntu and is accessed via the command `ideaIC`.

Programming language

The programming language to be used for your software is Java. The architects of the drone delivery service have chosen Java version 14 as the version of Java which will be used throughout the project. You should ensure that the code which you submit can be compiled and run on a Java 14 installation. Java has been chosen as the development language for the service because the specifics of the hardware which will be purchased to run the service are not yet known so it is important to chose a language which is portable between different operating systems. Any libraries which you include in your project must similarly be implemented in Java for maximum portability.

—◇—

The version of the Java language which is the default on DICE Ubuntu is Java 14, specifically Java 14.0.2. If you are developing your application on a version of Java which is greater than 14 you should take care not to use language features or APIs which are not available in a Java 14 installation. You can set the Java language version to 14 in IntelliJ IDEA to prevent yourself from using APIs from higher versions of Java.

If you plan to work on your own machine and do not already have Java installed we recommend that you install Java 14.0.2 for compatibility with DICE Ubuntu. This version of Java is available for download from `https://www.oracle.com/uk/java/technologies/javase/jdk14-archive-downloads.html`. You will have to create a (free) Oracle account to download Java 14.

—◇—

We expect you to be already familiar with Java. If you are not, or if you would benefit from a refresher on Java, we recommend the textbook *Java Precisely* by Peter Sestoft, third edition published by MIT Press in 2016, as providing a concise and clear introduction to Java.[14]

Documentation for your code

Your code should contain documentation in JavaDoc syntax. This is a Java comment syntax which can be used to generate human-readable documentation in the form of HTML pages. A description of JavaDoc is online at `https://www.oracle.com/uk/technical-resources/articles/java/javadoc-tool.html`. The HTML format of your JavaDoc can be generated in IntelliJ IDEA using Tools → Generate JavaDoc ....

Project management

The build system to be used for your project is Apache Maven, a Java-based build system which manages all of the project dependencies in terms of Java libraries which you use, and enables you to build your system into a single self-contained JAR file (the *über JAR*) for deployment. This JAR file has all of your code and all of the libraries that you use together in one place. IntelliJ IDEA CE comes with Maven installed so you do not need to download Maven separately.

—◇—

The course lectures will explain use of the Maven build system; we do not expect you to be already familiar with Maven.

Using third-party software and libraries

This practical exercise allows you to use free software, but not commercial software which you must pay for or license. One free software development kit (SDK) which you should find useful is the Mapbox Java SDK which provides classes and methods for parsing and generating GeoJSON maps. Instructions for adding the Mapbox Java SDK to your project are available at `https://docs.mapbox.com/android/java/overview/`.

[14] Many Java textbooks are available so if you cannot get a copy of Java Precisely please feel free to choose another textbook. Alternatively, many tutorials on Java are available online, including the (slightly dated but still very useful) Java Tutorial from Oracle at `https://docs.oracle.com/javase/tutorial/`.

Informatics Large Practical

Coursework 1

Stephen Gilmore and Paul Jackson
School of Informatics, University of Edinburgh

1.1 Introduction

This coursework and the second coursework of the ILP are for credit; weighted 25%:75% respectively. Please now read Appendix A for information on good scholarly practice and the School's late submission policy.

—◇—

In this project you are creating a Java application which is built using the Maven build system. We will begin by using IntelliJ IDEA to create the project structure.

1.2 Getting started

If you are working on your own laptop you should begin by downloading IntelliJ IDEA, if you do not already have it. Download it from `https://www.jetbrains.com/idea/download/`. On DICE, IntelliJ IDEA is available via the `ideaIC` command.

—◇—

Next, create a new Maven project in IntelliJ IDEA by choosing File → New → Project . . . , and choosing Maven Project as the option. If you have downloaded Java 14 but not yet used it in IntelliJ then use the Project SDK dropdown and choose Add JDK . . . to add it now. This will set Java 14 as the value for Project SDK.

—◇—

Check the option "Create from archetype . . . " and choose org.apache.maven.archetypes:maven-archetype-quickstart. (There will be other archetypes in the list named quickstart; be sure to get the one which has the prefix org.apache.maven.archetypes.)

—◇—

On the next page, edit the Artifact Coordinates and fill in the options as shown below:

Group Id:	`uk.ac.ed.inf`
Artifact Id:	`ilp`
Version:	`1.0-SNAPSHOT`

On the next page leave the values as they are and click "Finish". Your project will be created.

—◇—

You should now have a working Maven project structure. Note that there are separate folders for project source and project tests. Note that there is an XML document named `pom.xml` where you can place project dependencies. Two Java files have been automatically generated for you: `App.java` and `AppTest.java`.

1.3 Setting up a source code repository

(This part of the practical is not for credit, but it strongly recommended to help to protect you against loss of work caused by a hard disk crash or other laptop fault.)

—◇—

In the Informatics Large Practical you will be creating Java source code files and Maven project resources such as XML documents which will form part of your implementation, to be submitted both here and in Coursework 2. We recommend that these resources be placed under version control in a source code repository. We recommend using the popular Git version control system and specifically, the hosting service *GitHub* (`https://github.com/`). GitHub supports both public and private repositories. You should create a *private* repository so that others cannot see your project and your code.

—◇—

Check your current Maven project into your GitHub repository. Commit your work after making any significant progress, trying to ensure that your GitHub repository always has a recent, coherent version of your project. In the event of a laptop failure or other problem, you can simply check out your project (e.g. into your DICE account) and keep working from there. You may have lost some work, but it will be a lot less than you would have lost without a source code repository. A tutorial on Git use in IntelliJ is here: `https://www.jetbrains.com/help/idea/set-up-a-git-repository.html`

1.4 The implementation task

For this coursework we will implement some fundamental Java classes and methods which will be useful also for Coursework 2. The functions which we will implement are concerned with the movement of the drone and with the delivery cost for items. Your implementation will be judged on three criteria: *correctness*, *documentation*, and *code readability*.

(a) Your implementation must have a Java package named `uk.ac.ed.inf` with a class named `LongLat` for representing a point. The constructor for this class should accept two double-precision numbers, the first of which is a longitude and the second of which is a latitude. The class should have two public double fields named `longitude` and `latitude`.

(b) The `LongLat` class should have a no-parameter method called `isConfined` which returns `true` if the point is within the drone confinement area and `false` if it is not.

(c) The `LongLat` class should have a method `distanceTo` which takes a `LongLat` object as a parameter and returns the Pythagorean distance between the two points as a value of type `double`.

(d) The `LongLat` class should have a method `closeTo` which takes a `LongLat` object as a parameter and returns `true` if the points are close to each other in the sense given on page 3 and `false` otherwise.

(e) The `LongLat` class should have a method `nextPosition` which takes an `int` angle as a parameter and returns a `LongLat` object which represents the new position of drone if it makes a move in the direction of the angle, following the definition of a move given on page 3.

(f) Your project should have a class named `Menus` in the package `uk.ac.ed.inf`. This class should have a constructor which accepts two strings which specify first the name of the machine and second the port where the web server is running.

(g) The `Menus` class should have a method `getDeliveryCost` which accepts a variable number of strings of type `String...` and returns the `int` cost in pence of having all of these items delivered by drone, including the standard delivery charge of 50p per delivery.

1.5 JUnit tests

To help you with getting these methods correct, a set of JUnit tests will be made available from the ILP LEARN course web page in the file `AppTest.java`. Place this in the `test/java/uk.ac.ed.inf` folder of your project, replacing the auto-generated `AppTest.java` file which is there already.

—◇—

Your code should pass all of the tests in this file. Before you can run these tests you must start the web server on port 9898 as described in Appendix B below.

1.6 Allocation of marks

A total of 25 marks are allocated for Coursework 1 according to the following weighting.

Correctness (15 marks): Your application should correctly implement the classes and methods described in the list above.

Documentation (5 marks): The methods which you implement should have JavaDoc comments which provide brief but clear descriptions of the purpose of the method and its return value and the role of each parameter passed to the method.

Code readability (5 marks): Your Java code should be well-structured and clear with idiomatic use of the Java language. You should consider the readability of your code, thinking that it will be passed on to the developers of the drone delivery service to extend and maintain as their needs change.

1.7 Preparing your submission

Make a compressed version of your `ilp` project folder using ZIP compression. Your `ilp` project folder is normally found in the folder `~/IdeaProjects`.

- On Linux systems use the command `zip -r ilp.zip ilp`.
- On Windows systems use Send to > Compressed (zipped) folder.
- On Mac systems use File > Compress "ilp".

You should now have a file called `ilp.zip`. In order to streamline the processing of your submissions, and help avoid lost submissions, please use exactly the filename `ilp.zip`. The archiving format to be used is ZIP only; do not submit TAR, TGZ or RAR files, or other formats.

1.8 How to submit

Ensure that you are LEARN-authenticated by visiting `http://learn.ed.ac.uk`. Go to the ILP LEARN page. Click on the *Assessment* link in the left-hand margin bar and then the link that says *Coursework 1*. Use the *Browse My Computer* option to find and upload your `ilp.zip` file. When finished, make sure that you click *Submit*.

—◇—

This submission mechanism should allow you to make multiple submissions. Later submissions will overwrite earlier ones. Submissions which arrive after the coursework deadline will be subject to the School's late submission penalties as detailed at `http://web.inf.ed.ac.uk/infweb/student-services/ito/admin/coursework-projects/late-coursework-extension-requests`. Extension Rule 1 will be applied for submissions for Coursework 1 and Coursework 2. This states that "Extensions are permitted (7 days) and Extra Time Adjustments (ETA) for extensions are permitted." The complete statement of this rule is available at the URL above.

Informatics Large Practical

Coursework 2

Stephen Gilmore and Paul Jackson
School of Informatics, University of Edinburgh

2.1 Introduction

As noted above, Coursework 1 and Coursework 2 of the ILP are for credit; weighted 25%:75% respectively. Information on good scholarly practice and the School's late submission policy is provided in Appendix A.

—◇—

You are now to extend your project from Coursework 1 into a complete implementation of the drone delivery service as described in this document. You can re-use any code which you produced for Coursework 1 including the code in the Maven XML document named `pom.xml`. You are free to edit, refactor, or delete any code from Coursework 1, keeping only the code that you find useful.

—◇—

This coursework consists of the report, the implementation, and a collection of output files written by your implementation. The code which you submit for assessment should be readable, well-structured, and thoroughly tested.

2.2 Report on your implementation

You are to submit a report documenting your project containing the following. Your report should have two sections as described below.

1. *Software architecture description.* This section provides a description of the software architecture of your application. Your application is made up of a collection of Java classes; explain why you identified *these classes* as being the right ones for your application.

2. *Drone control algorithm* This section explains the algorithm which is used by your drone to control their flight around the locations of interest and back to the start location of their flight, while avoiding all of the no-fly zones and trying to maximise the drone's score on the *sampled average percentage monetary value* metric described on page 10.

   This section of your report should contain two graphical figures (similar to Figure 6 in this document) which have been made using the `http://geojson.io` website, rendering the flights of your drone on two dates of your choosing on top of the background provided by the file `testing/all.geojson`.

The maximum page count of your project report is 10 pages, with title pages, references, appendices, and all other material included in the page total. Given the length, your report does not need a table of contents. You cannot submit reports which are over 10 pages in length, but shorter submissions will be accepted. The choice of font, font size, and margins is up to you but please consider the readability of your submission, and avoid very small font sizes and very small margin sizes. Reports of few pages tend to attract few marks; consider 10 pages to be a *goal*, as well as a limit. Your report must be in PDF format in a file named `ilp-report.pdf`.

2.3 Source code of your application

You are submitting your source code for review where your Java code will be read by a person, not a script. You should tidy up and clean up your code before submission. This is the time to remove commented-out blocks of code, tighten up visibility modifiers (e.g. turn `public` into `private` where possible), fix and remove TODOs, rename variables which have cryptic identifiers, remove unused methods or unused class fields, fix the static analysis problems which generate warnings from IntelliJ, and refactor your code as necessary. The code which you submit for assessment should be well-structured, readable and clear.

2.4 Results from your drone

In addition to submitting your source code for assessment, your submission should include 12 output files giving the results of trying your drone against the sensor visits to be made on specific days of the year (DD/MM/2022 where DD=MM). These 12 files should be at the top level of your project directory.

```
drone-01-01-2022.geojson
drone-02-02-2022.geojson
        ⋮
drone-12-12-2022.geojson
```

All of the files submitted should have been generated by the version of your application which you submit for assessment. You do not need to submit any version of the Derby database `derbyDB`.

2.5 Things to consider

- Your submitted Java code will be read and assessed by a person, not a script. It should contain helpful comments in JavaDoc format, documenting your intentions. Your submitted code should be readable and clear.
- Your code will be compiled and executed starting from a Derby database of lunch orders. It should generate and populate the database tables as described in the section of this document starting on page 11. The generated tables will be processed by a script so they must have the table names, column names, and types specified above.
- Logging statements and diagnostic print statements (using `System.out.println` and friends) are useful debugging tools. You do not need to remove them from your submitted code; it is fine for these to appear in your submission. You can write whatever you find helpful to `System.out`, but the content of your output files must be as specified above. An excessive level of logging can be counter-productive, causing the user not to read the log output, thereby defeating the purpose. Consider what should be logged, and log sparingly.
- Error messages should be written to `System.err`, not `System.out`.
- Your application should be robust. Failing with a `NullPointerException`, `ClassCastException`, `ArrayIndexOutOfBoundsException` or other run-time error will be considered a serious fault.

2.6 Allocation of marks

A total of 75 marks are allocated for Coursework 2 according to the following weighting.

Report (20 marks): You are to provide a document describing your implementation. Your document should be a clear and accurate description of your implementation as described in Section 2.2 above. The two sections of the report are equally weighted, with 10 marks for each section.

JavaDoc documentation (10 marks): You are document your Java code using the JavaDoc format. Your documentation should be informative and useful, clearly describing the classes, fields and methods of your project. Your JavaDoc code should compile to give a valid HTML document.

Implementation (30 marks): Your submission should faithfully implement the drone behaviour described above, hosted in a framework which allows the drone to make a maximum of 1500 moves on any day. Your application should be usably efficient, without significant stalls while executing. Your code should be readable and clear, making use of private values, variables and functions, and encapsulating code and data structures.

Correctness and effectiveness (15 marks): The flightpaths in the `flightpath` database table generated by your application will be tested to ensure that the moves made by the drone are legal according to the description given above, considering the drone confinement area and the no-fly zones described above. The drone's score on the *sampled average percentage monetary value* metric described on page 10 will be considered: the higher the score the better the quality of the drone.

2.7 Before you submit

You are creating a Java application which is built using the Maven build system. Follow these steps to ensure that your submission builds successfully with the Maven build system.

1. In IntelliJ, open the Maven panel in the top right-hand corner and choose the Maven Lifecycle option "package" as shown in Figure 2.7. (If you are not using IntelliJ as your IDE you can instead run the command `mvn package`, or use whatever means your IDE provides for running this command.)

   This must produce a JAR file in the `target` directory named `ilp-1.0-SNAPSHOT.jar`. Check that your JAR has this exact name. If it does not, modify your project settings or your Maven `pom.xml` file to fix this. Submissions which cannot build a runnable JAR file from the project's `pom.xml` file should anticipate losing marks for implementation correctness here.

2. Start the web server on port 80, or another port number of your choosing.

3. Start the database server on port 5217, or another port number of your choosing.

4. Run your JAR file with the command

   ```
   java -jar target/ilp-1.0-SNAPSHOT.jar 01 01 2022  80  5127
   ```

   (Replace 80 and 5127 with the appropriate number if you used different numbers when you started the servers.) This must produce an output file in the current working directory named

   - `drone-01-01-2022.geojson`

   Check that the GeoJSON output file has this exact name when the date input as a command line argument to the `java -jar` command is 01/01/2022. If it does not, modify your Java code to fix this. Submissions which write a wrongly-named file or otherwise fail to create an output file should anticipate losing marks for implementation correctness here.

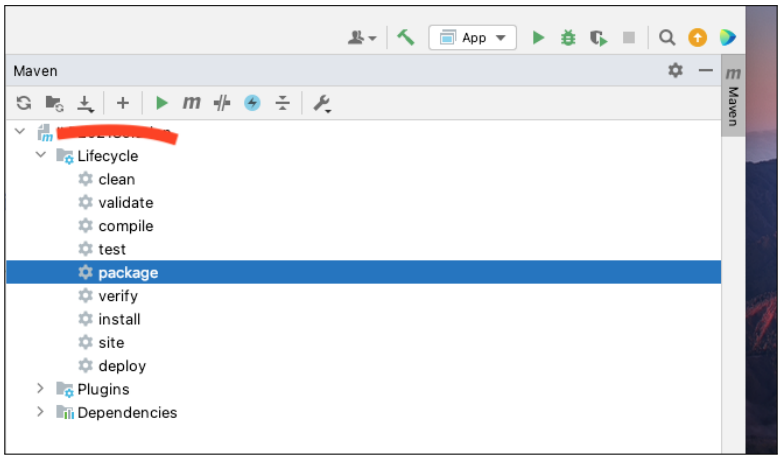

Figure 2.7: Issuing the Maven lifecycle `package` command from the Maven panel in IntelliJ. This is done to build the *über JAR* file for the project.

2.8 Running your project on DICE

(It is not compulsory that you test your project on DICE before you submit but if you wish to here are the sort of commands which you will need.)

—◇—

First, copy your *über JAR* from your own machine onto DICE with a secure copy command like

```
scp target/ilp-1.0-SNAPSHOT.jar s1234567@student.ssh.inf.ed.ac.uk:/home/s1234567
```

This will copy your *über JAR* into your home directory on DICE. Copy the webserver materials and the database materials similarly. Copy the Derby database software to your DICE account and install it. Log into your DICE account and the `student.compute` server like this.

```
ssh s1234567@student.ssh.inf.ed.ac.uk

ssh student.compute
```

(It is necessary to log on to `student.compute` because the `student.ssh` machine does not have Java.) Edit your `~/.bashrc` file to define the `DERBY_HOME` environment variable as shown on page 26.

—◇—

To avoid colliding with other users who are also running their code on `student.compute.inf.ed.ac.uk` you should choose two random numbers for the web server port and the database server port. Choose four-digit numbers beginning with 8 or 9. Say you choose 8712 and 9503.

—◇—

Run the web server in the background with

```
java -jar WebServerLite.jar /path/to/web/server/content 8712 &
```

Run the database server in the background with

```
java -jar $DERBY_HOME/lib/derbyrun.jar server start -9 9503 &
```

Finally you will be able to run your *über JAR* on `student.compute.inf.ed.ac.uk` with

```
java -jar ilp-1.0-SNAPSHOT.jar 01 01 2022 8712 9503
```

2.9 Packaging your submission

- Run your code twelve times to generate the twelve `drone-`*`DD-MM-YYYY`*`.geojson` files described above. These files must be at the top-level of your project structure.
- Make a compressed version of your `ilp` project folder using ZIP compression.
  - On Linux systems use the command `zip -r ilp.zip ilp`.
  - On Windows systems use Send to > Compressed (zipped) folder.
  - On Mac systems use File > Compress "ilp".

  You should now have a file called `ilp.zip`.

2.10 How to submit

Ensure that you are LEARN-authenticated by visiting `http://learn.ed.ac.uk`. Go to the ILP LEARN page. Click on the *Assessment* link in the left-hand margin bar and then the link that says *Coursework 2.* Use the *Browse Local Files* option to find and upload your files

- `ilp-report.pdf` and
- `ilp.zip`

In order to streamline the processing of your submissions, and help avoid lost submissions, please use exactly these filenames. The submission format for your report is PDF only; do not submit DOCX files, TXT files, Markdown files, or other formats. The archive format for compressed files is ZIP only; do not submit TAR, TGZ, or RAR files, or other formats. When finished, make sure that you click *Submit*.

—◇—

This submission mechanism should allow you to make multiple submissions. Later submissions will overwrite earlier ones. Submissions which arrive after the coursework deadline will be subject to the School's late submission penalties as detailed at `http://web.inf.ed.ac.uk/infweb/student-services/ito/admin/coursework-projects/late-coursework-extension-requests`. Extension Rule 1 will be applied for submissions for Coursework 2. This states that "Extensions are permitted (7 days) and Extra Time Adjustments (ETA) for extensions are permitted." The complete statement of this rule is available at the URL above.

Appendix A

Coursework Regulations

Good scholarly practice

Please remember the good scholarly practice requirements of the University regarding work for credit. You can find guidance at the School page:

`https://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct`

This also has links to the relevant University pages. You are required to take reasonable measures to protect your assessed work from unauthorised access. For example, if you put any such work in a source code repository then you must set access permissions appropriately, limiting access to at most yourself and members of the ILP course team.

—◇—

The Informatics Large Practical is not a group practical, so all work that you submit for assessment must be your own, or be acknowledged as coming from a publicly-available source such as Mapbox GeoJSON sample projects, answers posted on StackOverflow, or open-source projects hosted on GitHub, GitLab, BitBucket or elsewhere.

Late submission policy

It may be that due to illness or other circumstances beyond your control that you need to submit work late. Submissions which arrive after the coursework deadline will be subject to the School's late submission penalties as detailed at

`http://web.inf.ed.ac.uk/infweb/student-services/ito/admin/coursework-projects/late-coursework-extension-requests`.

Extension Rule 1 will be applied for submissions for Coursework 1 and Coursework 2. This states that "Extensions are permitted (7 days) and Extra Time Adjustments (ETA) for extensions are permitted." The complete statement of this rule is available at the URL above.

Appendix B

Running the web server

Introduction

In the description above, we made several references to content which was stored on a web server. To ensure efficient access to this web server, and to avoid problems with firewalled access to the internet, you will run the web server on the same machine that you are using to run your Java application. The web server is needed both for Coursework 1 and Coursework 2.

—◇—

Download the web server application and its content from the "Course materials" section of the ILP course web page at

`http://course.inf.ed.ac.uk/ilp`

We will use a very lightweight web server which is implemented entirely in Java and has a very small memory footprint. Unpack the ZIP file containing the web server and the web server content into a directory on your machine. You can now run the web server using the command

```
java -jar WebServerLite.jar
```

This will start the web server on the default port (80) and it will serve the content in the `menus`, `words`, and `buildings` folders. You can check the web server is running by visiting the address

`http://localhost:80`

in your preferred web browser.

—◇—

All HTTP requests are logged to the console with output like this:

```
[Tue Sep 14 19:42:34 BST 2021] ... "GET /buildings/ HTTP/1.1" 200
[Tue Sep 14 19:42:36 BST 2021] ... "GET /buildings/no-fly-zones.geojson HTTP/1.1" 200
```

If you cannot run the web server on port 80 on your machine (say, because you already have a web server running there) then you can start the web server on a different port like this:

```
java -jar WebServerLite.jar /path/to/web/server/content 9898
```

Concretely, this command might be something like

```
java -jar WebServerLite.jar /Users/irwinchay/downloads/website 9898
```

```
java -jar WebServerLite.jar /home/s1234567/Year3/ILP/website 9898
```

if your UUN is s1234567 and you stored the website content in the `Year3/ILP/website` folder in your DICE filespace. You can now check that the web server is running on your preferred port by visiting the address:

`http://localhost:9898`

in your favourite web browser and you would start your Java application with a command like

```
java -jar target/ilp-1.0-SNAPSHOT.jar 02 02 2022 9898 1527
```

passing the web server port number (9898) and the database server port number (1527) as the two last command-line arguments.

You will need to have the web server running every time that you are running your Coursework 1 or Coursework 2 code. If the web server is not running then any attempt to connect to the web server to get JSON or GeoJSON files will throw a Java exception such as

```
java.net.ConnectException: Connection refused
```

Credits

If you would like more information about the web server which we are using, you can find it at the author's web page at

```
http://www.jibble.org/jibblewebserver.php
```

together with the source code of the web server.

Appendix C

Running the database server

Introduction

In the description above, we made several references to lunch orders which were stored in a database. To ensure efficient access to this database, and to avoid problems with firewalled access to the internet, you will run the database server on the same machine that you are using to run your Java application. The database is not needed for Coursework 1, so you don't need to get to grips with it right away, but it is needed for Coursework 2.

—◇—

In this project we will be working with the Apache Derby database software version 10.15.2.0. Visit the web page at

`https://db.apache.org/derby/derby_downloads.html`

and follow the link for the Apache Derby 10.15.2.0 release. Download the zipped binary distribution in the file `db-derby-10.15.2.0-bin.zip` and store this in a folder on your machine or in your DICE filespace as appropriate. Unzip the file. We will assume here that your username is `s1234567` and that you have stored the Derby binary distribution in a folder `Year3/ILP/Derby/` on your system. Change the commands below as appropriate for your UUN and the folder where you stored the Derby binary distribution.

—◇—

We want to define the environment variable `DERBY_HOME` to refer to the location of the Derby binary distribution on your machine, or in your DICE filespace. Here are examples of the kinds of commands used to do this on various operating systems.

Caution: Do not replace the use of `/home/s1234567/` with `~/` in the commands below. Apache Derby will not expand path abbreviations such as these when they are used in `java` commands later.

- DICE (with the `bash` shell)
  - ▷ Place this command in the file `~/.brc`
    ```
    export DERBY_HOME="/home/s1234567/Year3/ILP/Derby/db-derby-10.15.2.0-bin"
    ```
  - ▷ Open a new terminal window and check that `$DERBY_HOME` is a valid filepath with
    ```
    ls $DERBY_HOME
    ```
- macOS (Catalina or later, with the `zsh` shell)
  - ▷ Place the following command in the file `~/.zshenv`
    ```
    export DERBY_HOME="/Users/s1234567/Year3/ILP/Derby/db-derby-10.15.2.0-bin"
    ```
  - ▷ Open a new terminal window and check that `$DERBY_HOME` is a valid filepath with
    ```
    ls $DERBY_HOME
    ```

- macOS (Mojave or earlier, with the `bash` shell)
  - ▷ Place the following command in the file `~/.bashrc`
    `export DERBY_HOME="/Users/s1234567/Year3/ILP/Derby/db-derby-10.15.2.0-bin"`
  - ▷ Open a new terminal window and check that `$DERBY_HOME` is a valid filepath with
    `ls $DERBY_HOME`
- Windows (varies by version)
  - ▷ Become Administrator and open System > Advanced Settings > Environment variables.
    Check details for your version of Windows using Google or your favourite search engine.
  - ▷ Via the UI, achieve the effect of
    `set DERBY_HOME=C:\User\s1234567\Year3\ILP\Derby\db-derby-10.15.2.0-bin`
  - ▷ Open a new terminal window and check that `$DERBY_HOME` is a valid filepath with
    `dir %DERBY_HOME%`

—◇—

Download the Derby database of customer orders in the file `database.zip` from the "Course materials" section of the ILP course web page at

`http://course.inf.ed.ac.uk/ilp`

Unzip this file, and make a note of the folder where you unzipped it.

—◇—

To start the database server (the Derby Network Server), change directory to the folder where you unzipped `database.zip`. Make sure that you are in the `database` folder where you can see the files `derby.log` and `derbyDB`. To start the database server on the default port 1527, issue the appropriate command below:

```
java -jar /Users/irwinchay/downloads/db-derby-10.15.2.0-bin/lib/
derbyrun.jar server start
```

- DICE / macOS
  ▷ `java -jar $DERBY_HOME/lib/derbyrun.jar server start`
- Windows
  ▷ `java -jar %DERBY_HOME%\lib\derbyrun.jar server start`

If, for some reason, you cannot run the database server on port 1527 on your machine you can specify a different port using the `-p` flag. For example, to run the database server on port 9876 you would issue these commands:

- DICE / macOS
  ▷ `java -jar $DERBY_HOME/lib/derbyrun.jar server start -p 9876`
- Windows
  ▷ `java -jar %DERBY_HOME%\lib\derbyrun.jar server start -p 9876`

When you are finished with the database server and want to shut it down use the following commands. You will need to use the `-p` flag again to specify the port if you are not using the default port.

```
java -jar /Users/irwinchay/downloads/db-derby-10.15.2.0-bin/lib/
derbyrun.jar server shutdown
```

- DICE / macOS
  ▷ `java -jar $DERBY_HOME/lib/derbyrun.jar server shutdown`
- Windows
  ▷ `java -jar %DERBY_HOME%\lib\derbyrun.jar server shutdown`

When the web server is running on port 9898 and the database server is running on port 1527 you would, as previously, start your Java application with a command like

```
java -jar target/ilp-1.0-SNAPSHOT.jar 02 02 2022 9898 1527
```

passing the web server port number (9898) and the database server port number (1527) as the two last command-line arguments. If the web server is running on a different port, or the database server is, then modify the `java` command above as appropriate.

—◇—

Remember that you will need to have the web server and the database server running every time that you are running your Coursework 2 code. We considered in Appendix B what would happen if the web server is not running; if the database server is not running then any attempt to connect to the database to read lunch orders or to write information on the flightpath of the drone will throw a Java exception such as

```
java.sql.SQLNonTransientConnectionException:
     java.net.ConnectException:
          Error connecting to server localhost on port 1,527 with message
          Connection refused (Connection refused).
```

Appendix D

Constants defined in the coursework specification

This appendix contains a summary of the constant values introduced in this document. These values will not change when the service is operational, thus they can be defined as constants of the appropriate type in your Java code.

Constant	Value	Type	Defined
Distance tolerance in degrees	0.00015	double	Page 3
Maximum number of moves a drone can make	1500	int	Page 3
Length of a drone move in degrees	0.00015	double	Page 3
Angle value when hovering	−999	int	Page 3
Appleton Tower longitude	−3.186874	double	Page 3
Appleton Tower latitude	55.944494	double	Page 3
ForrestHillLongitude	−3.192473	double	Page 4
ForrestHillLatitude	55.946233	double	Page 4
KFC longitude	−3.184319	double	Page 4
KFC latitude	55.946233	double	Page 4
Top of the Meadows longitude	−3.192473	double	Page 4
Top of the Meadows latitude	55.942617	double	Page 4
Buccleuch St bus stop longitude	−3.184319	double	Page 4
Buccleuch St bus stop latitude	55.942617	double	Page 4

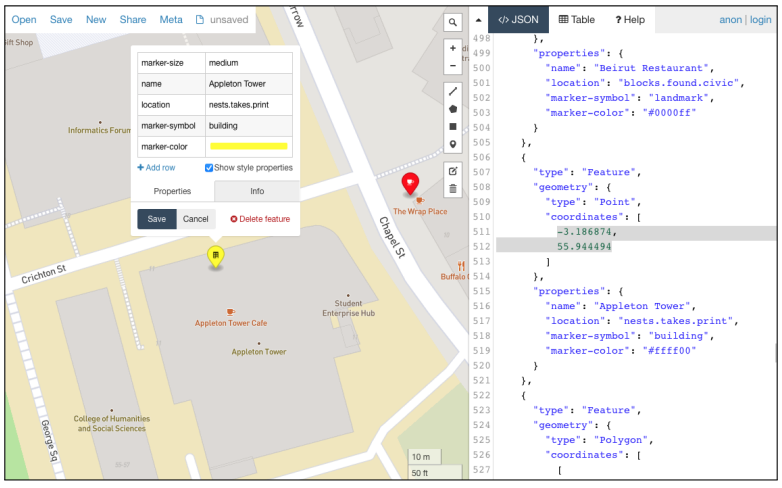

Figure 8: The location of the Appleton Tower as specified in `testing/all.geojson`

Appendix E

Using the Piazza Forum

Details

The Informatics Large Practical has a discussion forum on Piazza. You can register yourself to this forum at `https://piazza.com/ed.ac.uk/fall2021/infr0905120212ss1sem1` — please enrol with your own name, not a pseudonym or screen name. If you don't have one already, you will need to create a Piazza account.

Guidelines

Subscribing to the Informatics Large Practical Piazza forum is optional, but strongly encouraged. Questions posted to the forum may be answered by the course lecturers or by another student on the course. Please read the following notes to ensure that you have the best experience with the forum. These guidelines are based on several years of experience with course fora, where issues such as those below have arisen.

- Anonymous and pseudonymous posting on the forum is not allowed so please enrol for the course Piazza forum with your own name. Please be aware that, however they may appear to you, posts on the forum are not anonymous to the course lecturers. The forum is available only to students who are enrolled on the ILP this year. The course lecturers reserve the right to delete the enrolment of anyone who is not (or appears not to be) registered for the ILP.
- Especially when commenting on another student's work, please consider the feelings of the person receiving your message. Please refrain entirely from comments criticising the progress of another student. Each of us works at our own pace and there are many different possible orders in which to tackle the work of the ILP. Perhaps you finished implementing something in Week 4, but that does not mean that everyone did.
- If you find some content on the forum helpful, or think that it is making a useful contribution to the course, please acknowledge this by clicking "Good question" or "Good answer" as appropriate; this encourages continued participation in the forum. The course lecturers will endorse answers which they believe to be helpful.
- Forum postings which intend to correct factual errors or resolve ambiguities in the practical specification are welcome. If necessary, the course lecturers will update this coursework document to correct the error/resolve the ambiguity.
- When asking for help with fixing a run-time error, such as an exception, please include what seems to be the most relevant part of the diagnostic error message that you receive, but please include as little of your code as possible. The course lecturers may edit or delete your post if you include too much program code. For the purposes of this practical, the Maven `pom.xml` file is regarded as program code.
- The Informatics Large Practical is an individual programming project so you are not allowed to share your code with others. Please bear this in mind when answering questions on the forum; do not post your solution as an example for someone else to borrow from. *Piazza is not StackOverflow:* please do not post minimal working examples for others to copy and use.

- Many questions on Piazza tend to be of the form "Do we need to do *V* for Coursework 1?" or "Are we expected to do *W* for Coursework 2?". You already have the answers to these questions. This document, the one you are reading right now, contains the definitive statement of what is required for each coursework. It is the *coursework specification*. If this document does not say that it is necessary to do *V* for Coursework 1, or to do *W* for Coursework 2, then you do not need to do those things.
- Forum postings which ask for part of the solution to the practical are strongly discouraged. Examples in this category include questions of the form "What is the best way to implement *X*?" and "Can I have some hints on how to do *Y*?"
- Questions about the marking scheme for the practical are strongly discouraged. Examples in this category include questions of the form "Which of the following alternatives would get more marks?" and "How much detail is required for *Z*?"

Appendix F

Document version history

1.0.0 (September 22, 2021): Initial version of this document issued.

1.0.1 (September 23, 2021): Minor revision with these changes.

1. (Page 2). Clarified when orders are taken and delivered.
2. (Page 2). Clarified how recharging affects deliveries.
3. (Page 3). Corrected the definitions of longitude and latitude.
4. (Page 3). Simplified the description of Pythagorean distance.
5. (Page 3). Added a clarification of what hovering means.
6. (Page 7). Clarified how lunch orders are added to the drone.
7. (Page 21). Improved the instructions for running a JAR file on the `student.compute` server.

1.0.2 (September 28, 2021): Minor revision with these changes.

1. (Page 3). Corrected the location of Appleton Tower to (-3.186874, 55.944494).
2. (Page 29). Corrected the location of Appleton Tower to (-3.186874, 55.944494).
3. (Page 29). Added an image showing the location of Appleton Tower.