Informatics Large Practical

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

Document version 1.0.0
First issued on: September 18, 2019
Date of this revision: September 18, 2019

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 one important component 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 11th October	25	25%
Coursework 2	16:00 on Friday 6th December	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

The task for the Informatics Large Practical is to develop an autonomous ("smart") component which is to play against a human component in a location-based strategy game. The smart component is an autonomous drone which competes against a human player who is acting as the pilot of a remote-controlled ("dumb") drone which moves only in response to directions from the pilot.

— ⋄ —

The autonomous drone and the player controlling the remote-controlled drone are playing a location-based game where the goal is to collect cryptocurrency coins and power from virtual charging stations which have been distributed around the University of Edinburgh's Central Area. Your implementation of the drone should be considered to be a prototype in the sense that you should think that it is being created with the intention of passing it on to a team of developers who will maintain and develop it further. For this reason, the clarity and readability of your code is important. Given that it is based on collecting coins and power, and as we all know, money is power, the game is called *PowerGrab*.

— ⋄ —

In the PowerGrab game, the locations of the charging stations are specified on a map. A new map is released every day; each map has fifty charging stations. Cryptocurrency coins and power are collected from a charging station by flying a drone close to its location on the map, by which we mean that the drone makes a sequence of short moves to bring their location step-by-step nearer to the location of the charging station. For the purposes of the game, a player will be judged to be close enough to a charging station to be able to collect coins and power from it over-the-air if they are within 0.00025 *degrees* of the charging station. (Degrees are used as the measure of distance in the game instead of metres or kilometres to avoid unnecessary calculations.) As a simplification in the game, 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

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

Charging stations dispense coins in a fictional cryptocurrency called *Powercoin*; these real-valued coins are collected in order to increase the player's score in the game. Charging stations also dispense power which is used by the drone to power its flight. The maximum payload which a changing station can have is a credit of 125.0 coins and 125.0 units of power. However, a charging station can also store *debt* and *negative quantities of power*, which cancel out the equivalent positive value stored in the drone. A drone cannot store a negative amount of coins or a negative quantity of power; the minimum which it can have is 0.0 coins and 0.0 power.

- For example, if a user with 35.5 coins and 15.0 power comes within 0.00025 degrees of a charging station with -10.2 coins and -20.8 power then afterwards the user will have 25.3 coins and 0.0 power whereas the charging station will have 0.0 coins and -5.8 power.

Transfer of coins and power between the changing station and the drone is automatic; there is no way to prevent the transfer of coins and power from taking place if the drone is within 0.00025 degrees of the charging station. Note that the drone only connects to one charging station at a time and it always connects to the *nearest* charging station. This becomes significant when more than one charging station is in range.

— ⋄ —

The initial position of the drone is specified when play starts. At the start of the game, the drone has 0.0 coins and 250.0 units of power. Every move by the drone consumes 1.25 units of power. The game ends when the drone has made 250 moves or has run out of power, whichever comes sooner. The drone's final score is the number of coins that it has stored at the end of the game.

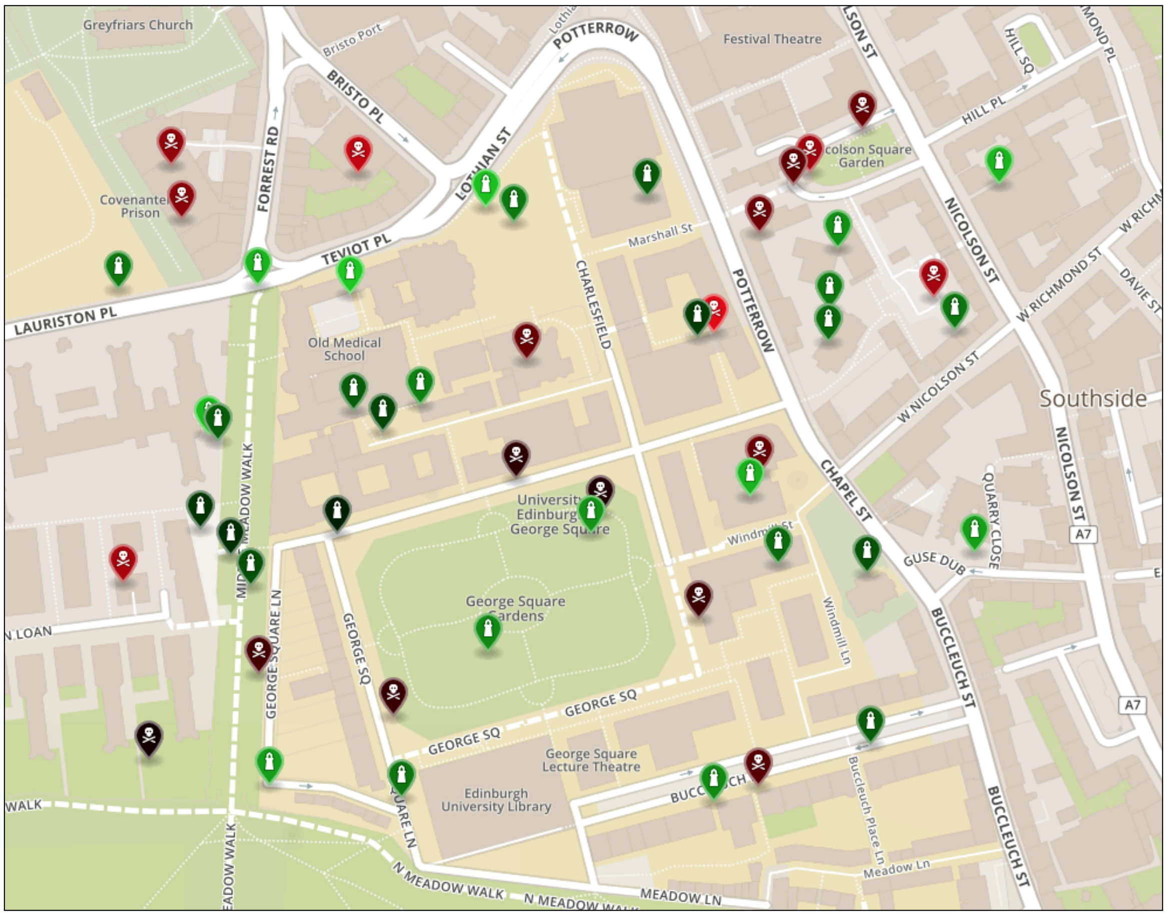


Figure 1: A *PowerGrab* map rendered by the website http://geojson.io/

—⋄—

Maps show the locations of the charging stations and also distinguish *positive value* charging stations (where both coins and power are positive) from *negative value* charging stations (where both coins and power are negative). There are no *mixed* charging stations (where one of coins or power is positive, but the other is negative).

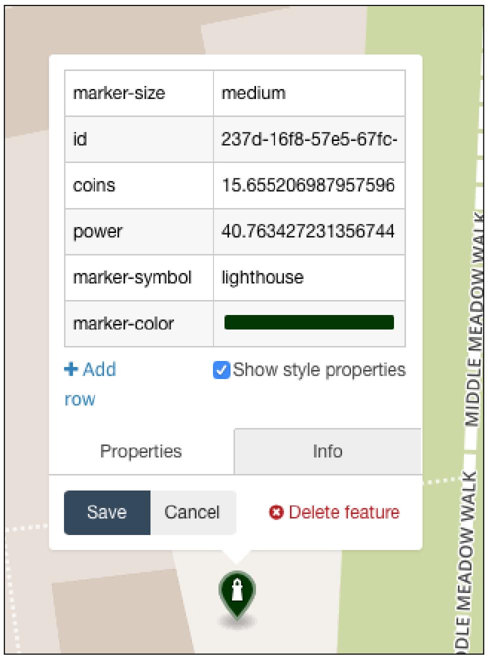
—⋄—

The use of colour (as in, green for positive, red for negative) distinguishes positive from negative charging stations. In addition, the shape of the icon (lighthouse or skull-and-crossbones) is used to distinguish positive from negative. Finally, brightness is used to differentiate stations with a lot of value (in terms of coins and charge) from stations with little value — bright green indicates strong positive value; bright red indicates strong negative value. During gameplay, the drone should try to visit positive value charging stations and try to avoid negative value charging stations.

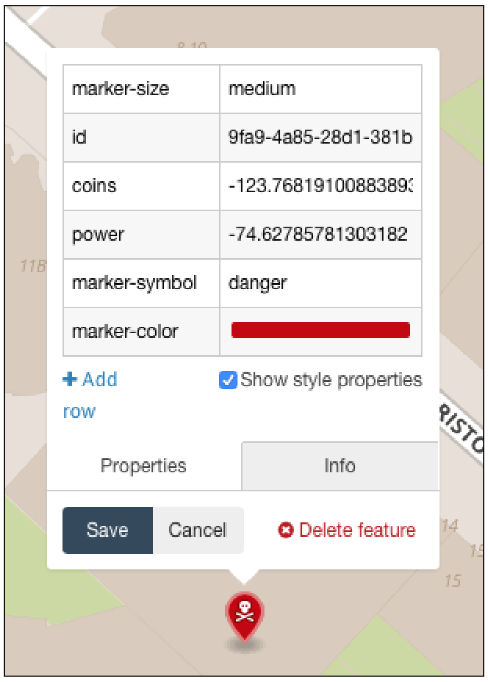
—⋄—

Maps showing the locations of the charging stations are made available as Geo-JSON documents, a JSON format which is specialised for describing places and geographical features. It is difficult for a person to interpret this JSON document directly, but we can render it with some mapping software, such as that provided by the http://geojson.io/ website. Figure 1 shows us what the map from Figure 2 looks like when rendered.

```
{
  "type": "FeatureCollection",
  "date-generated": "Tue Jan 01 2019",
  "features": [

    {
      "type": "Feature",

      "properties": {
        "id": "237d-16f8-57e5-67fc-2d3e-1ca1",
        "coins": "15.655206987957596",
        "power": "40.763427231356744",
        "marker-symbol": "lighthouse",
        "marker-color": "#003800"
      },

      "geometry": {
        "type": "Point",
        "coordinates": [
          -3.1917158579021225,
          55.94404781601724
        ]
      }

    },

    {
      "type": "Feature",

      "properties": {
        "id": "9fa9-4a85-28d1-381b-a0d2-5e75",
        "coins": "-123.76819100883893",
        "power": "-74.62785781303182",
        "marker-symbol": "danger",
        "marker-color": "#c60000"
      },

      "geometry": {
        "type": "Point",
        "coordinates": [
          -3.190260653365977,
          55.94587364601307
        ]
      }

    },
    ...
  ]
}
```



marker-size	medium
id	237d-16f8-57e5-67fc-
coins	15.655206987957596
power	40.763427231356744
marker-symbol	lighthouse
marker-color	🟩



marker-size	medium
id	9fa9-4a85-28d1-381b
coins	-123.7681910088389
power	-74.62785781303182
marker-symbol	danger
marker-color	🟥

Figure 2: A map with charging stations in Geo-JSON format.

Every map has the same format; it is a FeatureCollection containing a list of Features. Every Feature has Point geometry with two coordinates, a longitude and a latitude[1]. There will always be 50 Features in the FeatureCollection of each map, each one being a charging station.

— ◇ —

The Features in the map have properties which are metadata which tell us information about the charging station, and also markup which tells the `geojson.io` site how to render the feature as a placemark on a map. Every charging station has a 24-digit hexadecimal identifier ("id") which uniquely identifies this station.

— ◇ —

All points on every map have a latitude which lies between 55.942617 and 55.946233. All points on every map have a longitude which lies between −3.184319 and −3.192473. This defines the PowerGrab playing area as illustrated in Figure 3. *The drone must at all times remain within the playing area.*

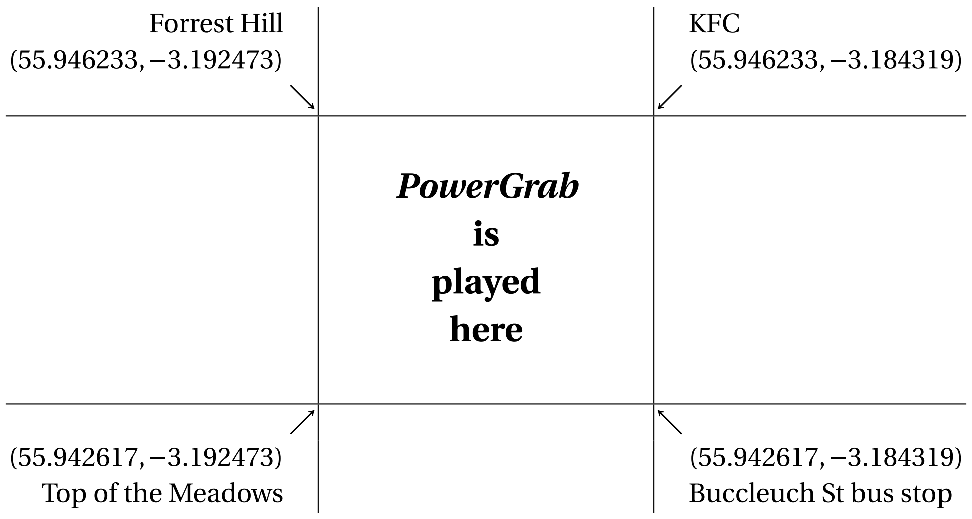


Figure 3: The PowerGrab play area

The PowerGrab maps are stored as Geo-JSON files on a web server. The maps for 2019 and 2020 are available at `http://homepages.inf.ed.ac.uk/stg/powergrab/`, under the name `powergrabmap.geojson`, indexed by year, month and day in the format YYYY/MM/DD. For example, the map for September 15, 2019 is stored at `http://homepages.inf.ed.ac.uk/stg/powergrab/2019/09/15/powergrabmap.geojson`

— ◇ —

The drone *cannot fly in an arbitrary direction*: it can only fly in one of the 16 major compass directions as seen in Figure 4. These are the primary directions North, South, East and West, and the secondary directions between those of North East, North West, South East and South West, and the tertiary directions between those of North North East, East North East, and so forth.

— ◇ —

[1] In this project, longitudes will always be negative (close to −3) and latitudes will always be positive (close to 56).

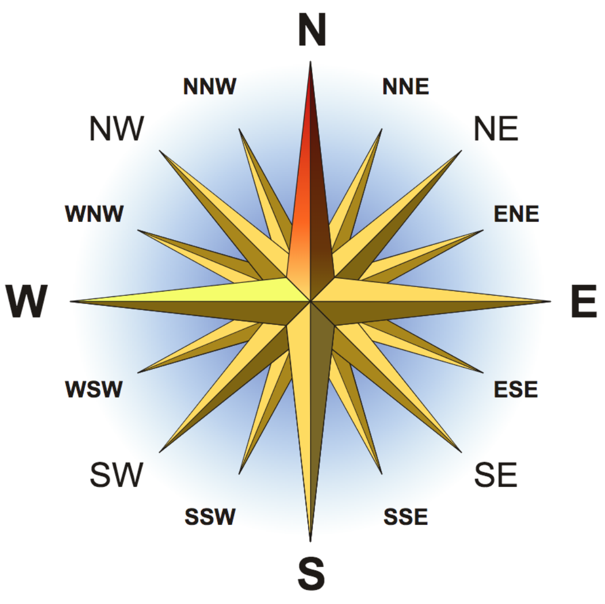


Figure 4: The 16-point compass rose. Original SVG image by I, Andrew pmk, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=2249878

As the drone flies, it travels at a constant speed and consumes power at a constant rate. In each move, the drone consumes 1.25 units of power, and moves a distance of 0.0003 degrees.

— ⋄ —

Wind speed and direction are not modelled in the game; the drone behaves in the game as though it is flying on a perfectly still day with no wind[2]. This means that the drone travels the same distance no matter which direction it is flying in. Obstacles such as trees, lampposts and buildings are also not modelled in the game; we can think that the drone is flying high enough that it flies over all the buildings in the George Square area.

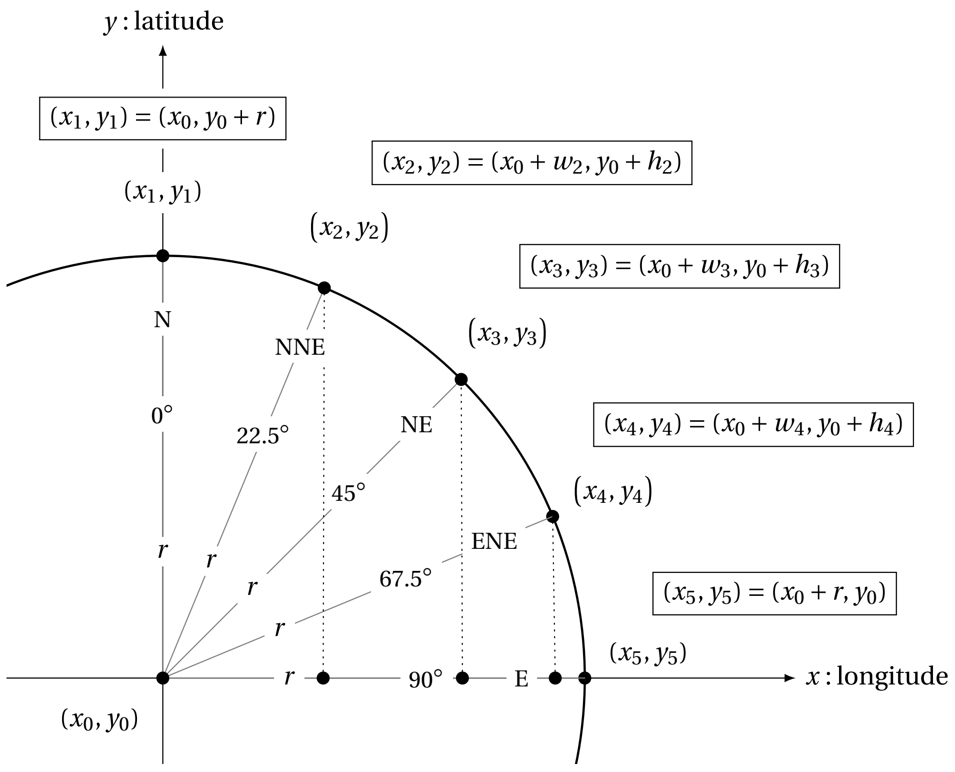
— ⋄ —

When the drone is flying, it is necessary to determine its next location from its current location and the direction of travel. Figure 5 shows the five next possible locations if the drone is travelling North, or East, or some compass direction in-between. When considering small changes of latitude or longitude, as we are doing here, it is acceptable to approximate the earth's surface as a plane. This means that, knowing that the drone travels a distance r, which is 0.0003 degrees, we can calculate the drone's next position using either simple arithmetic or simple trigonometry. We consider right-angled triangles with hypotenuse r, width w_i, and height h_i when travelling NNE, NE or ENE. To get the new longitude, we add the width of the triangle to the old longitude. To get the new latitude, we add the height of the triangle to the old latitude. The respective widths and heights as used in Figure 5 are shown below.

going NNE	w_2	$r\cos(67.5)$	h_2	$r\sin(67.5)$
going NE	w_3	$r\cos(45)$	h_3	$r\sin(45)$
going ENE	w_4	$r\cos(22.5)$	h_4	$r\sin(22.5)$

If the drone was instead heading South or West, the latitude or longitude of the drone's position would be decreasing, so we would instead *subtract* the heights and widths of similar triangles.

[2]Admittedly, there being no wind is perhaps not very realistic for Edinburgh, but it is a useful simplification of the problem.



y : latitude
$(x_1, y_1) = (x_0, y_0 + r)$
$(x_2, y_2) = (x_0 + w_2, y_0 + h_2)$
$(x_3, y_3) = (x_0 + w_3, y_0 + h_3)$
$(x_4, y_4) = (x_0 + w_4, y_0 + h_4)$
$(x_5, y_5) = (x_0 + r, y_0)$
(x_1, y_1)
(x_2, y_2)
(x_3, y_3)
(x_4, y_4)
(x_5, y_5)
(x_0, y_0)
N
NNE
NE
ENE
E
0°
22.5°
45°
67.5°
90°
r
x : longitude

Figure 5: The five next possible locations for a drone which is travelling in a north-easterly direction. Starting at (longitude, latitude) of (x_0, y_0), the next possible locations are $(x_1, y_1) \ldots (x_5, y_5)$.

— ⋄ —

Finally, the object of the game is simply to collect as many coins as possible.

The implementation task

You are to develop a simulation framework which demonstrates two versions of an automated drone which plays the PowerGrab game as though it was playing against a human player. The first version has some specific limitations and is *stateless*; this version is suitable for playing against a novice human player. The second version does not have these limitations and is *stateful*; this version is suitable for playing against an expert human player. A better implementation of the stateful drone is one which collects more coins while still keeping its runtime modest.

— ⋄ —

In the simulation of the game, each drone has 250 moves in which to grab as much power as possible and achieve the highest score that it can. The moves made by the drone, its location, and the coins and power which it currently has are reported after every move to provide a trace of the drone's play in the game.

The simulation framework

You are to develop a simulation framework which loads the PowerGrab map for a specific date and reports the first 250 moves which the drone makes when started at a specific latitude and longitude within the map. Assuming that the simulation framework is stored in `powergrab.jar` and the simulation is run with the command-line arguments:

```
    15 09 2019  55.944425 -3.188396  5678  stateless
```

then the simulator should load the PowerGrab map for 15/09/2019 and start the drone at a latitude and longitude of (55.944425, −3.188396). It initialises the pseudo-random number generator with the seed 5678 and chooses the stateless drone. (If the final command-line argument was `stateful`, then the simulator would instead choose the stateful drone.)

— ⋄ —

Your application may write any messages which it likes to the standard output stream but it should also write two text files in the current working directory. These are named *`dronetype-DD-MM-YYYY`*`.txt` and *`dronetype-DD-MM-YYYY`*`.geojson` where *`dronetype`* is either `stateless` or `stateful` depending on which version of the drone was used, and *`DD`*, *`MM`*, and *`YYYY`* are replaced by the day, month and year of the relevant PowerGrab map (for example, `stateless-15-09-2019.txt` and `stateless-15-09-2019.geojson` if these results were obtained when processing the PowerGrab map for September 15th, 2019). Please use hyphens (not underscores) in the file names and use only lowercase letters. Note that stateful is spelled with only one letter L. The content of these two files is the following.

***dronetype-DD-MM-YYYY*.txt** This file should be 250 lines long. (It can however be shorter than 250 lines if the drone runs out of power.) It contains each move of the drone in terms of the latitude and longitude of the drone before the move, the direction it chose to move, the latitude and longitude of the drone after the move, and the values of the coins and power for the drone after the move. For example, the first line of this file could be:

```
55.944425,-3.188396,SSE,55.944147836140246,-3.1882811949702905,0.0,248.75
```

This says that the drone was initially at (55.944425, −3.188396), then decided to move in a south-south-easterly (SSE) direction to (55.944147836140246, −3.1882811949702905), and after that move was completed it had 0.0V coins and 248.75 units of power. Compass directions should always be written entirely with capital letters using the abbreviations which appear in Figure 4 of this document.

***dronetype-DD-MM-YYYY*.geojson** This file is a copy of the PowerGrab map which is located at the web address `http://homepages.inf.ed.ac.uk/stg/powergrab/`*`YYYY`*`/`*`MM`*`/`*`DD`*`/powergrabmap.geojson`, with the addition of a trace of the drone's flightpath (for example, as seen in Figure 6 and Figure 7), using the method for adding lines to a map which will be outlined in the course lectures.

The stateless drone

The stateless drone is intentionally limited in order that it is a more suitable opponent for novice players of the PowerGrab game. Other than its knowledge of its position on the map, the stateless drone has no local state (i.e. the class has no fields other than a pseudo-random number generator) and hence its strategy for choosing the next move cannot depend on previous moves.

— ⋄ —

The stateless drone is thus *memoryless*, and it is limited in another way; the stateless drone only has *limited look-ahead*, meaning that its decision of the next move to make can only be based on information about the charging stations which are within range of the sixteen positions where the drone can be *after one move from its current position.* The stateless drone is not allowed to scan the entire map to decide where to go next; its decision where to move next *must be based on local knowledge* and guided by the general PowerGrab gameplay of *trying to move towards charging stations with positive value, while avoiding charging stations with negative value if possible.*

— ⋄ —

When the flightpath of a stateless drone is plotted on the map, as in Figure 6, it shows a jittery path with quite a lot of back-and-forth motion which sometimes backtracks to positions where the stateless drone has been before (because, being memoryless, it cannot remember where it has been). However, the flightpath of a stateless drone is not totally chaotic and random. We can see this because it attempts to avoid negative value charging stations (marked with a red skull-and-crossbones marker on the map) and it attempts to visit positive value charging stations (marked with a green lighthouse marker on the map).

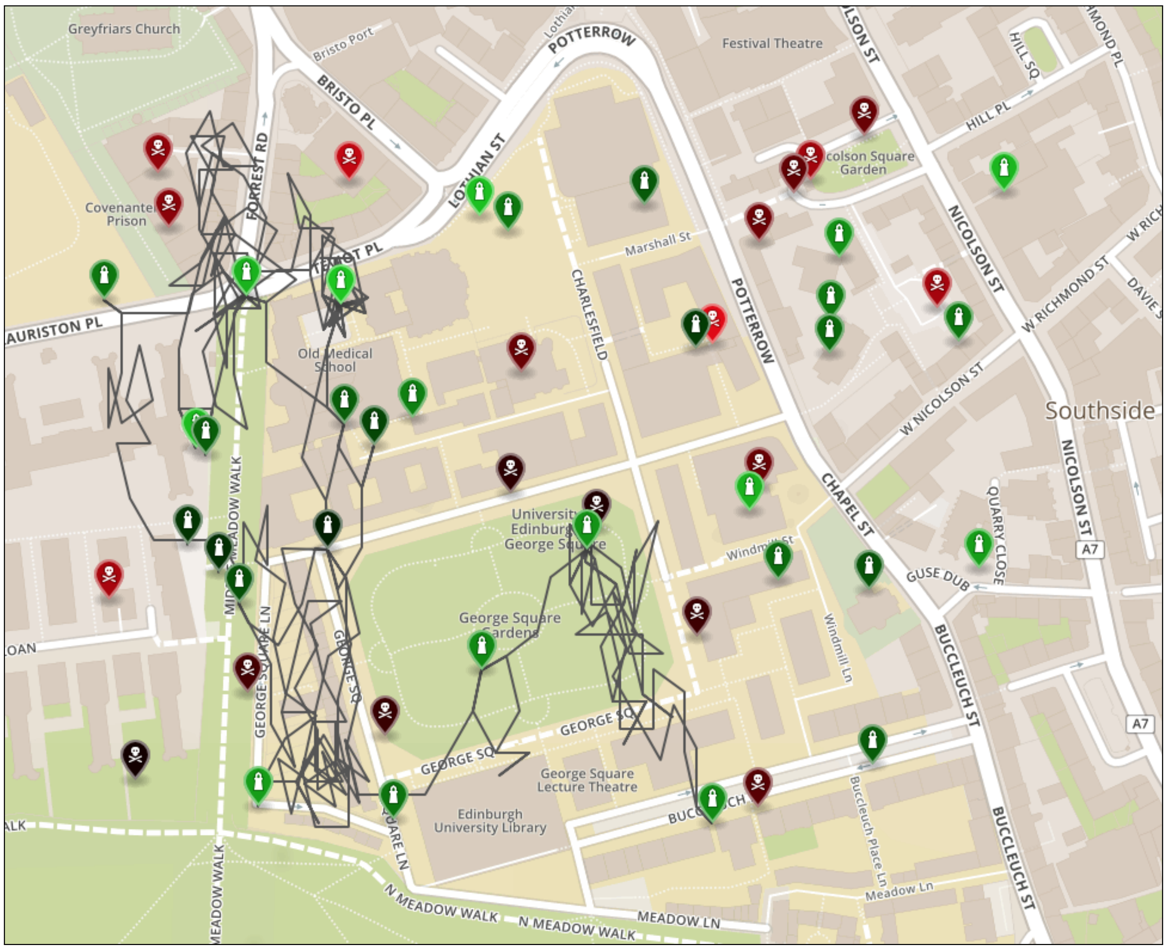


Figure 6: A *PowerGrab* map showing the (mostly random) path of a stateless drone which starts in George Square Gardens.

The stateful drone

There are no limitations placed on the stateful drone; it can use whatever strategy it likes to play against the human player of the game (here, assumed to be an experienced or expert player).

— ⋄ —

The stateful drone can remember any aspect of the earlier gameplay which it thinks will help it later. Further, it can examine the entire map before making its decision of where to move next. (There is no *limited lookahead* for the stateful drone as there was for the stateless drone.)

— ⋄ —

When the flightpath of a stateful drone is plotted on the map, as in Figure 7, it shows a much more purposeful path with much less of the back-and-forth motion that we saw with the stateless drone. The flightpath of the stateful drone shows it seeking out positive value stations (marked with a green lighthouse marker on the map) and attempting to avoid negative value stations (marked with a red skull-and-crossbones marker on the map).

— ⋄ —

When all of the positive value stations have been visited the stateful drone may randomly fly about because it doesn't know what else to do, or it might adopt a safe holding pattern away from negative value stations.

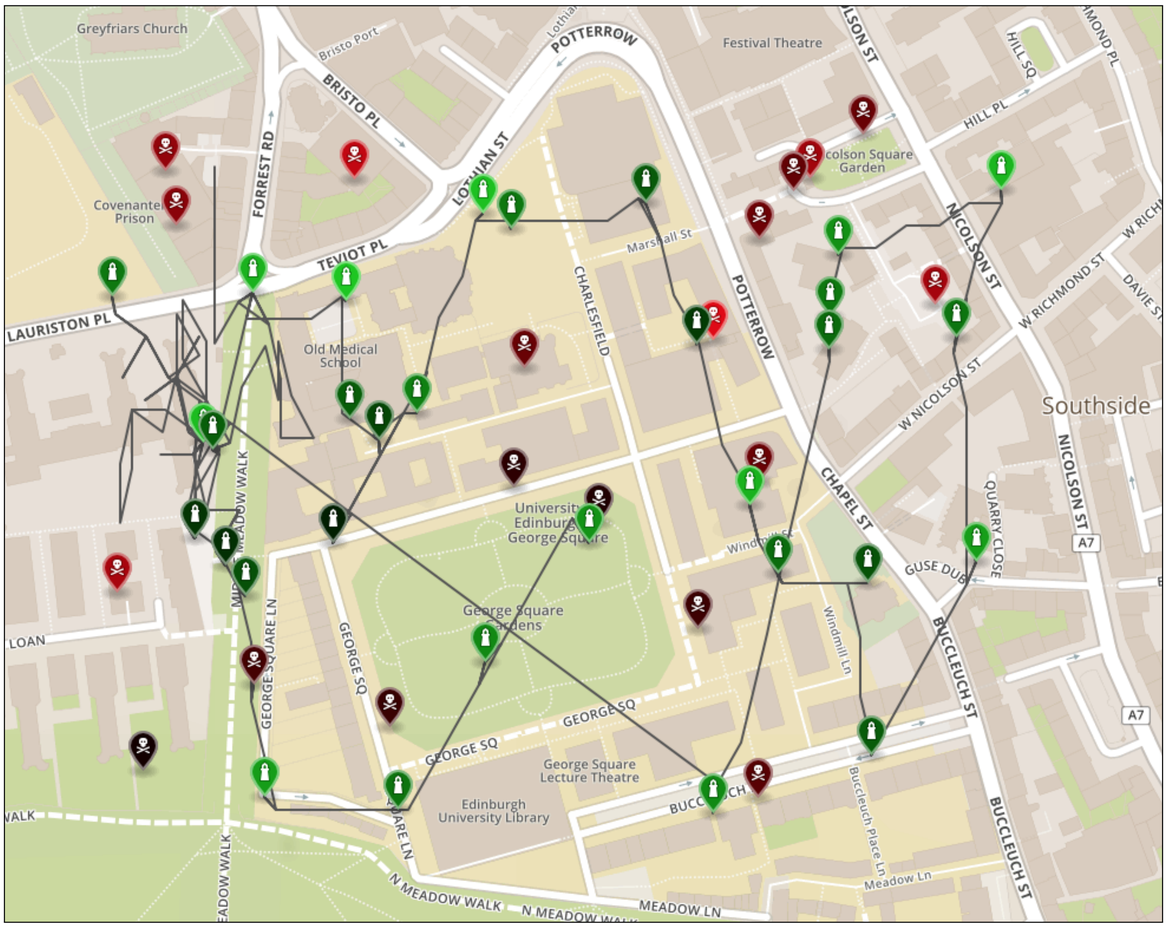


Figure 7: A *PowerGrab* map showing the (mostly directed) path of a stateful drone which starts in George Square Gardens.

Programming language

The programming language for this project is Java. At the time of writing, the version of Java which is available on Dice is 1.8.0. This means that your Java source code should be compatible with version 1.8, and run on a version 1.8 virtual machine.

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 Geo-JSON maps. Instructions for adding the Mapbox Java SDK to your project are available at `https://docs.mapbox.com/android/java/overview/`.

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 Eclipse to create the project structure.

1.2 Getting started

If you are working on your own laptop you should begin by downloading Eclipse, if you do not already have it. Download it from `https://www.eclipse.org`. On DICE, Eclipse is available via the `eclipse` command.

— ⋄ —

Next, create a new Maven project in Eclipse by choosing File → New → Project . . . , and choosing Maven Project as the option. Leave unchecked the option which reads "Create a simple project (skip archetype selection)"; this is unchecked by default. On the following page, choose the archetype "maven-archetype-quickstart" (this is the default). On the final page, fill in the options as shown below:

Group Id:	uk.ac.ed.inf
Artifact Id:	powergrab

Find "JRE System Library" in the Package Explorer and select "Properties". Change "Execution environment" to be "JavaSE-1.8".

— ⋄ —

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 Maven project resources such as XML documents and Java files which will form part of your implementation, to be submitted 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 Eclipse is here: `https://eclipsesource.com/blogs/tutorials/egit-tutorial`

1.4 The implementation task

For this first coursework of the ILP you are to implement a critical part of PowerGrab game, the function which calculates the immediate next position of the drone, given the current position and the direction of flight. Your implementation will be judged on three criteria: *correctness*, *readability*, and *efficiency*.

— ◇ —

Your implementation must have a package named `uk.ac.ed.inf.powergrab` with a class called `Direction`, which defines constants:

```
Direction.N,
Direction.S,
...
```

and so on for all sixteen compass directions used in this practical. This class does not have any other functionality.

— ◇ —

Your implementation is also to contain another class named `Position`, as outlined below:

```
package uk.ac.ed.inf.powergrab;
public class Position {
    public double latitude;
    public double longitude;

    public Position(double latitude, double longitude) { ... }

    public Position nextPosition(Direction direction) { ... }
    public boolean inPlayArea() { ... }
    ...
}
```

Your class can contain other fields, other constructors, and other methods, but it must contain those listed above, and they must be individually labelled as `public` so that they are visible outside the current package. Your submission can contain any other classes which you have implemented; these will be ignored in this coursework because our interest here is only in the class `Direction` and the class `Position`.

—◇—

The method `nextPosition` in the `Position` class is to implement the function which returns the next position of the drone when it makes a move in the specified compass direction. (Refer to Figure 5 for the specification of this function, remembering that the drone travels 0.0003 degrees in a move. I.e. $r = 0.0003$.)

—◇—

The method `inPlayArea` tests whether or not this `Position` lies in the PowerGrab play area as specified in Figure 3 of this document.

—◇—

To help you with getting the `nextPosition` and `inPlayArea` methods correct, a set of unit tests will be made available from the ILP course web page in the file `AppTest.java`. Your `nextPosition` and `inPlayArea` methods should pass all of these tests.

1.5 Preparing your submission

Make a compressed version of your `powergrab` project folder using ZIP compression.

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

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

1.6 How to submit

Please submit your implementation work from your DICE account using this command:

```
submit ilp cw1 powergrab.zip
```

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

1.7 Allocation of marks

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

Correctness (15 marks): Your function should be a correct implementation of the "next position" function for the drone, giving the correct next position for all sixteen possible compass directions.

Readability (5 marks): Your function should be clear and concise. The implementation of the `Direction` and `Position` classes should be readable and clear.

Efficiency (5 marks): The "next position" function will be executed multiple times in the PowerGrab game, and while testing the game, so it is important that it should be efficient. The game development team have not yet decided on the Java compiler which will be used to compile the release version of the game, so the compiler optimisations which will be applied are not yet known. The safest assumption in this setting is to assume that the compiler will not optimise your function at all and that you should explicitly code any optimisations which can be made to the "next position" function.

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.

— ⋄ —

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 three 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. Identify class hierarchical relationships between classes: which classes are subclasses of others?

2. *Class documentation.* Provide concise documentation for each class in your application. Explain each class as through providing documentation for a developer who will be maintaining your application in the future.

3. *Stateful drone strategy* This section explains the strategy which is used by your stateful drone to improve their score relative to the stateless drone. You should explain what is remembered in the state of the stateful drone and how this is used to improve the drone's score.

   This section of your report should contain two graphical figures (similar to Figure 6 and Figure 7 in this document) which have been made using the `http://geojson.io` website, rendering one flight of your stateless drone and one flight of your stateful drone on a PowerGrab map of your choosing. It can be any of the available PowerGrab maps, but make sure that the same map is used for both the stateless drone and the stateful drone.

— ⋄ —

The maximum page count of your project report is 15 pages, with title pages, table of contents, references, appendices, and all other material included in the page total. You cannot submit reports which are over 15 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.

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 static analysis problems, and to refactor your code as necessary. The code which you submit for assessment should be well-structured, readable and clear.

2.4 Results from the two drones

In addition to submitting your source code for assessment, you should also submit 48 output files giving the results of trying your drones against the maps on specific days of the year[3] while starting the drone at a latitude and longitude of $(55.944425, -3.188396)$. You should submit a ZIP file archive `ilp-results.zip` containing the following files:

stateless-01-01-2019.txt	stateless-01-01-2019.geojson	stateful-01-01-2019.txt	stateful-01-01-2019.geojson
stateless-02-02-2019.txt	stateless-02-02-2019.geojson	stateful-02-02-2019.txt	stateful-02-02-2019.geojson
⋮	⋮	⋮	⋮
stateless-12-12-2019.txt	stateless-12-12-2019.geojson	stateful-12-12-2019.txt	stateful-12-12-2019.geojson

All of the files named `stateless-*.*` should have been generated using the same version of your application using the stateless drone and all of the files named `stateful-*.*` should have been generated using the same version of your application using the stateful drone.

2.5 Preparing your submission

- Make a compressed version of your `powergrab` project folder using ZIP compression.
  - On Linux systems use the command `zip -r powergrab.zip powergrab`.
  - On Windows systems use Send to > Compressed (zipped) folder.
  - On Mac systems use File > Compress "powergrab".

  You should now have a file called `powergrab.zip`.
- Place your 48 output files in a folder called `ilp-results` and compress it in the same way. You should now have a folder called `ilp-results.zip`.

2.6 How to submit

Please submit your report and your implementation work from your DICE account using these commands:

```
submit ilp cw2 ilp-report.pdf
submit ilp cw2 powergrab.zip
submit ilp cw2 ilp-results.zip
```

In order to streamline the processing of your submissions, and help avoid lost submissions, please use exactly these filenames for the three submissions. The submission format for your report is PDF only; do not submit DOCX files, TXT files, or other formats.

[3]DD/MM/2019 where DD=MM.

2.7 Things to consider

- Your submitted Java code will be read and assessed by a person, not a script. It is not a waste of time to add comments to your code, documenting your intentions. Your submitted code should be readable and clear.
- Your output files `*.txt` and `*.geojson` will be processed by a script, not read by a person. Your text file must be formatted as described above, and your GeoJSON file must be a copy of the input GeoJSON file with the addition of a trace of your drone's flightpath.
- 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.
- Your application should be robust. Failing with a `NullPointerException`, `ClassCastException`, `ArrayIndexOutOfBoundsException` or other run-time error will be considered a serious fault. An exception to this is `java.net` or `java.io` exceptions which are thrown due to unavailability of the Informatics web server which hosts the PowerGrab maps. These will not be considered to be a fault with your implementation.

2.8 Allocation of marks

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

Report (30 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 three sections of the report are equally weighted, with 10 marks for each section.

Implementation (30 marks): Your submission should faithfully implement the stateless and the stateful drones of the *PowerGrab* game, hosted in a framework which allows the drone to make 250 moves against a map for a particular 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. Where it is appropriate to do so, your classes should be structured in a way which makes use of the Java class hierarchy mechanism. All else being equal, code with comments should receive a higher mark than code without comments. Everyone thinks that their code is self-documenting, but it isn't.

Output (15 marks): The output files which you submit will be tested to ensure that the moves made by the drone are legal according to the rules of the PowerGrab game, and the class of drone which is being used. The quality of the drone strategy implemented will be graded depending on the score which the drone achieves: the higher the score, the better the quality of the drone.

Appendix A

Coursework Regulations

Good scholarly practice

Please remember the University requirement as regards all assessed work for credit. Details about this can be found at:

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

Furthermore, you are required to take reasonable measures to protect your assessed work from unauthorised access. For example, if you put any such work on a public repository then you must set access permissions appropriately (generally permitting access only to yourself, or your group in the case of group practicals).

—⋄—

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 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. The School of Informatics late submission policy aligns with the university's Assessment Regulations which applies the following penalty: *5 percentage points will be deducted for every calendar day (or part thereof) it is late, up to a maximum of 7 calendar days.* However, to this University policy, the School of Informatics also adds the constraint that *late submission will only be accepted if no submission in time has been made.* For more information, see

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

The staff of the Informatics Teaching Organisation will approve extension requests up to 7 days in accordance with the "good reasons" which are defined in the University's assessment regulations.

—⋄—

Please note that things that would *not* be considered good reasons include (among other things) last-minute computer problems and loss of work through failing to backup your source code. Frequent commits of your work into a remote source code repository such as GitHub provide an off-site backup of your work which can allow you to continue from where you left off in the case of a disk or other hardware failure, so we recommend committing your work frequently.

Appendix B
The small print

Introduction

This appendix contains some details about the practical which are relevant to know, but are not perhaps the first things that you would think of to ask.

- *Webserver unavailability.* Your simulation framework begins by downloading a map from the School of Informatics web server. In the rare cases where the web server is down or overloaded it will not serve the map to your application. This will cause a Java exception to be thrown and your application will be unable to simulate the game. This will not be counted as a flaw in your application. Your application will be re-run and tested again when the web server is back up and serving maps as usual.

- *UML diagrams.* UML diagrams are a widely-used notation for describing software systems. However, this course does not teach UML diagrams and not all of the students on this course have taken a UML course previously. For this reason, there is no expectation that your documents will include UML diagrams and, consequently, there are no marks allocated to them in the marking scheme.

Appendix C

Using the Piazza Forum

Details

The Informatics Large Practical has a discussion forum on Piazza. This is available online at the address `https://piazza.com/ed.ac.uk/fall2019/infr09051ilp/home`. You can register yourself to this forum at `https://piazza.com/ed.ac.uk/fall2019/infr09051ilp`

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.

- 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 D

Document version history

1.0.0 (September 18, 2019): Initial version of this document issued.