Informatics Large Practical

Implementation report of the ‘Powergrab’ app

Alexandru Ionescu

Table of contents

1. Software architecture description

The software consists of more classes. These are Direction, Position, App (which is the main class of the project), Station, Drone, StatelessDrone and StatefulDrone. They represent important parts of our application as a whole and give a clear structure and understanding of our approach in this project.

Direction and Position classes were implemented in the beginning of the application as they are elementary parts in the process. Direction was designed to have in one place all the choices the drone has to go to at each step. On the other hand, Position contains the calculations required by drone moving in a possible direction and updating the new position based on the choice made.

App represents the main class of the project and it shows the important steps from the beginning of the project to the end such as reading the map, extracting information, running the correct type of the drone and writing in a file information required.

Station class is an important part of our software as it represents the blueprint of the object that our drone deals with throughout the game. We wanted to have all the information necessary from a station in one place to easily access it when needed and also to achieve encapsulation of data (we store only power, coins, latitude and longitude of the stations).

The Drone class represents the blueprint of the object drone and contains methods and attributes of both stateless and stateful version which are relevant for our game. It has two subclasses, each one corresponding to a version of the drone.

The StatelessDrone and the StatefulDrone classes are the subclasses of Drone and they contain different functionality of drone together with methods and attributes needed. We've used one of the primary concepts in Object Oriented Programming, which is inheritance in this case. In this way, the information is made manageable in a hierarchical order.

2. Class documentation

Direction class.

This is a special “class” in Java, an Enum, that represents a group of constants - the 16 directions that the drone is allowed to go to (N, NNE, NE, ENE, E, ESE, SE, SSE, S, SSW, SW, WSW, W, WNW, NW, NNW). The values in an enum can also be called by index.

Position class.

This class provides fields and methods of a Point for our application (we say Point to the pair of latitude and longitude that the drone is locating at). It has attributes latitude and longitude and methods nextPosition (used when drone moves on the map; it finds the new position corresponding to one out of the 16 possible directions) and inPlayArea that returns whether the drone goes out of the map or not. As the nextPosition is executed multiple times in the game, we precalculate values for different angles of sin and cos that we will need in each of the 16 directions in order to increase the efficiency of the function.

Station class.

This class contains relevant information of a station for the game. We keep in it the latitude, the longitude, the power and the number of coins of a station. We provide getters and setters to have better control of class attributes and methods. It is an important class for our application as we extensively use information of the stations throughout the game both in the case of stateless and stateful drone.

App class.

This is the main class of the application which contains reading the map and extracting relevant information from it (such as coordinates, coins, power of the stations), finding out the type of the drone and running the code corresponding to it, and writing in a file the information required for the given map and path of the drone. We have an array of Stations of 50 positions since we know that there are 50 stations on the map. In this way, we keep an index for the stations and identify them with it. We use a BufferedWriter to write the information required for the drone in a separate file.

It also contains:

- a private method readJsonFromUrl that takes a String as parameter (the url of the map) and returns the JSONObject of the text within the url
- a private method readAll that takes a Reader as parameter and returns the String of the text within the Reader by reading character with character and append it to a StringBuilder; this method is used in readJsonFromUrl
- a private method addPath that takes a String as parameter and adds a "Feature" which is of type "LineString", which contains the path of our drone; this is useful as we need to write in a separate file some information regarding the movement of the drone in the game on different maps

Drone class.

Drone class has information about the drone that is used both by stateless and stateful drone (such as coins, power, position, number of moves) and the game (charging distance, power consumed at every move). It also contains a method that is used by both versions of the drone:

- the method searchStations is responsible for searching the stations that are within charging distance at this point; it adds to the ArrayList possibleChargingStations the distance towards the stations and in the HashMap mapChargingStations the distance with the corresponding index of them

StatelessDrone class.

Stateless class is responsible for implementing the first version of the drone, which is limited in terms of memory and look ahead. It also has:

- a method getRandomWithExclusion that takes as parameters a Random, an integer for start, an integer for end and an array of integers with excluded indices, and returns a Random index between start and end without the excluded indices

- a method fillArrayNegativeOrPlayout that takes as parameters an array to search in and the number of elements that the array to return should have; we find in the parameter array the indices of the stations that are either out of the play area or have negative gain in their charging range and put them in a new array which we return

- a method tryDirections that fills an array of 16 positions with the gain that we would have in a direction once we go there; we are interested in the closest station within charging distance in order to find the gain

- a method emptyStations (it is used in stateless drone) that makes the transfer of the coins and power between the drone at this point and closest station which is in charging distance; given the rules of the game, we take into consideration the following cases:

  Suppose droneCoins stands for drone's coins. In terms of coins:

a. if the station is positive, the drone's coins increase with the station's coins and station remains empty;
b. if the station is negative and the absolute value of it is smaller than drone's coins, droneCoins decreases with the absolute value of the station and station remains empty;
c. if the station is negative and the absolute value of it is bigger or equal than drone's coins, we add droneCoins to station's coins and droneCoins becomes 0.

The same analysis goes in terms of power between drone and stations.

- a private method decideDirection that takes as parameters an int n and array of int exNegativeOrPlayout (which contains the indices of the directions that lead us to a point which is either out of play area or which provides the drone with negative gain once arrived there); this method is responsible for the decision regarding the direction that the drone should follow at a given moment and takes into consideration the following cases:

  a. not all 16 directions go into a point where we have negative gain in that region
  - i) none of the 16 directions go into a point where we have positive gain in that region, but we have at least a direction that goes into a "neutral" point so

we go random, excluding the directions that either go into a point where we have negative gain or which is out of play area

- ii) there exist at least a direction that takes us to a point where we have positive gain

b. all 16 directions go into a point where we have negative gain so we go to the one with minimum negative gain

StatefulDrone class.

Stateful class contains the implementation of the second version of the drone, which can remember any aspect of the earlier gameplay and is able to scan the entire map in order to decide where to go next. This is a crucial capability of the drone in order to be efficient. First of all, we have a private array visited that we will fill later as we arrive at the stations and a private ArrayList negativeStations in which we add the indices of the negative stations on the map. We have the function startGame in which the play of the stateful drone is created. We note the number of moves with numberMoves and power of the drone with dronePower. As long as numberMoves is under 250 and dronePower is not 0 we do the following:

Firstly, we are searching for positive stations that are not yet visited and we add the distances from this point to them to an ArrayList. We also use a HashMap from Double to Integer as we will sort the ArrayList in ascending order and we want to have a correspondence to indices of stations. We now decide what is the closest positive station that is accessible to go to (in order not to collect negative coins on the way), which was not visited already, and move in that direction. As we want to take the shortest route to the destination station, at each step we see which directions get us in a point that does not have negative stations within charging distance and then sort those by distance from that point to the destination.

This class also contains the following methods:

- a method findAvailableDirections that searches for directions that lead to a point that is in play area and does not have negative stations within charging distance; in an ArrayList possibleDirections we add distances to the points in the directions found and in a HashMap directionMap we keep the correspondence between distance and direction

- a method addNegativeStations in which we put in an ArrayList the indices of the negative stations on the map
- a method searchPositiveStations that scans the map and finds the positive stations, adding in an ArrayList positiveStations the distance to those stations and in a HashMap positiveStationsMap the correspondence between the distance and the index of each station (which can be from 0 to 49)
- a method emptyStationsStateful (used in stateful drone) that makes the transfer of the coins and power between the drone at this point and closest station which is in charging distance and was not visited yet; we tackle the same cases as mentioned above in stateless drone
- a method isFree that takes the index of a destination station as a parameter and returns true if within the charging distance of the station which has the parameter as index there are less than 3 negative stations and false otherwise; this is used in our approach for this version of the drone when we want to decide towards which station to go at a given moment
- a method hasNegatives that takes a Position as parameter and returns whether there are negative stations in the charging distance area of the point given as parameter
- a method hasArrived which takes a point of type Position and an index of a destination station and returns whether the point is within charging distance of the station with index given as parameter
- a method completeNumberMoves which makes the drone move until it gets to 250 moves when there are no stations to visit anymore; it finds the first direction which is in play area and which leads to a point with no negative stations within charging distance and the drone moves towards it

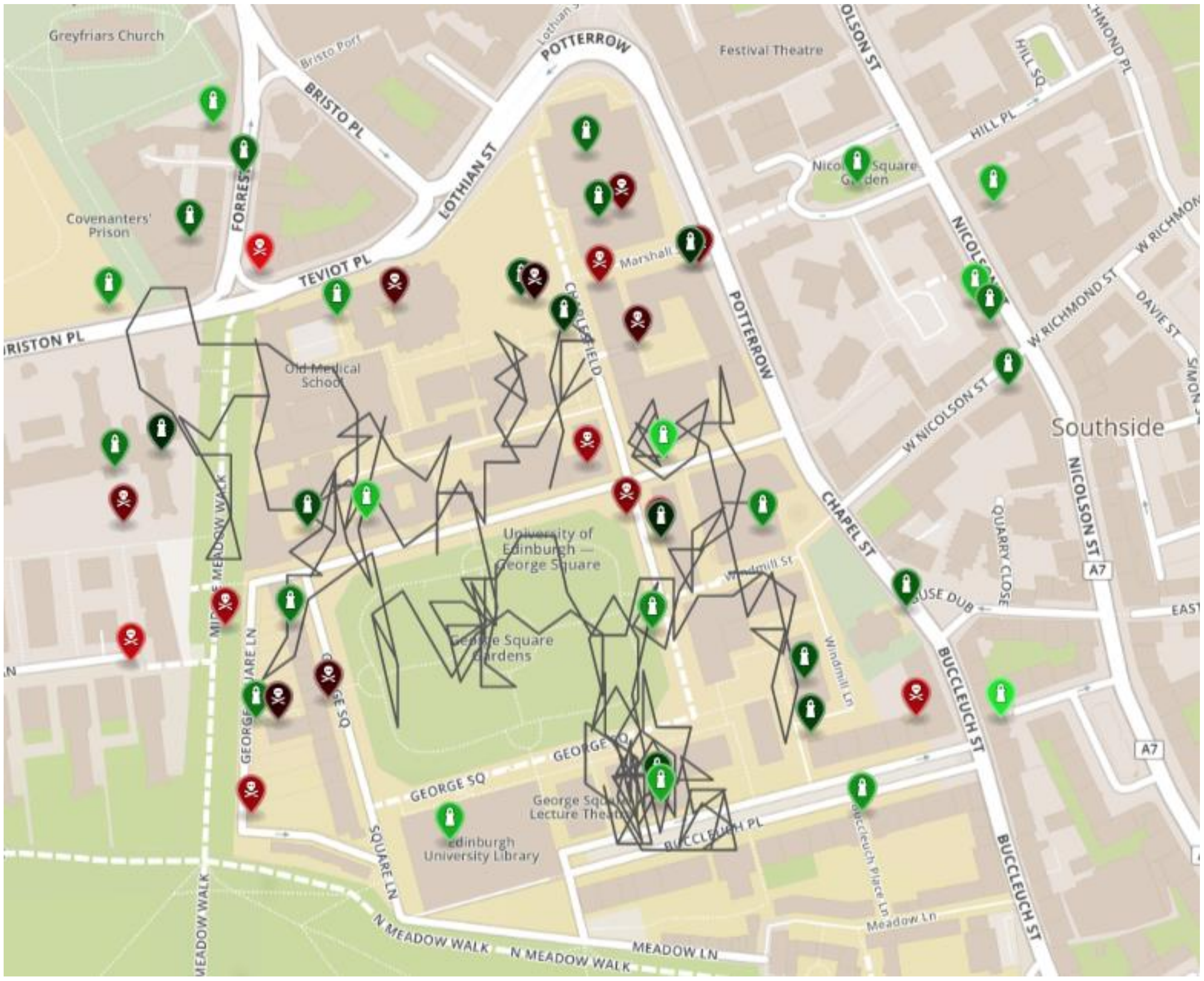
3. Stateful drone strategy

Stateful drone consistently makes use of the capability of scanning the entire map in order to decide where to go next. More precisely, at current moment, the drone figures out what is the closest positive station to go to (by calculating distances to all positive stations from this point) and if it is "free" and starts moving in that direction. We also keep track of visited stations in order not to arrive at a station twice. For that, we have an ArrayList positiveStations that we fill in with the indices of the stations that have positive coins and are not visited yet (we use an array visited for this). We also have a HashMap with distance and corresponding index that we'll use to decide what station is closest. If it is the case that there are no positive stations to take into consideration it means that we are done, that we've covered all the stations we wanted.

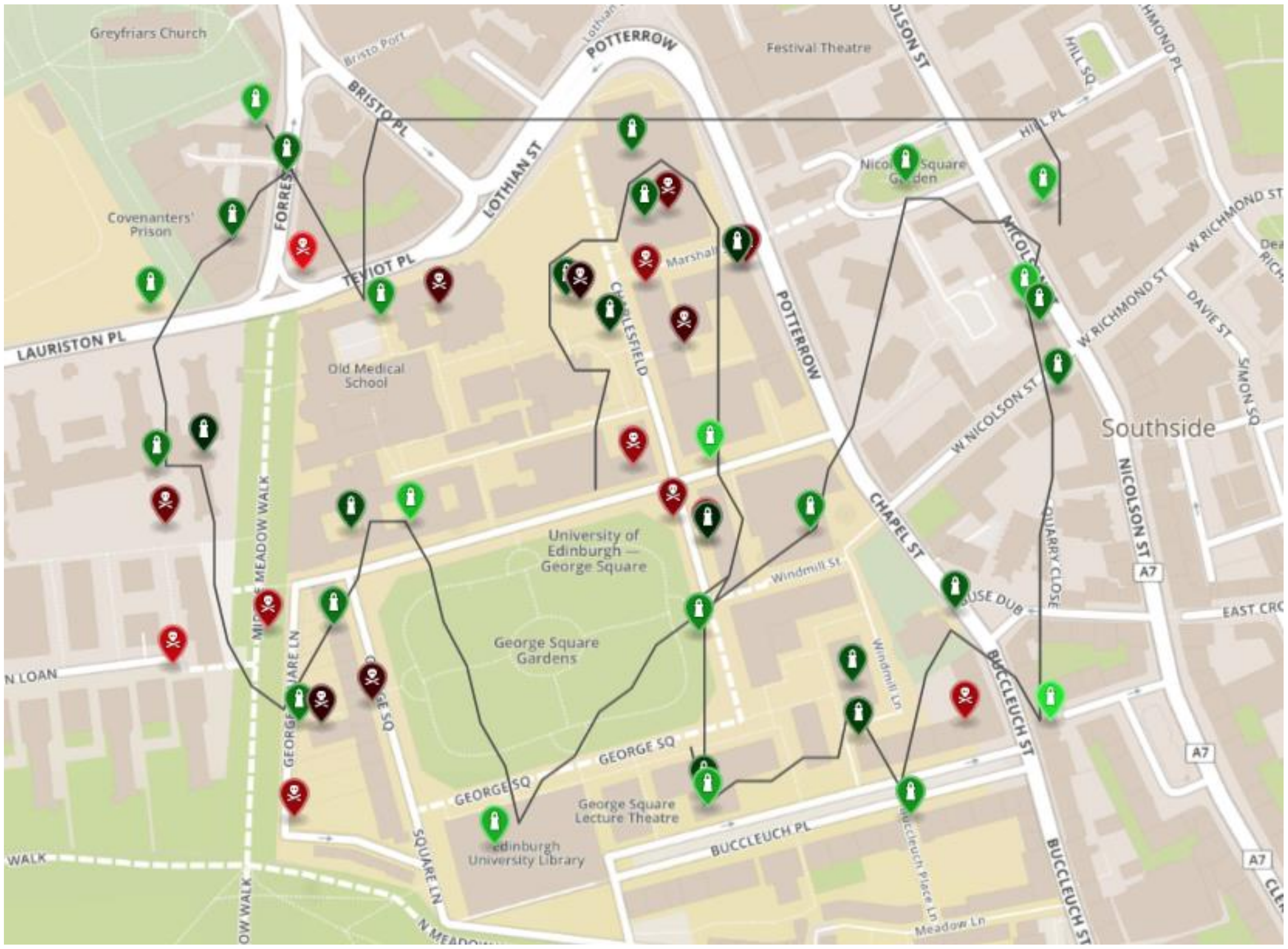
Once we decided where to go, in order to take the shortest route to the destination station and avoid negative stations on its way, we apply the following procedure. At each step we take into consideration only directions that gets us on a point that does not have negative stations within charging distance, and sort them by the euclidean distance from that point to the destination. For this, we use an ArrayList possibleDirections and a HashMap directionMap (with distance and direction). We also have the function hasNegatives that tells us if a point has negative stations within charging distance.

Then we decide to go in the direction of the shortest route out of those filtered in terms of negative stations. Of course, at each move, we update the number of moves and drone power and once we've done 250 moves we stop.

We said above that we only care about the stations that are "free". We made the following assumption. Given the nature of distribution of stations on the map, it is rarely the case that a positive station is surrounded by more than two negative stations (within charging distance). If this happens, we may not be sure that we can arrive at the station without collecting negative coins and power, as for example, the station may be in a corner of the map and three negative stations are enough to block it. We then decide to skip this positive station in order not to risk wasting power trying to arrive at it and take into consideration the next one in our list. Moreover, it might be the case that our drone, trying to find a path as mentioned above, it goes back and forth and visits the same point again and again. If we observe that, we stop by going to this destination station (and mark it as visited) and come back to recalculating what positive station is closest and repeat the process.



A PowerGrab map showing the path of the stateless drone which starts in George Square Gardens



A PowerGrab map showing the path of the stateful drone which starts in George Square Gardens

Total coin scores: stateless vs stateful

Map date	Stateless drone score	Stateful drone score	Perfect score
01/01/2019	979.2777	1880.4355	1880.4355
02/02/2019	937.1440	2109.1979	2136.1936
03/03/2019	991.8886	1928.4747	2177.0789
04/04/2019	846.3500	2274.8107	2274.8113
05/05/2019	1493.8360	1831.6849	1877.1597
06/06/2019	1258.6707	2137.4634	2137.4634
07/07/2019	984.1117	1888.3777	1903.8746
08/08/2019	850.5687	1815.7859	1815.7861
09/09/2019	1050.7031	1682.4008	1800.3646
10/10/2019	1128.4181	2301.3778	2301.3782
11/11/2019	1139.7004	1949.7308	1974.8003
12/12/2019	1240.1919	2492.0883	2492.0884

The stateful drone is trying to get close to a perfect score as often as possible. For some maps, it obtains a perfect score, meaning that it captured all the possible positive stations. Almost in all cases, it gets more than 93% of the total score, with many ocassions in which it takes at least 95%. Stateless drone obtains various percentages of the total score, as it heavily depends on the distribution of the stations on the map.

Acknowledgements

Made use of Object Oriented Programming concepts when writing code and helped by the following tutorials and books:

- https://dzone.com/articles/object-oriented-programming-concepts-with-a-system
- https://fall14cs.files.wordpress.com/2016/03/object-oriented-software-engineering-practical-software-development-using-uml-and-java-2005.pdf
- http://index-of.es/Programming/O'Reilly%20Desining%20Series/O'Reilly%20Head%20First%20Object-Oriented%20Design%20and%20Analysis.pdf

Applied Greedy algorithm as part of the application and benefited from the articles:

- https://medium.com/cracking-the-data-science-interview/greedy-algorithm-and-dynamic-programming-a8c019928405
- https://medium.com/algorithms-and-leetcode/greedy-algorithm-explained-using-leetcode-problems-80d6fee071c4

Reading JSON from a url had help of this doc:

- https://stackoverflow.com/questions/4308554/simplest-way-to-read-json-from-a-url-in-java

Learnt how to use exceptions in Java and made use of the following:

- https://www.tutorialspoint.com/java/java_exceptions.htm

Writing in a file in Java made use of the website:

- https://www.baeldung.com/java-write-to-file

Debugging the software in Eclipse had help from this:

- https://www.eclipse.org/community/eclipse_newsletter/2017/june/article1.php

Generating a random number within a range but excluding some was inspired from the post:

- https://stackoverflow.com/questions/6443176/how-can-i-generate-a-random-number-within-a-range-but-exclude-some

Understanding how Random class is used in Java had help of the website:

- https://www.journaldev.com/17111/java-random

Working with JSON in Java made use of the following:

- https://www.baeldung.com/java-org-json
- https://www.tutorialspoint.com/json/json_java_example.htm

Working with Java, understanding basic features of the language and different approaches to programming in Java helped from these resources:

- http://wavelino.coffeecup.com/pdf/EffectiveJava.pdf
- https://docs.oracle.com/javase/tutorial/
- https://www.cs.cmu.edu/afs/cs.cmu.edu/user/gchen/www/download/java/LearnJava.pdf
- http://www2.nsru.ac.th/tung/java_doc/Core%20Java%20Volume%20I-%20Fundamentals%209th%20Edition-%20Horstmann,%20Cay%20S.%20&%20Cornell,%20Gary_2013.pdf