Access Aber - Pathfinding

Final Report for CS39440 Major Project

Author: Jostein Kristiansen (jok13@aber.ac.uk)
Supervisor: Dr. Myra Scott Wilson (mxw@aber.ac.uk)

5th May 2015
Version: 1.1 (Release)

This report was submitted as partial fulfilment of a BSc degree in
Artificial Intelligence And Robotics (GH76)

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Wales, UK

Declaration of originality

In signing below, I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for plagiarism and other unfair practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the sections on unfair practice in the Students' Examinations Handbook and the relevant sections of the current Student Handbook of the Department of Computer Science.
- I understand and agree to abide by the University's regulations governing these issues.

Signature ....

Date ....

Consent to share this work

In signing below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Signature ....

Date ....

Ethics Form Application Number

The Ethics Form Application Number for this project is: 867

Student Number

120097405

Acknowledgements

I am grateful to my supervisor Dr. Myra Scott Wilson, for being of great support to me throughout the duration of this Major project, and for helping me with issues outside the scope of this project as well.

Abstract

Navigating built-up areas can be a challenge for people with reduced mobility, who can find it impossible to climb stairs, open manual doors, climb roads with a degree of inclination above a certain threshold, etc. This project addresses some of these issues by creating a pathfinding/route-planning application, with the aim to help people with reduced mobility find their way around the Aberystwyth University campuses by giving them a suitable route to follow.

Using some heuristics designed to identify and avoid certain areas, and running the A* pathfinding-algorithm on these heuristics, with positional data downloaded from OpenStreetMap [15], the application can calculate the shortest accessible path from one position on the campuses to another, and present this on a map for the user to follow.

CONTENTS

Chapter 1

Background & Objectives

1.1 Background

The Aberystwyth University campuses contain many stairs and steep inclines, and finding a route to somewhere can be challenging even when you are not in a wheelchair, as there are a lot of different buildings and alleyways on campus as well. To make the university campuses more accessible for people with reduced mobility, an application for route-planning on them is needed.

There are many route-planning applications available on the market, like Google Maps [4] or OpenStreetMap [15], but none of them address people with disabilities directly when planning routes, which is why a system like this one is needed.

This project is based in part upon a different project named ‘Access Aber’, which was developed by Aberystwyth University in the summer of 2014. In the ‘Access Aber’application, the route-planning consists of a very long list of manually plotted routes from one location on the university campuses to another, and back again. This approach is not ideal, as any changes to the infrastructure of the campuses will have to be updated in each of the routes separately. Additionally, the complete list of routes can be hard for a user to manage, due to the large number of options choose from.

As it is very impractical to manually search through a large list of routes to find one that suits your needs, a system which employs one or more pathfinding-algorithms was required to make the application easier to use. The pathfinding-algorithms will find an accessible path between the user’s current, and desired position, and return this path to the user. This way, the user only needs to know where they are, and where they want to go, and the algorithms will do all of the planning for them.

1.2 Analysis

As a key part of this project is the pathfinding-algorithm(s) employed to plan the routes, a fair bit of effort was put in to identifying which algorithm(s) best suited this particular application. The algorithm(s) would need to find the best possible route between two points, and avoid any obstacles along the way. It would also need to be fast, as the system has to plan routes in near-real-time, and use as little memory as possible, as it was meant to be implemented onto mobile devices.

A* is the chosen pathfinding-algorithm for this system. There exists other, more specialised, pathfinding-algorithms which can exhibit better performance than A* when implemented into route-planning software like this. But as A* is very easy to implement, and works in polynomial time, it has a good enough performance on a small area like the Penglais campus. If the system is to be expanded to include larger areas like the entire town of Aberystwyth however, then algorithms like the ones discussed in: [2, 5, 19] would be more appropriate than A*. A* gets slower on larger datasets, whereas the aforementioned algorithms can maintain fairly good performance even when dealing with large amounts of data.

The final list of objectives is as follows:

- Build an example database of Nodes(points on a map) for navigating the Penglais and LLanbadarn campuses.
- Extract the data relevant to pathfinding, and store connected Nodes together in Ways(Set of Nodes forming a structure, i.e. individual roads, pathways, buildings, etc.).
- Implement pathfinding-algorithm(s) for searching through the connected Nodes.
- Create heuristics for assisting the pathfinding-algorithm(s).
- Export the found routes to a map.

List of potential ambitious requirements which were not implemented due to time constraints: (Ideas for their implementation in the future can be found in section 5.2)

- Build a database of Nodes inside buildings.
- Make the system transferable to mobile devices.
- Implement some functionality to keep track of the user's movements.
- Ask Aberystwyth University for copies of their records on accessibility.
- Test the application alongside representatives for the disabled people at Aberystwyth University.

The benefits presented by the system discussed in this document over the other 'Access Aber'system, are as follows:

- dynamic route-planning.

  This has the benefit that the user only needs to specify where they are, and where they want to go; if localisation is enabled on their device, they do not even need to specify where they are, as their localisation software can do that for them. The system will build a route by looking at a database of nodes(i.e. points or locations), and find a suitable path from this data.

- The system uses external data

  As all the data used for route-planning is downloaded from a single external source [15], and this data is constantly being updated by the community: any changes to the infrastructure of the campuses can be updated in the database, and be reflected in all routes, without having to change anything inside the application itself.

- The routes can be adapted to the user's disability

  Different users are limited in different ways. A manual-wheelchair user might find it hard to climb some of the slopes that motorised wheelchairs can climb with ease for example. But as the data used for route-planning is labelled very well [8, 11], it is possible to exclude certain types of nodes from a route if this is required.

  This means that many different users can be represented, and the routes can be adapted to better reflect the user's abilities. All that is needed is to slightly modify the pathfinding for each type of disability, to avoid certain obstacles along the route.

1.3 Process

The life cycle models used for this project are Extreme Programming(XP), and Test-driven development(TDD).

Some functional requirements were decided upon at the beginning of the project, but as they were likely to change somewhat during development of the system, they were mostly written to suggest functionality, and guide the project in the right direction. Some of the functional requirements were dropped after a while, as they proved too ambitious for the time given.

When coding, unit-tests were always created before any functionality was added, making sure that only the code required to pass the tests was written at any time. This made it easier to make sure that no functionality was lost when the code was refactored, which is a an important part of XP.

Using XP and TDD, functionality could be added over time, and improved upon later. This had the advantage that a working iteration of the system was developed early on in the project, and any flaws in the system could be improved upon when they became apparent at that point. This way, unfamiliar parts of the system could successfully replace more familiar, but less efficient, and therefore temporary, parts of the system with ease after the first iteration, as all the unit tests were already in place to make this possible.

The system has been refactored so many times that the original methods are gone now, but the tests for those methods still exist, just modified and merged with other tests to fit some new functionality that has been added.

Testing has been an important part of development, and every method has one or more tests associated with it, to make sure that everything works as intended.

Chapter 2

Design

2.1 Overall Architecture

The system takes an .osm file [13] containing data about locations around the Aberystwyth University Campuses, and copies all this data into two internal databases.

One database holds information about specific points(‘Nodes’ [9] with coordinates), and the other holds information about collections of these points(‘Ways’ [12] forming a structure).

Multiple ‘Ways’can reference the same ‘Node’, forming a connection between them, but each ‘Way’will also have a set of other ‘Nodes’, some of which are not linked to any other ‘Way’.

The data in these two databases can be searched through by a pathfinding-algorithm, and a route from one ‘Node’in the database, via the connections inside the ‘Ways’, to another ‘Node’can be found.

2.1.1 Relations [10]

The .osm file downloaded from OpenStreetMap also stores something called a ‘Relation’. A ‘Relation’is an abstract collection of ‘Ways’, which form an even larger structure than a single ‘Way’does. This can be the perimeter of the town of Aberystwyth, one of the Aberystwyth University campuses, a block of houses etc.

An internal database containing a register of all these ‘Relations’has not been built, because the pathfinding-algorithm employed (A*) works best at the level of ‘Ways’, rather than collections of ‘Ways’containing collections of ‘Nodes’, which is what a ‘Relation’is.

To improve the pathfinding, algorithms like the ones found in: [2, 5, 19] could be employed, and these would be able to take into account the ‘Relations’as well; limiting the search-space, and as a result improving runtime. Those algorithms can decide upon a path by looking at areas, rather than single points on a map. i.e They can identify whether an area has entrances and exits in places that are relevant for the pathfinding before expanding any Nodes in that area, whereas A* just expands the most promising node, which can sometimes lead to a dead end.

2.1.2 map

Originally, the route found by the pathfinding-algorithm was to be exported to a map, and displayed to the user. The route is formatted in a way that makes it easy to export, but implementing the map was too time-consuming, and could unfortunately not be done in the end.

2.1.3 Mobile application

The system was meant to be able to run on mobile devices, but this has not been tested due to time-constraints. For the application to be useful for people, it should be able to run on at least one mobile operating-system with a GUI, and any positioning-systems in the device should help guide the route-planning.

2.2 Classes and methods

This system has been designed to keep related methods together in useful classes, while trying to keep unrelated classes as separate as possible; usually in a different package. This way, it is easier to find a specific part of the program, like the internal-database constructors, or pathfinding-algorithm.

For instance: The class tasked with building and storing the internal databases of ‘Nodes’and ‘Ways’from a given .osm file, builds the databases, and returns one or both databases to whichever class asks for them. There are methods for searching through the elements in the internal databases, but these have been put in a separate class.

This separation in particular makes it possible to create other classes that build similar databases (with data from GoogleMaps [3, 4] for example) at a later point, and still use the one class for searching through all the internal databases for a specific entry.

2.3 tests

All unit tests for a single class are grouped together in one test-class, and named in a manner which makes them easier to find; the tests for the SearchDatabase class are put in one test-class named SearchDatabaseTest for instance. This naming-convention is repeated for every class and test-class, and makes it easier to find any given test-class, as they all share their name with the class they are testing, and ends with ‘Test’.

There is also a separate test-class named AllTests, which will run every test-class in the system in a specified order. This is to make it easier to test the system as a whole, rather than testing individual classes one after another.

2.4 Database

Originally, the system was designed to create its own database based on input from the user. So classes were written to construct things like: buildings, floors, lifts, roads, stairs, doors, etc. But as soon as it was discovered that OpenStreetMap has a database [13] of ‘Nodes’ [9], ‘Ways’ [12], and ‘Relations’ [10] that is free to use for the public (as long as the data is used under the same licenses as they use [7]), those classes were replaced by others that were made explicitly for dealing with the data in that database instead.

Using the OpenStreetMap database instead of building a new one has several advantages.

1. It saves time, as someone has already catalogued most, if not all, of what is needed for pathfinding, and more

2. The internal database can be easier to keep up to date, as a larger community will look after the data, and update it with relevant changes.

   If the above is not wanted: the data can be downloaded, and edited privately (but license rules still apply [7])

3. The data can be more accurate, as many people will have edited it to better represent reality

2.4.1 data

All data returned by the pathfinding-algorithm are formatted as ‘Nodes’. This means that all the returned data will contain the fields that were present in the original database downloaded from OpenStreetMap [8, 11, 13]. These fields are: id(String), visible(Boolean), version(String), changeset(String), timestamp(String), user(String), uid(String), lat(Double), lon(Double) [9].

In reality, only the ‘id’, ‘visible’, ‘lat’, and ‘lon’fields are needed for navigation(There is an optional field called: alt, which is supposed to represent altitude, but this is rarely used [9]). But as the other fields do not have any significant effect on the performance of the system (apart from memory requirements), and have to be read anyway in order to read the other fields, they have been kept as part of the internal database. The other fields can however be removed in the future if this is needed, and this should not have any effect on the methods and classes currently in the system.

2.4.1.1 format

The ‘id’field is a String because, even though it consists solely of numbers, no calculations have to be made on the data, and the field is only supposed to be a unique identifier. The ‘id’is not completely unique however; a ‘Node’, ‘Way’and ‘Relation’can have the same ‘id’. So the ‘id’is only unique in its respective collection.

The‘version’, ‘changeset’, ‘timestamp’, ‘user’, and ‘uid’fields are all Strings for the same reason as ‘id’. They can contain numbers, but no calculations have to be made on the data, so they only need to be stored in a format that is easy to handle them in.

The‘visible’field is a boolean because it can only be true or false, nothing else. Because of this, it makes more sense to store it as a binary value rather than a String.

The ‘lat’and ‘lon’fields are doubles because they are values that will need to be used in calculations. Using BigDecimal() instead of double to get more accurate calculations(more decimal places) was considered, but deemed unnecessary as the accuracy of double seems to be good enough for the calculations performed in this application.

2.5 Algorithms

There are two types of algorithms that are important in this application, and choosing the correct algorithm will likely have a great effect on the performance of the system. These two types of algorithm are: Pathfinding-algorithm, and sorting-algorithm.

2.5.1 Pathfinding-algorithm

The algorithm will need to be able to find the shortest and best path from any one start-position on the university campuses, to a goal-position elsewhere.
The algorithm would also need to find a path fairly quickly, as the user will expect to be presented with a route to follow almost immediately.
To make the routes more suitable for the user, the algorithm would also need to be able to determine what constitutes an obstacle, and avoid those when planning the route.

2.5.1.1 A*(AStar)

A* has been chosen as the most suitable pathfinding-algorithm for this version of the system because it is easy to implement, fulfils all the criteria above, and is optimal and complete; meaning that it will find the shortest path to a goal if a path exists.

As mentioned earlier, there are other pathfinding-algorithms which can outperform A* greatly on large-scale navigation [2, 5, 19]. But as A* works in polynomial time, and this system only has to plan routes in a small area, it has an acceptable performance here. However, if the system is to be improved in the future, then A* should be replaced by a more specialised pathfinding-algorithm to improve performance overall.

2.5.2 Sorting-algorithm

A* depends on a few lists of items to assist in the pathfinding, and to make sure that the search is optimised. These lists should be sorted in an appropriate way.

One of the lists in this system contains all of the nodes that have not been expanded yet, and grows significantly for each step in the search, as each Node will usually be connected to many other Nodes. This list is therefore sorted in reverse order, where the Nodes closest to the goal Node is put at the back of the list, and expanded first by the pathfinding-algorithm. This way, inserting new nodes into the list will usually only result in a few entries being moved backwards, as the children of a Node will usually be about as close to the goal-Node as the Node they were expanded from(parent Node).

QuickSort was considered for sorting the lists in the system, but it has a worst-case time-complexity on nearly-sorted lists(O(n^2)), which is the kind of data it would be presented with. Insertion Sort however, has a best case performance on nearly-sorted datasets, and only has a time-complexity of O(nk) when each unsorted element in the input is k places away from its sorted position in the list.

2.5.2.1 Insertion Sort

As this algorithm is very efficient at inserting a few entries from one list into a larger sorted list, it is ideal here. Additionally, as each new entry is likely to be fairly close to the Node it was expanded from physically, it is also likely to be sorted into a position close to that node in the list; lowering the time-complexity, as the k in O(nk) is predictably small.

The algorithm sorts in-place, which means it only requires a constant amount of memory O(1) to sort a list. It removes an element from one list, and moves it to another, resulting in no extra memory being used to sort the list.

The algorithm works online, which means it is able to sort a list as it receives it, and does not require all the data to be available as the list is sorted. As new Nodes are constantly being discovered when others are expanded, the new Nodes have to be sorted into the list of unexpanded Nodes right away. This means that many of the Nodes will probably be moved around later, when more Nodes are inserted into the list. But as it is impossible to predict exactly where those Nodes might be inserted before they are found, the algorithm just has to sort the data it is given.

2.6 maps and coordinates

As the data returned by the pathfinding-algorithm is formatted like ‘Nodes’, it might be possible to use the ‘id’field as the basis for creating a route on a map from OpenStreetMap, as the ‘id’is a unique identifier in the OpenStreetMap database that the data was downloaded from [15].

Alternatively, the lat and lon fields could be translated into a geographical position, and be used directly on any map using the WGS84 Spatial Reference System(The same as GPS uses) [6], which might be more appropriate and transferable than using a unique identifier which is only readable by a OpenStreetMap map-application using the same database as this system.

As the lat and lon coordinate-fields are based on the WGS84 Spatial Reference System [6], they will need to be converted into a different format to be suitable for use on maps using other coordinate-systems [14].Which could happen, as the OSGB36 and ED50 coordinate systems are also widely used in Great Britain [16].

Again: The system does not have a map at the moment. But it should be possible to use the data returned by the pathfinding-algorithm to plot points on most maps, as long as the coordinates are converted into the correct coordinate-system.

2.7 Programming

The programming language used in this system is Java. This is both because many devices use Java, which can make it easier to release the application on mobile without changing the source-code too much, and because it was the only programming language taught to the AI & Robotics degree-scheme. So programming in any other language would have required spending additional time just to learn that language.

The IDE used in this project is Eclipse. It is said to be an excellent Java IDE, and has been used to complete Java-assignments in university for a few years now, making it a familiar system. No other alternative IDEs have been considered, as Eclipse provides all the functionality needed to complete the system.

Chapter 3

Implementation

3.1 Database

As mentioned previously, this system has been built in iterations, where each iteration has created a working version of the system, and improved upon some functionality from the previous iterations. At first, the system was designed to build its own database, and perform searches on that data. But when a better alternative was found [15], the entire system was redesigned to handle the data in that database instead.

The first iterations of the system read the .osm [13] file(database) as a text document, and all Classes and Methods were designed to read the .osm file every time a new Node was expanded. This resulted in poor performance, as the .osm file is in fact an XML-document, and should be treated as such.

All later iterations of the system use an XML-reader [18] to search through the data in the .osm file, and this has greatly improved the performance of the system. Searching for specific data takes far less time, and when the data is found, finding a field in that data [9, 12] can be done very quickly. The first iterations of the system using the XML-reader would still search through the file for a specific entry whenever a new node was expanded, but this was improved upon later.

The latest iterations of the system still use the XML-reader [18], but the data in the database is now copied into two internal databases (more specifically: List ArrayList, as it automatically increases its own size when it is filled) to make it easier to find specific data. The .osm file actually contains three separate types of data [9, 10, 12], but as mentioned earlier: the pathfinding-algorithm(A*) works best when using only two of these. So creating a third database would be unnecessary in this case.

The Ways in the database of Ways will have fields identifying what type of Way they are, but they will also have a list of all the Nodes contained in that Way. This is another reason for having two databases instead of one; as multiple Ways can contain the same Node(forming connections between Ways). To preserve memory, every Way will create references(pointers) to the Nodes it is connected to in the other database, instead of making a local copy or new Node. This makes sure that the Node actually exists before it is referenced, and that every Way will reference the same Node, ensuring that a connection is made as it should.

3.2 Search

The pathfinding-algorithm used in this system has also gone through a few iterations, and been improved upon over time. The first few iterations of the system used the Breadth-First-Search(BFS) algorithm which employed a First-In-First-Out queue, where every child of a Node was expanded before every child of those children were expanded etc. The Nodes were expanded until the desired goal-Node was found, and the search could stop.

As BFS examines every child of a Node, and does not discriminate at all, finding a path somewhere could take quite some time. The algorithm can be complete and optimal, just like A*, and find the best path if a path can be found. But it will only find the path passing through the fewest number of Nodes, which is not necessarily the most appropriate path in an environment where other factors like inclination and mobility also matter. If all nodes were the same distance away from each other, and no other factors had to be considered, then the algorithm would be optimal. But in this case it is only complete.

3.2.1 A*(AStar)

All later versions of the system have used the A* pathfinding-algorithm, which has improved efficiency greatly.

A* finds the child-nodes closest to the goal-Node, and by taking into account the distance travelled from the start-Node, expands the Node with the least travelled distance, and shortest distance to the goal-Node.

A* can also be made to discriminate against certain Nodes, meaning that it is given the ability to determine if a Node is suitable for navigation or not. The heuristics of the search will look at the Ways with connections to a Node, and depending on their type, either forward their Nodes to be part of the pathfinding, or ignore them if they are unfit for navigation. This way, the pathfinding will only be performed on Ways which the user is able to navigate, making it possible to tailor the routes to the user's needs by excluding/including more Ways in the search.

3.2.1.1 Implementation of A*

The A* pathfinding-algorithm has been implemented incorrectly in this system, and is essentially 'greedy Best-First Search'. This is due to an oversight during programming, and no functionality was added to keep track of the distance travelled from the start to another Node. This was discovered a bit too late, and there was no time to correct it.

The algorithm is complete, as it uses a list of visited Nodes to avoid infinite loops, but it is not optimal, and will finish the search as soon as the goal is found, even though a better solution might exist. The algorithm only looks at a node's distance to the goal, and does not take into account the total distance travelled from the start. This means that a sub-optimal solution can be found in some cases where the total distance travelled would have lead to the algorithm choosing a different path.

It should not be too much of a challenge to keep track of the total distance travelled, as this is just a matter of each Node copying their parent Node's distance travelled, and adding the distance from the parent to the child to this number. But to make the routes even better, the inclination of roads should be translated into a weight, and used in conjunction with the distance, which might prove a little harder.

The weight of the incline could for instance be equivalent to the length of an imaginary flat road which takes as much energy/time to travel along as the incline. This would encourage the algorithm to take detours, but only on roads that are short enough to make it worth avoiding the incline. Finding exactly how to weigh the inclines would require some more research though.

3.2.2 Ways

The search, or pathfinding/route-planning, is performed on the Nodes inside Ways, and connections between Ways can be established when the same Node is referenced within two or more Ways. As each Way usually has tags specifying its type [11], it is possible to exclude some Ways from the search straight away; unless the Way is contained within a Relation [10], in which case it will have its type specified in the Relation.

However: The Ways inside Relations do not seem to be suitable for navigation anyway, as they seem to represent abstract areas, rather than structures like roads, paths, or stairs. So excluding these Ways from the search might help with planning sensible routes for the user as well.

The ordering of Nodes inside a Way carries some significance, as they are listed in sequence; from one end of the physical structure the Way represents to the other(Or back again in some cases). So the first Node could represent the northern end of a road, the last Node could represent the southern end, and the Nodes between them would specify the shape of the road.

Keeping this in mind, certain Ways will have a Tag [11] indicating its degree of inclination. The degrees will either be based solely upon the height-difference between the first and last Node in the Way, or the percentage of inclination based on the height-difference divided by the length of the road. The degrees will be positive or negative depending on where the Way is considered to start and end. So the order in which Nodes are listed inside this Way will carry great significance in this case.

3.3 Fulfilment of requirements

Most of the items in the final list of requirements are present in the system, and even though some of the goals have not been reached, the application is able to find a route and return the coordinates to the user. Further information about the objectives that were not reached can be found in section 5.2.

- Build a database of Nodes(points on a map) for navigating the Penglais and LLanbadarn campuses.

  A database has been downloaded from OpenStreetMap, and the pathfinding has been tested on data on the Penglais campus.

- Extract the data relevant to pathfinding, and store connected Nodes together in Ways(Set of Nodes forming a structure, i.e. individual roads, pathways, buildings, etc.).

  A database has been downloaded from OpenStreetMap containing all of this data already, and two Lists have been made to copy and separate the data.

- Implement pathfinding-algorithm(s) for searching through the connected Nodes.

  A pathfinding-algorithm(greedy Best-First search) has been implemented, and can be used to find a path between Nodes.

- Create heuristics for assisting the pathfinding-algorithm(s).

  Heuristics have been made to exclude certain Ways from the search, and to identify which Nodes are closest to the goal.

- Export the planned routes to a map.

  This has not been accomplished, as it is a larger undertaking than what was first imagined.

Chapter 4

Testing

4.1 Overall Approach to Testing

Test Driven Development has been used to make sure that every method has at least one associated unit test. This has proved invaluable, as Extreme Programming requires frequent refactoring of the methods in the system, and the tests have made sure that all functionality is maintained after the methods have been modified.

Each class in the system has an associated test-class which is dedicated to testing the methods within that particular class. This makes it possible to only test the classes we are interested in, and pinpoint errors to specific methods in that class.

4.2 Automated Testing

A test-class which runs all the other test-classes has been created to make it easier to test the system as a whole, rather than testing individual classes. This test-class will run every test-class in a specified order, so that issues that are not apparent when testing a single class, can become obvious when every class is tested together, using and manipulating the same data.

Most of the tests will automatically pass or fail based on a few parameters in the tests, but some will require the user to review the output of the tests. This means that not all of the tests are fully automated, and that a little bit of manual effort has to be put in to make sure everything works as intended.

4.2.1 Unit Tests

As each class has its own dedicated test-class, and every method in the class is tested by at least one unit test in that test-class, there are quite a few unit tests in this system.

Having this many tests gives a sense of security whenever any methods have to be refactored, as the tests will fail if any functionality is missing, or results have changed after the refactoring. There is however a possibility that some unit tests are missing, or written poorly, so that methods that should fail their tests are actually passing them; resulting in a false sense of security. But having all these unit tests is better than not having them, surely.

see Appendix B for examples of unit tests.

4.2.2 Stress Testing

The system has only been tested on distances of a few hundred metres, and the algorithms have not really been given a challenging route to plan yet.

Some tests should have been written to make sure that the system cannot find Nodes that are in an inaccessible area, and the pathfinding should have been tested on a larger area than the Penglais campus, just to see how performance drops on larger-scale navigation.

4.3 Integration Testing

As there is no map in this application, it is hard to make sure that the routes planned by the system are actually usable.

There is however a top-level test-class for testing every test-class together on the same data, which makes sure that the classes are able to work as if in a real application, and successfully manipulate the same data. But without a map to show the resulting route on, it is very hard to determine whether the passing tests actually mean that the system works as intended, or if the route is impossible to follow somehow.

See Appendix B for the top-level test-class used for integration testing.

4.4 User Testing

The system was planned to be tested on representatives for the disabled people at Aberystwyth University, but as it does not work on mobile devices, and does not have a map yet, this has not been possible.

The system would need to be improved a bit more before it is ready to be tested on potential users. Implementing a map, and moving the application to mobile would be the first steps in this direction.

Chapter 5

Evaluation

5.1 Completed work

The finished system is a pretty simple route-planning application, that is able to find a route from one point on the Aberystwyth University campuses to another. The application works with any data downloaded from OpenStreetMap [13, 15], so it should also be able to find a route in areas outside the Aberystwyth University campuses as well.

5.1.1 Requirements

The requirements for the project have changed over time, as some of the initial requirements were a little too ambitious to achieve in the time given. And the final list of requirements details a fairly simple route-planning application.

The initial list of requirements included things like: a working mobile-application, utilising positioning-systems to aid in navigation, and planning routes inside/through buildings. Even though it would be great if all the initial objectives could be achieved, and the route-planning improved upon, the final system is at least somewhat functional, and only really missing a map, and a way to better specify a start and goal position.

5.1.2 Design

I believe that my design decisions were appropriate for developing a system in the time given, and I have tried to tie them up to the requirements as best as I can. I have tried to provide a reason for all of my decisions, and mentioned any alternatives I considered before choosing a particular solution.

As I have used Extreme Programming for this project, the design has evolved over time. But I think this has made it possible for me to better justify the decisions I have made, as I have tried alternative approaches to every part my system before arriving at a solution.

It is very likely that there are better alternatives to what I have done, and I would not dare to suggest otherwise. But I also think that my system is mostly okay for now, and even though it is not perfect, it really only needs to display the routes on a map before someone can use it.

5.1.3 tools

The database contained in the .osm file [13] has to be downloaded from the OpenStreetMap website [15] regularly if the system is to be kept up to date. It could also be maintained privately, as the license terms state this is okay [7], but the program will still have to abide by that license if it uses OpenStreetMap data.

I considered using data from Google Maps [4], but they seem to be a bit stricter with how their data is used [3]. So I chose to stick with OpenStreetMap, since their terms seem more reasonable, and they appear to have more detailed Nodes on the Aberystwyth University campuses(I think. But I don't know if I can prove that).

Using an external database like this was definitely the right choice for me, as I really did not have time to walk around creating my own.

5.1.4 usefulness

The system is mostly ready to be used by people, it really only needs a map, and a better way to specify a start and goal position, as you currently need to find the exact id of a Node in the database. The pathfinding-algorithm is able to find a route, and return it to the user, and the system can work with any .osm data(in the same format as I have used [13]), so the application is not restricted to plan routes solely on the Penglais campus, but can be used anywhere where OpenStreetMap data is available.

Whether the application is ready to be used by people with reduced mobility is another issue entirely. The pathfinding is currently avoiding stairs, but is not concerned about steep inclines, automatic vs manual doors, shortcuts through buildings, or any other specific accessibility concerns. So the route returned by the application might not always be appropriate for a user with reduced mobility, even though it will be fine for someone with full mobility.

5.1.5 Other project aims

As mentioned previously: The system has a working pathfinding-algorithm which can plan a route based on data from OpenStreetMap, and it is able to return this data to the user.

There is no map associated with the returned route, and the user will be presented with a set of coordinates rather than a graphical line to follow.

To specify where the route should start and where it should end, the user has to input the id of the Nodes they want to represent these two positions, rather than specify coordinates, or building names, which would be easier for them to do.

I feel like I am close to having a usable route-planning application, I just need to address the issues above before it is practical to use for anyone.

5.1.6 What I would have done differently

I have used Extreme programming for this project, and even though it has helped me greatly in some areas, it has also made it a little too easy for me to go in the wrong direction early on. So I might have chosen to use some plan-driven development if I had to start over again. But I do know a lot more about route-planning now than I did when I started, so I don't think Extreme programming was necessarily a bad choice either, as it allowed me to experiment before landing on a solution to a problem.

If I had to start this project again, I would have done a little more research into existing databases, as finding the OpenStreetMap database made a couple of weeks-worth of writing my own database-constructor obsolete. I downloaded the OpenStreetMap database very early on in development, but I didn't understand how to read it until a few weeks later, and had to rewrite pretty much everything to better fit the database. But even though it was definitely worth it to rewrite everything, it would have been better to not have to do it in the first place.

5.2 Future work

1. Implement a map to make the routes visible to the user

   The system is already returning the coordinates [6] of the Nodes in the planned route. So hopefully, exporting the route to a map would be as simple as placing these points on the map using their coordinates, and drawing a line between them

2. Build a database of Nodes inside buildings

   This would be a great addition to the system, as many routes can be shortened significantly by passing through buildings rather than going around them. The routes can also become more accessible by taking into account lifts to get to different levels

3. Make the start and goal positions easier to select

   This could include searching for a specific building(by name), or finding the Node closest to the position the user specified.

4. Transfer the system to mobile devices

   This would make the application portable, and possible to use while on the move.

5. Use positioning-systems to track the user's movements

   This would make the routes more dynamic, as the pathfinding-algorithm could be made to recalculate the route when the user goes off-track. It could also define the start position of the user automatically; only requiring the user to specify where they want to go.

6. Ask the university for copies of their records on accessibility

   This would make it possible to improve upon the internal databases, and even update the OpenStreetMap database if this is permitted by the university

7. Perform User Testing with representatives for the disabled people at Aberystwyth University
   This would make it possible to find flaws in the system, and improve it to better cater to the users' needs

Appendices

Appendix A

Third-Party Code and Libraries

The XML-Reader, and Node and Way constructors, are heavily based upon code copied from a tutorial-website [18]. Without this code, the system would likely still read the .osm file as a text document, and it would probably still read through the file every time a Node or Way had to be found.

Much of the code has been altered to better fit this application and the data found in the .osm file. But as it is still based on the code downloaded from that website [18], and the underlying structures are mostly the same, it should be referenced.

The tutorial is open content under the CC BY-NC-SA 3.0 DE license [1].
The source-code is distributed under the Eclipse Public License [17].

Appendix B

Code samples

2.1 Examples of test-classes

2.1.1 Constructor test-class

```
public class NodeTest {

Node testNode = new Node();

@Test
public void GetAndSetLatitudeShouldReturnAResult() {
assertTrue(testNode.getLatitude()==0);
testNode.setLatitude(12.34);
assertTrue(testNode.getLatitude()==12.34);
}
@Test
public void GetAndSetLongitudeShouldReturnAResult() {
assertTrue(testNode.getLongitude()==0);
testNode.setLongitude(43.21);
assertTrue(testNode.getLongitude()==43.21);
}

@Test
public void GetAndSetIdShouldReturnAResult() {
assertNull(testNode.getId());
testNode.setId("id");
assertTrue(testNode.getId().equals("id"));
}
@Test
public void GetAndSetVisibleShouldReturnAResult() {
assertNull(testNode.getVisible());
testNode.setVisible(true);
assertTrue(testNode.getVisible());
}

@Test
public void GetAndSetVersionShouldReturnAResult() {
assertNull(testNode.getVersion());
testNode.setVersion("ver");
assertTrue(testNode.getVersion().equals("ver"));
}

@Test
public void GetAndSetChangesetShouldReturnAResult() {
assertNull(testNode.getChangeset());
testNode.setChangeset("Chng");
assertTrue(testNode.getChangeset().equals("Chng"));
}

@Test
public void GetAndSetTimestampShouldReturnAResult() {
assertNull(testNode.getTimestamp());
testNode.setTimestamp("time");
assertTrue(testNode.getTimestamp().equals("time"));
}

@Test
public void GetAndSetUserShouldReturnAResult() {
assertNull(testNode.getUser());
testNode.setUser("jok13");
assertTrue(testNode.getUser().equals("jok13"));
}

@Test
public void GetAndSetUidShouldReturnAResult() {
assertNull(testNode.getUid());
testNode.setUid("uid");
assertTrue(testNode.getUid().equals("uid"));
}

@Test
public void testToString() {
assertFalse(testNode.toString().isEmpty());
}
}
```

2.1.2 Search test-class

```
public class AStarTest {

@BeforeClass
public static void PopulateLists(){
if(BuildDatabase.getNodes().isEmpty()||BuildDatabase.getWays().isEmpty()){
BuildDatabase.readConfig("map.osm");
}
}
@Test
public void shouldGetDistanceBetweenTwoPoints(){
assertTrue(AStar.distanceBetweenPoints(3.2, 2, 1, 4.1)==4.3);
assertTrue(AStar.distanceBetweenPoints(1, 4.1, 3.2, 2)==4.3);
}
@Test
public void ShouldExpandTheStartNodeAndSortTheResultingChildrenByTheirProxin
List<Node> unsortedTestNodes = new ArrayList<Node>();
List<Node> sortedTestNodes = new ArrayList<Node>();

AStar.setStartNode(SearchDatabase.searchForNode("2641099872"));
AStar.setGoalNode(SearchDatabase.searchForNode("3274334109"));

Node startNode=AStar.getStartNode();

unsortedTestNodes=AStar.getNavigatableConnectedNodes(startNode.getId());
sortedTestNodes=AStar.sortByDistance(sortedTestNodes,unsortedTestNodes);

for(int i=sortedTestNodes.size()-1;i>0;i--){
assertTrue(AStar.distanceBetweenPoints(sortedTestNodes.get(i).getLatitude(),
<=(AStar.distanceBetweenPoints(sortedTestNodes.get(i-1).getLatitude(), sorte
}
}
@Test
public void ShouldFindShortestRouteFromOneNodeToAnother(){
List<Node> path = new ArrayList<Node>();
AStar.setStartNode(SearchDatabase.searchForNode("2947828308"));
AStar.setGoalNode(SearchDatabase.searchForNode("2947828315"));
Node startNode=AStar.getStartNode();
Node goalNode=AStar.getGoalNode();

path=AStar.search(startNode, goalNode);
assertFalse(path.isEmpty());
System.out.println("start");
for(int i=path.size()-1;i>=0;i--){
System.out.println(path.get(i));
}
System.out.println("finish");
}
}
```

2.1.3 Example of top-level test-class

```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

import database.BuildDatabaseTest;
import database.FieldTest;
import database.NodeTest;
import database.SearchDatabaseTest;
import database.WayTest;
import route.AStarTest;

@RunWith(Suite.class)
@SuiteClasses({FieldTest.class, NodeTest.class, WayTest.class, AStarTest.cla
public class AllTests {

}
```

Annotated Bibliography

[1] "Cc by-nc-sa 3.0 de." [Online]. Available: http://creativecommons.org/licenses/by-nc-sa/3.0/de/deed.en

[2] A. Botea, M. Müller, and J. Schaeffer, "Near Optimal Hierarchical Path-Finding," *Journal of Game Development*, vol. 1, no. 1, pp. 7–28, 2004.

[3] Google. Google maps - terms of service. [Online]. Available: https://developers.google.com/maps/terms

[4] Google, "Google Maps," https://maps.google.com/, 2015.

[5] A. Kishimoto, A. Fukunaga, and A. Botea, "Scalable, Parallel Best-First Search for Optimal Sequential Planning," in *Proceedings of the International Conference on Automated Scheduling and Planning ICAPS-09*, Thessaloniki, Greece, 2009, pp. 201–208.

[6] National Geospatial-Intelligence Agency. Wgs84. [Online]. Available: http://earth-info.nga.mil/GandG/wgs84/

[7] OpenStreetMap Foundation. Openstreetmap license. [Online]. Available: www.openstreetmap.org/copyright

[8] ——. Osm - features. [Online]. Available: http://wiki.openstreetmap.org/wiki/Map_Features

[9] ——. Osm - nodes. [Online]. Available: http://wiki.openstreetmap.org/wiki/Node

[10] ——. Osm - relations. [Online]. Available: http://wiki.openstreetmap.org/wiki/Relation

[11] ——. Osm - tags. [Online]. Available: http://wiki.openstreetmap.org/wiki/Tags

[12] ——. Osm - ways. [Online]. Available: http://wiki.openstreetmap.org/wiki/Way

[13] ——. Osm-xml file format. [Online]. Available: http://wiki.openstreetmap.org/wiki/osm_xml

[14] ——. Wgs84 conversion in openstreetmap. [Online]. Available: wiki.openstreetmap.org/wiki/Converting_to_WGS84

[15] ——, "OpenStreetMap Homepage," http://wiki.osmfoundation.org/wiki/Main_Page, 2004.

[16] Ordnance Survey. A guide to coordinate systems in great britain. [Online]. Available: http://www.ordnancesurvey.co.uk/docs/support/guide-coordinate-systems-great-britain.pdf

[17] The Eclipse Foundation, "Eclipse public license - v 1.0," www.eclipse.org/legal/epl-v10.html.

[18] L. Vogel. Java and xml - tutorial. [Online]. Available: www.vogella.com/tutorials/JavaXML/article.html

[19] S. Yang and A. K. Mackworth, "Hierarchical shortest pathfinding applied to route-planning for wheelchair users," in *Proceedings of the Canadian Conference on Artificial Intelligence, CanadianAI 2007*, Montreal, PQ, May 2007, pp. 539–550.