Adapting Access Aber making use of native Android functionality.

Final Report for CS39440 Major Project

Author: Thomas Keogh (thk11@aber.ac.uk)
Supervisor: Dr. Myra Wilson (mxw@aber.ac.uk)

17th April 2015
Version: 1.1 (Draft)

This report was submitted as partial fulfilment of a BSc degree in Computer Science (G400)

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 .....

Acknowledgements

I am grateful to...

I’d like to thank...

Abstract

The aim for this project was to provide an alternative android native application to the current Access Aber web application, while also improving on the groundwork laid out by the initial work. This was done due to the help it can provide to users who are visiting the University as well as providing at least some assistance to those with disabilities. Aberystwyths campus can be a hard place to navigate and the benefits provided from a mobile device can benefit users to a large degree. The application was developed for the android OS using the Android Studio IDE and relies heavily upon the Google Maps API. It provides a range of services to the user, some of which were brought over from the existing application and improved, some new. These include route finding around campus, user refined displays of building locations, route plotter with application compatible outputs and a location based help system. This dissertation covers the original analysis, requirements, design and implementation of the project as well as details on the testing and design decision made throughout the development. The conclusion is an android application which met its original requirements specification and adds further complexity to the initial web version of the project. This includes route grading and the representation of this along with an extendable solution to route finding using Google Maps and graph searching techniques.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Background & Objectives

1.1 Background

1.1.1 Background and Preparation

Background research was compromised of analysing existing systems like the original Access Aber, previous work completed by the developer that used map technology within Android and general research into the map technology to gather a strong understanding of exactly what was possible.

A majority of time was spent analysing Access Aber and the currently existing criticisms of it, a lot of the problems were clear without the user feedback that had been provided but some criticisms were more subtle but easy to understand like how the application provides no main menu as such causing the user to feel lost from the start. It was also obvious that at a technical level, the included features were not overly complex, at least not to complete on the Android platform. Due to the application originally being developed on a system which allowed it to be run both on iOS and Android there probably were limitations from both sides which leads to having to develop around the weak elements in both platforms. As stated developing for a single platform should help us avoid these problems while also giving us the ability to take advantage of what is there. Due to this a large amount of research was completed into both UI design within Android as well as the technical benefits both Open Street Maps and Google Maps could provide the development.

Research was also completed into exactly how the mesh of locations would be created and represented, within the design specification it was concluded that a searchable graph was a possible solution and one worth researching further. This also meant analysing how other existing applications had mapped roads to graphs and the general area of map representation. While this gave a fair amount of answers further decisions needed to be made on both how to search the constructed graph and how to make it an extendible system for future additions. This was one of the main conditions brought up in meetings with the original 'customers' of the Access Aber application, it had to be functional after the initial developers had left. It was also described that it would be beneficial for the application to be developed in such a way that a possible future development that involved a central file store of the information for ranging versions of the application. Due to this further research was done on how we could leave the application in such a way that it facilitated the addition of this feature possibly without using it upon the current cycles completion.

Past work by the developer was also analysed for anything of use that may come out of it, the application analysed was proven to have functional working code and as such was a place where possible solutions could be found. This included implementation of maps in an Android environment, the logging of a route which is something that has been previously outlined as a key requirement and a fair amount of small features relating the monitoring of a users location and the information which can be gathered from that. Most information gathered from this however was fairly irrelevant, it was decided it was not the best place for referencing in future.

Finally the Android API and several Android libraries were examined for the benefits they could provide the development with, along with the search for several features that had already been selected for inclusion within the application. This ranged from simple research on Expandable List Views to research on the best way to present the application in an intuitive way, a lot of factors were gathered from the original feedback which guided a lot of the research. While a slide function for the screens was visually appealing the possibility of it confusing users was something that removed it as a possibility very quickly. Research was also performed into Google's Material Design program, a set of guidelines for good design within the Android environment. It contained a large amount of information relating to designing professional and good looking applications, while the information was too much to be fully applied in the projects time line some key ideas were taken from it relating to the design of elements within an application and general theme layout. Furthermore some sites were analysed for basic colour themes and how to implement a simplistic design without it looking unprofessional.

1.1.2 Interests

This project has been chosen due to a variety of reasons, revolving around both the developers interests and the possibility of the project leading to something that can have real world influences. The main source of interest is that the project allows for a development environment that takes input from the real world, mobile devices are of a great deal of interest to the developer due to the data that can be gathered from them and the manipulation of this data. Mobile devices are steadily changing the way we live, with applications that not only help us find where to go but tell us what we can do when we get there. Where other people have liked and even what is best suited to us based on past choices we have made and the device has noted. While some systems outside of mobile development clearly have massive real world influences it is felt by the developer that the easy entry and possibilities provided with mobile development make it one of the most accessible and revolutionary platforms.

With the application in concern providing information to the user it also means we can filter what information is shown based on user details. If the user says i cannot manoeuvre stairs for whatever reason, we could implement a function to provide paths with no stairs, this could obviously be expanded to include searching based on a range of filters. This could also be condensed down to a single graph, with different links becoming available with different information. This kind of possibility is what makes mobile such an interesting platform, not only can we show a map and a route through it, we can then guide a user through it, even ask them for information regarding it once they are done. Possibilities provided are wide and interesting, with so many choices available the future of the application is impossible to predict because of the many paths it could possibly follow.

Another source of interest is the real effect this application could have for an organisation such as Aberystwyth University, the confusing and difficult to traverse campus is not a problem

that can be resolved easily. With everything on campus set in stone it is simply not possible to move buildings and locations around. However with technology we can facilitate the use of campus without any physical additions to it at all. This means we can change the physical use of something with technology that does not itself directly effect or control the environment it is deployed in. An interesting possibility would be to examine users manoeuvring of campus before the application is introduced and afterwards, as well as the effect it has on how comfortable disabled users feel on campus. With the application very literally changing the way a user traverses the campus it is possible that some of the more unused walkways could become more travelled due to their inclusion in the application. Further research will have to be done on most effective travel paths around campus, it is likely that the ones currently most travelled are the quickest but that is not definite.

1.2 Analysis

A broad analysis of the problem with the currently existing Access Aber application is that while it provides solutions, they are long, problematic and make the user follow a convoluted process that may not be obvious to the common person, the exact person we are aiming to help with the application. A prime example of the problems posed is in the route finding within the application, it exists but requires the user to scroll through a huge list of possible routes and then takes them to a summary screen before finally letting the user see the route they need to walk. There are some very obvious, stand out ways, that this can be improved including the implementation of simple features like drop-down boxes for destinations and the removal of the summary screen.

Most of the problems in the application come from simple solutions to more complex problems, simple in the way that they do not fully cover the problem or provide a real benefit to the system. This appears throughout the application and based on the information provided seems to have been caused by the short time period it was developed in. One aim for this project is to provide a much more polished result with each problem solved properly rather than meeting requirements with a 'it will suffice' mentality.

Access Aber's initial form will be treated as a kind of groundwork for development, while no code will actually be taken from it the features will be adapted and built upon. This way we can build not only a port of the application to Android we can build a better version which in turn can act as ground work for any future work on the project, be it with the Android application or another new version on another platform. Having this groundwork and the feedback on it means that the some of the problems that need fixing are already highlighted for us, while the time line of the project may not be long enough for some of the features to be fixed most can at least be addressed.

A step by step guide which details various information such as major features and activities can be seen in the Design Specification (Section 2). The following requirements are what has been decided as key to the success of the application, they have been selected due to either their use to the user or the fact that they have always been included in the project and these aim to be improved versions.

1.2.1 Breakdown of Requirements

After much analysis the problem has been broken down into a set of requirements, some of the process to arriving at this point can be found in both the Requirements specification with further information in the Design Specification, however the reasoning will be fully discussed here. Each requirement will serve as a benchmark for the project, requirements were set as realistically as possible considering the time given and the work that needs to be completed.

1.2.1.1 Route Finding

Route finding is seen as by far the major feature within the application, not only is it the most technologically advanced it is also the most useful for our prospective users. As detailed, a graph system had been highlighted as the best way to achieve a representation of campus for us to apply a search algorithm to, with Breadth First Search (BFS from now) seen as a possible solution for the time being. This feature also has some planned extensions to it, firstly the path shown to the user should contain extra information pertaining to the path. This includes information about the difficulty of the route and features along the route such as stairs, it would be beneficial to have information for separate segments beyond their rating but this is a stretch goal and not something should realistically be expected. Due to the expected structure of the graph the implementation of displaying rating of segments within the paths should be attainable, with large routes being made up of many links between nodes it is almost to be expected in a system such as this.

It is key that the route finding solution be easily expandable, this is due to the maintainability of the application being key to its success. There is no guarantee that the original developer will be around for future expanding if the application is deemed fit for purpose and further developed, due to this it must be possible for the adding of new locations to be possible. An ideal solution would use a central file store to pull location information from, however the problem here which has been previously covered is that there is no guarantee the user has access to internet when using the application. Due to this a sufficient solution should see that the extending of route finding should be achievable by a novice programmer, ideally a common user if they are provided with clear instructions. A set of guidelines should be produced if the project is taken further.

A further stretch goal would be an implementation of route finding based on a sort of filter, whether this be a routes rating or features on the route. A current rough idea of this can be found in the Design Specification (2.1.1), this highlights the idea of using a separate graph altogether, this way if we make the search algorithm so that all it needs is a set of files in a set way we can have a variety of graphs present within the application. An ideal solution would use a consolidated graph, with different routes between different Nodes being considered, however this is seen as problematic in the current time frame.

A final possible addition, though not likely to be achievable in the time frame, is to provide access from anywhere to somewhere on campus. This means having a user be guided into the graph from areas outside of it, meaning dynamic routes would have to be created. Initial research shows this could be possible using the Google Directions API however it is likely that with other development ongoing this is likely to be feature developed outside of the initial time frame. However the use of this feature would be a real benefit for the application, a simplistic solution would have the user guided to the nearest node and then the search algorithm would take it from there. It is even possible that in future the whole graph would be replaced with the Directions API, this however would require the user to have access to the internet at all times and it is simply some-

thing that cannot be guaranteed. Due to this the hard coded graph and search related to the graph is likely to be present for the lifetime of the application due to the required off line capabilities.

1.2.1.2 Route Plotting

This desired features comes as a result of the original meetings, the 'customers' were unhappy with the current system of plotting routes, which were all done manually. Part of the solution is already achieved through the use of a graph, a location only needs to be linked to one other to have a route available to anywhere on the graph. However this should be further developed for the application to be seen as fully functional. The application should include a feature that lets a user log a route and have it printed out to a file compatible with the current route finding solution, this way the person responsible for the future maintenance of the application can visualize a route before they add it to the current system. A route should be able to be visualized while it is being created, so that the user is not blindly adding points. GPS co ordinates received from some devices can be far from where the actual user is. Functionality within the Google Maps API allows us to show the user where the phone thinks they are, this should help them plot accurate points.

The output file, the files used by the route finder, should also be human readable. This requirement is not as key as the others but should be achievable as the file reader will be coded for this exact purpose. Having a human readable file provides various benefits, mostly that of being easily editable by anyone who wishes to change information in the future. Its negatives are few, while a non human readable may be able to be smaller the benefits time wise will be minimal. This also means that route plotting using the in built feature is not required, files can still be made by hand due to the standard format to be laid out.

Finally the Route Plotter should provide a grading for the route that has been laid out by the user. This is the grading that will be displayed by colour on the route finder, while the initial rating is likely to be numerical this will be translated into the red, amber, green system that has been mentioned in the previous meetings. Having this easily identifiable system provides the user with a sense of understanding without having to state the feature on screen and cause cluttering in the UI. Ideally the route plotter will take into account cumulative distance, steps on the route and elevation. However after initial research elevation is not only sometimes unreliable but a complex overall problem, to do with both the misleading information it can provide and the problems in finding elevation. However an ideal solution would take all three factors into account fairly.

1.2.1.3 Building Display

Building display is a feature currently implemented fairly well within the original application and is something we can mostly just adapt, however there are improvements that can be made, mostly found due to the user feedback. This feature should allow for users to highlight building types and have them displayed on a map of campus, this way a user can browse around what they want rather than have to try and guess where certain areas of interest are. A user should also have there location displayed to them so that they know where a building is in relation to them.

One criticism that has come from the user feedback is that sometimes the display can be less than clear, this includes building departments sometimes not being named and just a general lack of clarity. A good example is buildings named after people just having the persons name, for example 'Hugh Owen'. Problems like this should be fixed for the requirement to be met. Again a

facility for easy addition to this feature should be added, preferable through on device files for the time being due to the previously highlighted problems with a central file store.

1.2.1.4 Location Based Help

Location Based help is seen as the easiest implemented feature, something that the web application also currently lacks. It is seen as more of a test than the other features which seem solid and long lasting, the resources available to the University may not be able to facilitate the full deployment this feature may require. In short the feature should be able to provide the user with the details of the closest person who could provide help, currently this is limited to the three libraries which have their accessibility officers details listed on line. However other buildings will have persons responsible for this its just their details need to be retrieved from either the departments the buildings are home to or the University.

A possible change to this implementation is the addition of porters to the help listed, if all three locations are too far away, a porters could be contacted instead. However a dialouge would have to be opened up with the University over whether they find these practices acceptable. For this application they will be implemented as a proof of concept. But in future they could either be removed or only available on set days when the demand for the services is expected.

A further idea which would have to be completed after the projects time line is to have two separate applications on open days, one for visitors which is the one being developed in this project and one for the helpers hired by the University for the day. The application provided to the Helpers would be linked with the Help function shown here, on Open Days the application would take a users request for help and give one of the Helpers their location. By doing this we provide a way for confused users or just users looking for a personal guide a way to acquire both of those. This could be completed in a fairly short amount of time with a developer who has experience with what is likely to be a PHP and SQL back end. However this is a future plan and not something to be expected from this application.

1.2.1.5 UI Design

Interface Design itself contains a set of requirements that can be mostly boiled down to one key aspect - an intuitive interface for the User. Currently there are several problems with the existing interface that can be fixed fairly easily, like the lack of a menu and unclear buttons. However a fully intuitive interface will take time to develop, testing and a good amount of inspiration from other sources. Within this project it is the aim to have a clean, practical layout that most users can just pick up and use without and previous knowledge of the layout. By doing this we guarantee less confusion and problem free use of the project, it also means we would get less users wanting to use the Help feature which if implemented would be a big bonus for the University and development team.

This requirement can also be broken down into further aspects than just 'intuitive design'. Firstly the theme used within the application must be consistent without, it would also be beneficial if it coincided with the Universities colours of yellow and purple but this will not be treated as a requirement as of now, due to the restrictions it places on the development team. Buttons should be labelled clearly with reactive designs to users touch, there are few issues worse than no responsive buttons that do not give the user the feeling of actually starting a process. The design should also

be simple with non overpowering colours, we want the user to feel comfortable and a confusing sea of bright colours over a map is not a good way to achieve this.

Future goals could be varied colour schemes based on colour blindness, with the application aiming to help those with disabilities ignoring those with colour blindness is a large oversight and is something that should be considered in the future. It may be possible to implemented in this version and is something that should be strived for. Another feature which is seen as a key requirement is Welsh language support. With the application being developed for a Welsh University the inclusion of this is key, current research points to this being more than possible despite Android not natively supporting Welsh as a current locale. This can be programmed in however meaning it should be achieved in the current time frame.

1.2.2 Itemized Requirements

Following is the list of requirements for the project, some requirements can also be seen in the requirements document however these detail a more up to date version with more focus on smaller details.

- FR1 - Route Finding - Guiding a User around campus, including the display of route grading and a no steps graph.
- FR2 - Route Plotting - Users must be able to plot their own routes around campus and print them to an application compatible file.
- FR3 - Location Based Help - Provide details of possible help available, possibly include the addition of contacting porters.
- FR4 - Building Display - A display of related buildings to be filtered by the user, contains accurate information on buildings and the departments in them.
- FR5 - Multi Lingual Support - Welsh translations provided in the application and a user choice on what language they would like.
- FR6 - A clean and simplistic UI with clearly labelled buttons and a mostly responsive design, possibly including colour blind themes.

1.3 Process

For the development of this project an adapted Feature Driven Development style was used, taking a lot of ideas from FDD but still changing the development style slightly. To begin with an overall design was specified in the Design Specification document, this included the design for the entire program. It covers a range of information, including an overall model which is the starting step of FDD, information on the key expected algorithms and key classes to be expected.

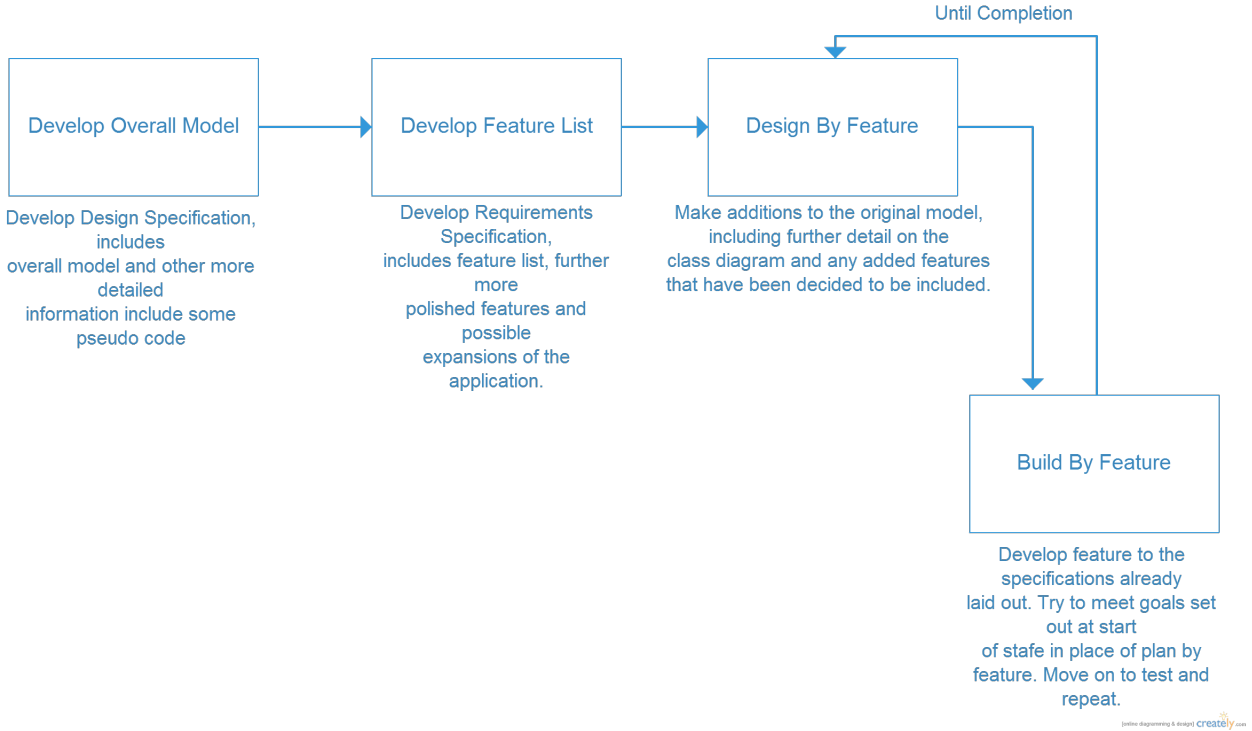
A type of feature list was also developed within the Requirements specification, while more verbose than may be expected with a feature list it still detailed the features expected and the finer details to be included. This document is more abstract and tends to not focus on the actual technology too much, just what is desired from the program itself. The Requirements Specification

also outlines key attributes that must be included within the application, this includes details on the maintainability of the project and possible choices for future expansion.

Planning by feature was something that was mainly omitted from the process, due to using a single developer all of the work load fell onto the single person. However milestones were still set for iterations of every three days or so, estimations on progress were made. A set feature was either expected to be completed or a quantifiable progress. Some difficult areas of the project led to targets not being met but overall the model provides a target to aim for in a given time frame.

Designing by feature was also a process that only partly took place, due to the majority of design happening near the start of the project a majority of the time the design by feature process involved the editing of existing documentation to better reflect what was now expected. As development continues designs for future features tend to change a lot due to shifting scope and time restrictions either becoming tighter or allowing more polished features. Having a design from the start meant that development could continually take place without major breaks for planning out a feature, this meant that once the application hit its main development stage it got quicker to develop due to a familiarity with the code base and the final image of the software becoming clearer as time progressed.

A lot of time after the initial design was completed was spent on development, this meant progress and development tended to be quick. Building by feature meant the application had modules added to it weekly, creating a progress that was quantifiable. This development technique was decided to be the best process for a smooth development period. It also helped that the application could clearly be visualized as having several different branches, most of which did not rely on one another. After development was finished tests were performed, both code wise and with the application itself ensuring the code worked as expected. It was also important to test that the code was expandable and that the plans for the future could be included in the way that was highlighted within the Design and Requirements specification.



Chapter 2

Design

The following section will cover the final design of the project, initial design can be seen in the Design Specification. The original design was completed when only a small amount of research had been completed and not all avenues of interest had been explored. Due to this good comparisons can be made between the two making it easy to highlight the areas of the design where large changes were made. The design covered is the cumulative output of the Design by Feature element in the process. This means some changes were still made during the implementation phase but those will be discussed in the relevant section.

2.1 Support and Development Tools

2.1.1 Development Environment

During the initial phases of development and prototyping Eclipse and the Android Development Tools plug in were used to help develop the application, this was due to the developer having experience with this platform and the code being sampled from other projects was also built within this platform. However not long after initial prototyping and tech spikes had started to take place the decision was made to swap over to Android Studio and develop from there. Most of the decision comes down to personal preference but there are definitely good justifications to swap, personal preferences included appearance and responsiveness as well as easy to use features such as inbuilt git support and intuitive interfaces.

One major contributing factor was the initial hassle in updating the ADT to their latest versions, this process alone took hours of research and tinkering to get fully working. During this time it was discovered that Google had stopped support for Eclipse ADT and would be focusing on the now fully released Android Studio. A decision was made, based on the advice from the Android API and other sites, to move over to Android Studio as it would quickly surpass Eclipse ADT with some features already being far more helpful.

One key feature that was seen as a bonus when compared to ADT was the interface editor, while this is present in the Eclipse version the process in Android Studio seemed much more responsive and seemed to give a clearer idea of what XML was being written to support the graphical changes. While nearly all interface development was completed using pure XML sometimes moving elements using the graphical layout gave good suggestions on exactly how a set layout could be achieved. While the code produced by this could sometimes be messy and convoluted it could also be cleaned up and improved.

Some of the choice to move over was also personal opinion of the developers, with so much time having to be spent in the IDE it had to be at least a suitable experience. After past experience with Eclipse and ADT causing issues due to freezing and crashes the supposedly much more stable AS was worth adapting to. AS also seemed to provide a much fuller auto complete feature, suggestions would be made sometimes within a single letter of a word being typed out, normally correct suggestions. Once used to this process code could be written up slightly quicker making the experience overall more enjoyable and productive. Overall the experience with AS was that it was far less cumbersome and much more enjoyable than Eclipse ADT. While currently there may not be huge differences the ones included were very helpful, the implementation of Gradle meant single lines could be used for the inclusion of dependencies and possible libraries. Easy inclusion like this meant that it was a lot more appealing to try out libraries and see what they added to the project. While Gradle provides many benefits not all were made of use within the project.

A final reason for choosing Android Studio was that after seeing various comments on Android discussion boards it became obvious that it was quickly becoming the industry standard, unsurprisingly due to the lack of support for Eclipse ADT. Adapting to AS seemed the most logical decision, made easier by its sleek appearance and performance.

2.1.2 Version Control

Version control was an easy problem to solve, as standard GitHub was used to upload versions of code and documents allowing the history of changes to be made and development easily tracked. It also meant that in future the project could easily be open sourced through the changing of the privacy settings on the repository. Using GitHub also meant that even when on the move developers could check the code base through the website to see if potential solutions could possibly be implemented.

One alternative to GitHub which was considered was BitBucket, BitBucket provided unlimited private repositories and a desktop client which could have been of a fair amount of use. However looking at reviews and the general design, the developer decided that while there were benefits in using BitBucket over GitHub, they did not really apply to the project due to its small size. In future if the project was undertaken by a larger team, or the developer was part of a larger team for a seperate project, BitBucket could be heavily considered due to its unlimited private repositories as standard.

Having the code stored off site as such also provides a range of benefits to the developers. It mostly frees up the developers to not worry about breaking any code, back ups from various times are all available with the ability to roll back to them with minimal effort. It also meant that the code was always safe, once the code had been moved to the GitHub servers its available for the future, no need to worry about corrupt drives or other similar issues.

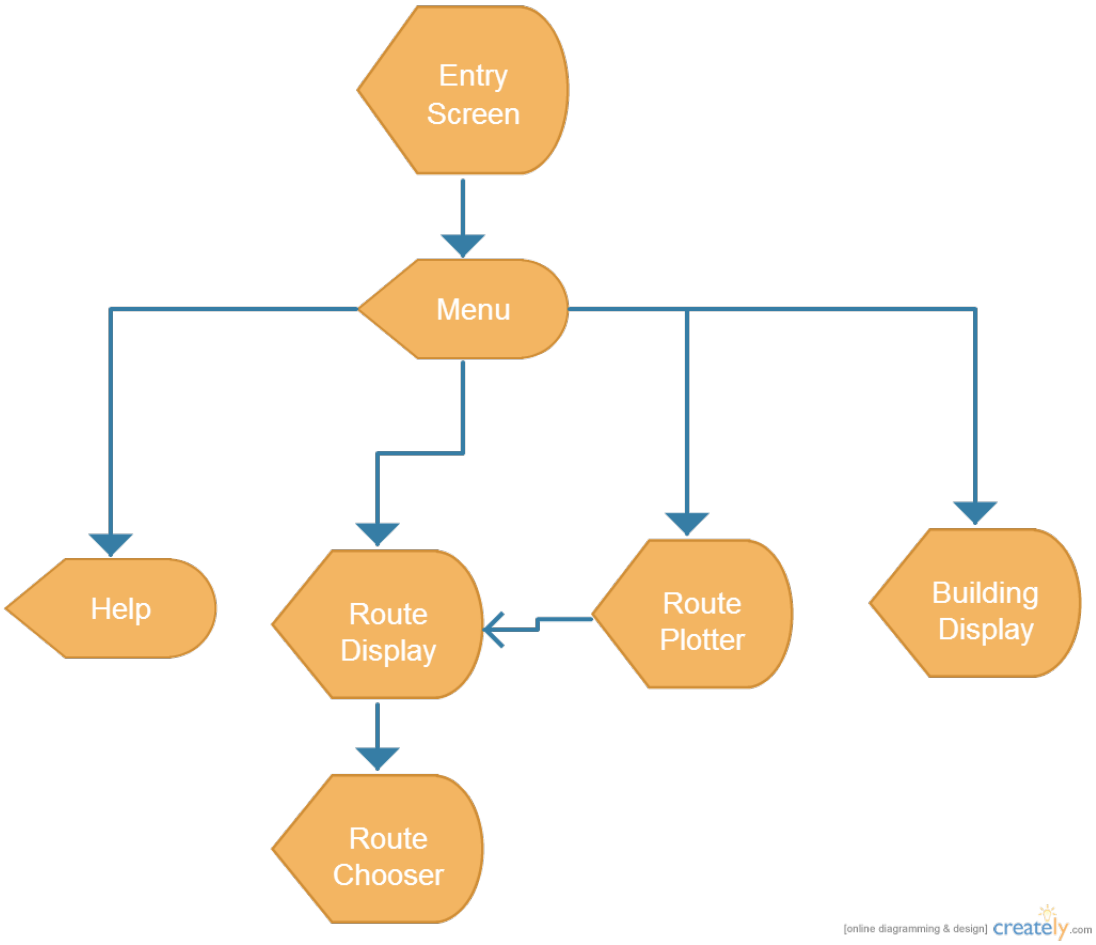
GitHub also provides a simple and easy to use graphical interface as an alternative to the command line interface. This interface was sufficient for the project being completed with only a few clicks allowing for the new code base to be committed and pushed to the on line version. While this may not seem a major benefit it still freed up some time and overall made the task of version control slightly more enjoyable.

An alternative to using GitHub for version control and code backup was either storing backups externally using flash drives and external hard drives or using the file store given to students by the University. However the benefits provided by these were more than covered by what GitHub provided, it was also decided that there was a minimal chance of losing any data stored through GitHub while it was possible that we could lose access to the file store for prolonged periods of time without prior notice. This has been previously experienced and was not something that the developer wanted to risk.

Without version control it would be possible to completely lose track of what had changed, as well as losing the project as a whole. Storing everything locally is extremely bad practice as any amount of extenuating circumstances could lead to either the loss or damage of the computer that the information is stored on. Furthermore GitHub provided statistics on when most commits were made and exactly what was committed, this allowed for estimations on not only how long a feature would take to develop based on what time had currently passed, but how long a future feature would take based on its complexity compared to past features. This meant that during development a fairly tight schedule of development was set out and mainly adhered by.

2.2 Overall Architecture

Initial design for the project was very broad becoming more focused over development time. With the Design Specification it is possible to see the first properly documented steps in preparing for the development time within the project, however other initial steps were taken first to help build a design from. At the start of the project a very abstract set of diagrams were drawn up to describe the desired boundaries for the project, this was mainly as a guide for further development but aids in demonstrating the vision of the project from the start. While these were drawn up on paper the digitized version will follow, it can be seen that the diagram is essentially a flow diagram that only involves the Activities used in the Android application. No screen details are given just the links between them, this has changed significantly over development but the basic blocks can be seen here. A link can be seen between Route Plotter and Route Display, this was to represent the output from one being compatible with the other.



2.2.1 Class Diagram and Justification

The final Class Diagram represents what was thought to be the final representation before the Build Feature segment in the process, during implementation some changes were made but the class diagram is a good representation of how the program actually works. Justification of the design will follow with diagram afterwards.

2.2.1.1 Justification

In this diagram the Menu Activity serves as an entry point into the program, in production it was always intended to have a simple screen before this, however for the case of the class diagram representing it is non informative as it does nothing but show a logo, it is not essential. Within the program the simplest function is the Help Activity, this Activity is created through a button click within the Menu Activity. Within the Activity is an ArrayList of Place, a Place is a simple representation of a location on the planet and is used repeatedly throughout the code for varying uses, in this case it represents a set of potential helpers. Within the Place objects a persons name is the Name variable and their details as the Description, these helper points will be loaded in from a file and then manipulated with methods within the Help function, these methods will load the points, sort them and display the closest one to the user.

One of the possible alternative options that was considered here was the inclusion of a Person object and instead of a Place object have a Building object which could have an array of People. However the choice was made to have people represented as places, this was due to an effort to simplify the programs structure. With Place being used for varying reasons without it being overcomplicated no need was seen to include two separate classes. If all places were buildings it would have been considered a worthwhile effort, however Places are also used to represent points on a Route and they are not always Buildings. As the program stands it is sufficient to have one class represent people and places, including buildings, however in future if more expansions are completed this may change. One area which may change this is the inclusion of porters or open day helpers in the category of 'Help'. Huge changes would need to be made to the structure of the program to support this feature however, including external hardware to act as a middle man between the application and the 'Help' on hand.

Slightly more complicated is the building display feature which is made up of several classes including the reuse of the Place class. In this case the Place class is used to represent a set of locations relating to a category defined by the user, this ranges from facilities on campus to lecture theatres. Within the Map Activities onCreate method is the code to render different views based on the request of the user, as the Map class is used to represent both route plotting and building display. While the same XML should be used, methods within the Activity should manipulate it, acting as a type of presenter. A choice which arose in the implementation of this is whether it would just be simpler to have two different Activities for the tasks, however with how little they both used and the fact both tasks used a map fragment it was realised it may be just as effective to use the same Activity.

With the Building Display function being used the class Place is now used to represent buildings, making use of their 'type' variable. Types are chosen from an Enumeration class which limits the pool of values, while it would have been possible to just use Strings the Enumeration removes the chance of mislabelling a buildings type as the pool of values are the only acceptable choices. This makes it easy to error check for erroneous data on the load in of the buildings, having the

added error checking assists in the robustness of the application.

All of the data loaded in is then used within the Map Activity, within the Activity are the methods to handle the representing of a set of locations based on their type variable. With us having a handle to all of the locations the displaying of them is easy to represent with the Google Map, using the showByType method and passing it the locations and a desired help will suffice in rendering any of the types we want. It is best to do this on a button click, likely from a pop up menu. By doing this we can pass the String used in the menu and have that serve as the type, all this requires is correct labelling of the items in the menu. We can then cycle throughout the data object and show the ones we need as marker objects on the map, showing both their name and description as was set as a requirement due to past user feedback. It is debatable whether all previous displayed locations should be wiped after a new category is selected, or whether a user should instead be able to choose a set of categories for display. To begin with a user will only be able to select one category at a time but this could change with future user feedback.

One of the more complex features which has involved a lot of design decisions has been the inclusion of the Route plotter feature, due to the complexity of route representation on a map many ideal features are simply impossible to implement in an ideal and functional way, some of this is down the the unreliable nature of near perfect accuracy from GPS. When a user requests to plot a route they are originally brought to the RouteDataEntry Activity, this is a simple activity which will let the user put in anything they like as the start point, to help with adding new original locations, but limit them to entering a pre existing location on the graph as a destination, thus ensuring a fully connected and traversable graph. It should also allow for the user to specify whether the route is for the step free or standard graph.

Once the user has completed entering the required data, the Map Activity should render itself based on the users request. In this case, the display button will be replaced with one which contains a menu for the methods required to accurately plot a route. A problem encountered in prototyping is the creation of a Place for the Route object based on a user's location, this has been down to the fact that GPS co ordinates are not always accurate, especially in built up areas. To combat this, it has been decided that a point will only be added to a Route on the user's command, with the user's location being displayed on the Map object. An alternative option would be to have the device log a point automatically every five seconds, however this caused many 'off' locations to be included. With the inclusion of these off locations, an almost jagged path was produced compared to what user defined logging times provides.

To aid with the visual representation of a route each time a user logs a point after the first a poly line should be used to represent the traversed path. This should be done using the connectPoints method which takes two Place objects from the Route. Having a visual representation provides much better results than having a user blindly plot a path, it also means that combined with a user being able to see there location they can accurately tell the line that will be drawn before it is. A small incremental counter should also be included within the XML file to log the steps encountered on a route. Once a user has finished logging the path they wish to add to the graph they should be able to print a program compatible file. This is handled by the printFile function which takes the completed Route, within this a variety of functions are called resulting in a final printed file. Methods called from within the printFile will be calcDistance to return the final distance between points and the calcGrade function to return the final grading of the route. Grading will be done by normalizing the total distance and steps to the same scale to figure out there 'rating'. A set of boundaries will be used to define the rating.

One major difference between this design and the previous included within the Design Specification is the change of what the Map object considers its route. Initially it was implemented so that the Activity stored information considering the grading, start and destination along with an ArrayList of Place. However it was recognised that this is a bad representation considering we already have a working Route object used within route finding. If the object represents a route in one part of the application but not another it could be considered confusing for future developers. An additional change is the inclusion of the data entry Activity, this was an oversight in the initial design and something that has been fixed in this version.

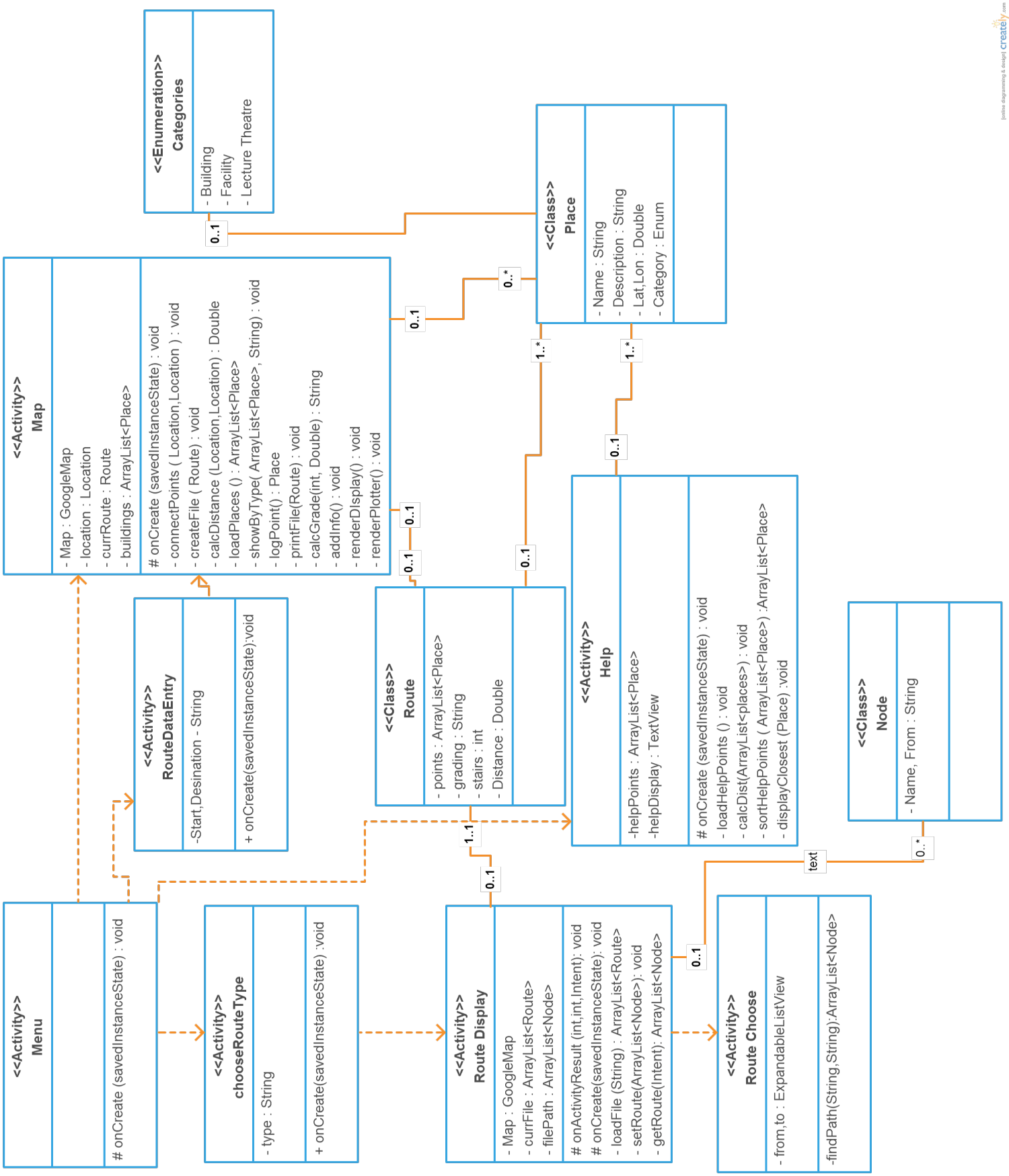
Route Finding within the program will be by far the most complex feature to implement. This is down to a variety of reasons including having to implement a search technique which performs reliably. Before any graph analysis is done the user should select what graph they want to use by making a choice between what is represented as two separate features. They should be asked if they want standard routes or easier routes, shown as a 'no step' option. While this appears as two separate features all it should realistically change is the graph used for search and traversal, this also opens up the possibility of having more than two graphs in future however more complex and effective solutions could be included.

Representation of a Route is something that a fair amount of thought has been put into before development, due to the importance of being accurate. Within the proposed system a Route is essentially details of two connecting Nodes on a graph and the link between them. As it stands the link between the nodes is represented as a set of GPS co ordinates which when plotted in order describe a set of destinations which lead to the final location. By doing this we help represent curves in the links between nodes. In this design a links weight is represented as a grading which is set based on criteria laid out in the Route Plotter Activity. Searching could be completed based on route grading but may be outside of the scope on this project.

After the choice of what type of routes they want returned the user should be taken to a Map object with a button which will take them to the RouteChoose Activity. This Activity will contain two expandable list views populated through a file within the relevant assets folder. Another option would have been hard coded values but these do not provide the expandability felt necessary. Once a user has chosen their start and destination they should be taken back to the Route Display Activity to see the route, before this happens the route will be found using a search technique within the Route Choose Activity. This method will return what is actually an array of Routes, by doing this we have segments to our routes and can manipulate its colour in segments allowing for the display of different colours based on the difficulty of small sections.

A choice which arises here is exactly how to search for a Route through the graph. Currently it has been decided that a simple Breadth First Search will be implemented. By doing this we can move through the graph and continue until we have found a small route which ends in the destination and build backwards from there. In future, if the graph expands to be quite large, it is suggested another more complex search technique be included if time is not found within this project.

Once a user is returned to the Route Display a set of methods will be run to handle displaying the routes. First the Route array will be pulled out of the Intent returned from the Chooser Activity, this will then be set using the setRoute method which will in turn use the loadFile method to get access to routes stored within the programs assets. As previously stated the Assets folder accessed will depends on the type of route finding the user has requested. A small window should also display the steps on a route and other information including the distance that has to be covered.


<<Enumeration>>
Categories
- Building
- Facility
- Lecture Theatre
0..1
<<Class>>
Place
- Name : String
- Description : String
- Lat,Lon : Double
- Category : Enum
0..*
1..*
<<Activity>>
Map
- Map : GoogleMap
- location : Location
- currRoute : Route
- buildings : ArrayList<Place>
onCreate (savedInstanceState) : void
- connectPoints (Location,Location) : void
- createFile (Route) : void
- calcDistance (Location,Location) : Double
- loadPlaces () : ArrayList<Place>
- showByType(ArrayList<Place>, String) : void
- logPoint() : Place
- printFile(Route) : void
- calcGrade(int, Double) : String
- addInfo() : void
- renderDIsplay() : void
- renderPlotter() : void
0..1
<<Activity>>
RouteDataEntry
-Start,Desination - String
+ onCreate(savedInstanceState):void
<<Class>>
Route
- points : ArrayList<Place>
- grading : String
- stairs : int
- Distance : Double
<<Activity>>
Help
-helpPoints : ArrayList<Place>
-helpDisplay : TextView
onCreate (savedInstanceState) : void
- loadHelpPoints () : void
- calcDist(ArrayList<places>) : void
- sortHelpPoints (ArrayList<Place>) :ArrayList<Place>
- displayClosest (Place) :void
<<Class>>
Node
- Name, From : String
0..*
text
<<Activity>>
Menu
onCreate (savedInstanceState) : void
<<Activity>>
chooseRouteType
- type : String
+ onCreate(savedInstanceState) :void
<<Activity>>
Route Display
- Map : GoogleMap
- currFile : ArrayList<Route>
- filePath : ArrayList<Node>
onActivityResult (int,int,Intent): void
onCreate(savedInstanceState): void
- loadFile (String) : ArrayList<Route>
- setRoute(ArrayList<Node>): void
- getRoute(Intent): ArrayList<Node>
1..1
<<Activity>>
Route Choose
- from,to : ExpandableListView
-findPath(String,String):ArrayList<Node>

2.3 Application Flow

With this application most users will be first time users, it is not likely it will be repeatedly used by someone after they are used to the campus. With this in mind it becomes clear that good UI design is important, along with logical flow of the application. A basic flow of screens can be seen in 2.2 however it describes a very broad series of events. In this section a detailed flow of data and screens will be detailed to help with the implementation phase. The diagrams following and the textual descriptions should be used as a framework to build around.

2.3.1 Overall Justification

A similar diagram to this can be seen in the Design Specification, section 4.1, however this revised version details updates including additions of screens and specific details on what should be contained within some algorithms and the alternative considered.

One of the changes included within this diagram is the inclusion of the Route Data Entry Activity, this is used for the user to enter the start and destination of the route they are plotting. While this obviously needed to be included it was not within the original diagram, a major decision revolving around this screen is exactly where it should be placed within the application. It is viable to include the screen between either the choice and the plotting screen, or have it used after the user chooses to upload the route they are currently plotting. It has been decided to include this before the user plots anything, this is due to to the idea that the user should provide broad details like the start and end before adding information about the detailed route. It follow a logical progression this way.

After these details are chosen the $addInfo()$ and $renderPlotter()$ functions are run. These should actually be run in the $onCreate()$ within the Map Activity. This way the information added in can be accessed from the past Intent to detail what to render and what to name the route. This also applies for the $renderDisplay()$ method, the decision on which should be run is actually made in the $onCreate()$ it just makes more sense to draw it as happening before the screen fully exists, which is technically true as the $onCreate()$ method is what is creating the screen.

Once at the Map activity various methods can be run depending on what was rendered. If the Building Display screen is shown the user can only select the type of a building to show, one type at a time. This has been a feature repeatedly mentioned, the hard decision to make is whether users will want to see multiple categories at a time. Testing after implementation will have to be completed to gather new user feedback as little is mentioned from the original user feedback. With it not being an issue previously it is considered to not be one currently.

If the user has selected to be able to log a point various possibilities become available, the main method being $logPoint()$ which allows the addition of a point, this should use a Location Manager to retrieve GPS co ordinates and add them to both the map and a Route object for printing later. Once a second point has been added, and continually from there, lines should be displayed to visualize the route. This is easily done using a poly line on the Map object and the co ordinates already retrieved. Other options include $cancel()$ which will reset the Route object and call $Map.clear()$ to provide a fresh start. Another decision was made here, it would be possible on cancelling a walk to bring the user back to the data entry screen, however it is being assumed that the user will have got that information right and if not will return themselves, forcing the restart of the whole process is unnecessary. A smaller method is actually the $addStair()$ function, this

should be run on the change of a value in the stair counter. It will simply update the value of the stairs in the Route object.

Finally the $saveRoute()$ function could be called, to print the Route object to a file compatible with the Route finder. Printing the file is fairly simple, it just has to match the format set out for Routes which will be discussed further on. A distinction to make here is that the file must be printed to External memory, saving to internal makes the file inaccessible unless the devise is rooted or to the application itself. A major decision made here, discussed previously in the original Design specification 2.1.2 and here in 1.1.1 , is whether to have it uploaded or just saved, in short internet on campus is not always available so the decision has been made to save locally to avoid problems relating to communications. Saving the route should also provide a grading, this will be decided through a small formula which considers total distance covered between points and the steps added on the route.

Most information on the Help section of the application has been discussed previously in Design 4.2.1. It is a fairly simple process, loading in help points and finding the closest using the $greatCircle()$ function. However a further design decision has been made which is hard to represent visually. This is if the value should be updated past the first run, whether the help screen should change while the user walks around with it open. However this idea has been omitted due to the problems and uncertainty it causes, if the method is run every ten seconds for an update, poor GPS signal could mean that when around the boundary of a help point the wrong one is shown occasionally causing confusion. To combat this it would mean taking a set of co ordinates over a time span and finding an average using them, giving a more accurate figure. However it has been decided to just use the first reading, GPS is mainly reliable and the problem revolving around a boundary is unlikely to appear, most people will be well within a building when they need help, not outside and close to another with a help point.

By far the most complex feature in the application is the route finder and a great deal of thought has gone into how to configure the set up to fit into what the developer thinks is possible during the design phase. While there are many ways to implement what is done, an affecting factor was always the extendibility of the project. At completion the project should have many open ends within it, to allow for the addition of further graphs,locations and make the editing of menus easy.

In the flow diagram the user is initially sent to the Route type activity, this should allow the user to choose between two separate modes, the step free mode or the standard mode. Step free will be a graph that has been built that avoids stairs. While it would be possible to build a more complex base graph and further change the algorithm the problems these implementations pose could create time restrictions on other areas of the project. Once chosen the type will be saved by the Route Display activity.

Once at the Route Display activity initially just a map of Aberystwyth and the campus will be shown as no route has been selected, the user should have to click a button to be taken to the Route Choice activity. On this page Expandable List Views should be used to simplify the route selection process, these should then be populated from a text file to help with addition of locations further on. After the user has chosen there locations, feedback should be provided on the page of exactly which locations are to be currently used. Giving feedback to the user through the entire project is key, something that will be discussed further in the design section.

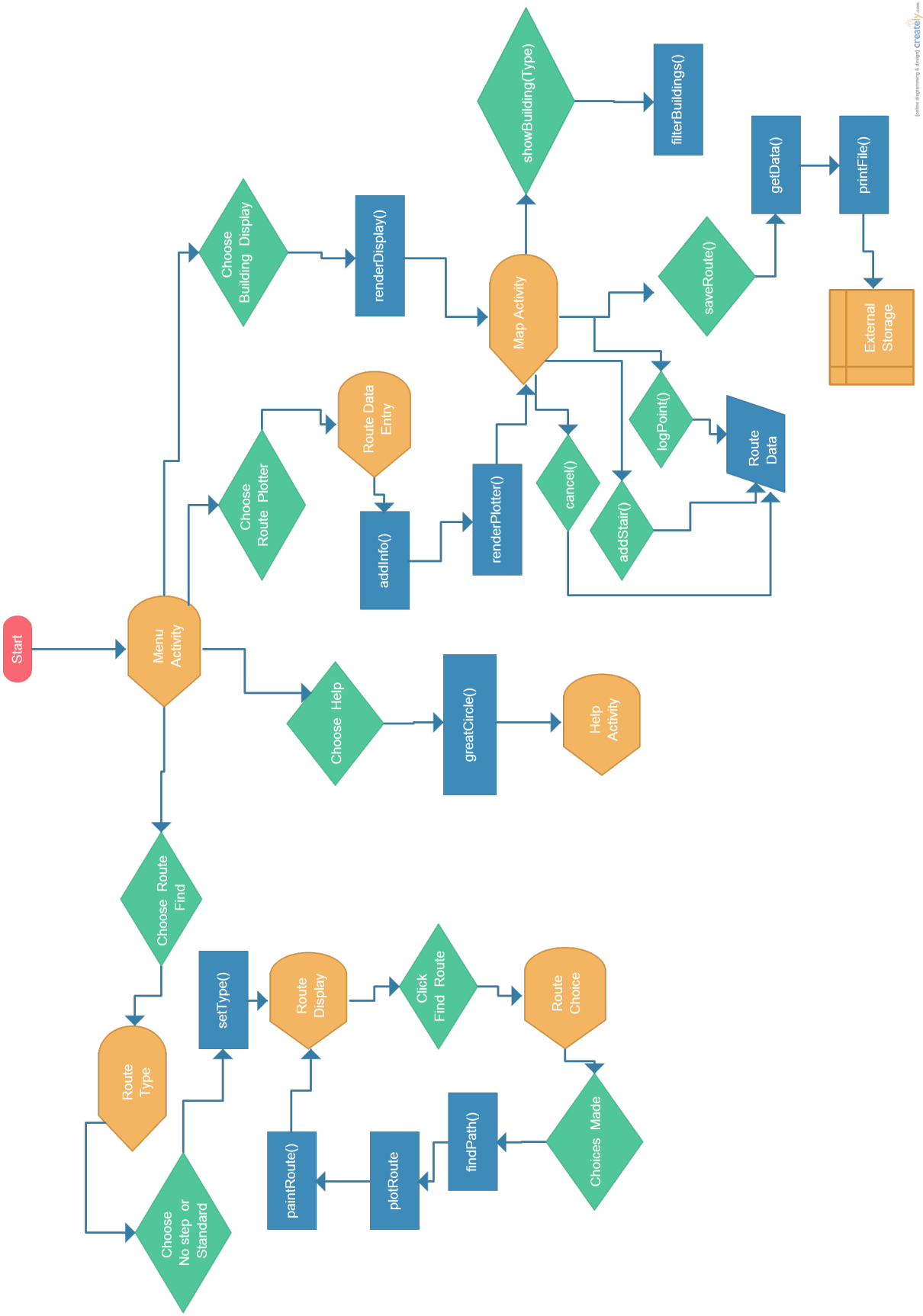
Once these have been chosen the Breadth First Search will be run to find the path, further details provided in section 2.5. What is likely to happen is that the algorithm will find a set of paths rather than one, these set of paths will build to link a full route from the start to the

destination. Implementing the system in this way fixes the problems with the maintainers having to add hundreds of routes just to add a new location, in this version all that has to be built in is one new route connecting the graph to the location and vice versa.

Plotting the route is seen as a separate task to finding it, finding a path through a graph will end with us just knowing the last accessed node which opened the destination. From there we need to plot backwards from the end. Moving through the nodes based on what location opened another should allow us to move backwards from the end node to the start. This also gives us the smaller routes to plot to display the full route to the user.

Painting the route is then fairly simple, we know which location opened another so we can just search for that locations file and paint the single route to the next one. Repeatedly doing this for each node until the destination should lead to the full route being displayed to the user.

Some consideration should be made to logging a users location on the route and showing it to them, however this can be enabled through the use of a Google Maps API method called $setLocationEnabled$ which should give us the required effect. From there this can be customised to provide the user with more information than a dot.


Start
Menu Activity
Choose Building Display
renderDisplay()
Map Activity
showBuilding(Type)
filterBuildings()
saveRoute()
getData()
printFile()
External Storage
Choose Route Plotter
Route Data Entry
addInfo()
renderPlotter()
cancel()
addStair()
logPoint()
Route Data
Choose Help
greatCircle()
Help Activity
Choose Route Find
Route Type
Choose No step or Standard
setType()
Route Display
paintRoute()
plotRoute
Click Find Route
findPath()
Route Choice
Choices Made

2.4 Sequence Diagrams

2.5 Major Algorithms

Throughout the application there are thought to be a range of major algorithms, most of these relate to the finding of a route and then the visualizing of that route. Currently the algorithm is expected to be a version of BFS when required to go past a single connecting node. Past designs of these algorithms can be seen in Design Specification 3.3. Following are the currently expected algorithms to find routes, calculate distance and read in the file format.

Initially route finding was shown in a single algorithm which is mostly correct, however due to the files containing all of the routes from a point the process has been changed. Initially a check will be made to see if there is a direct connection between start and end, if not then the BFS algorithms will be called. To start the algorithm should look as follows, this should check the original nodes connecting nodes. It logs the connecting nodes and if none are the destination they will be passed to the $deepFind$ which is the BFS.

It does this by using a boolean which is changed on a route being found. At the start of the BFS it should be noted that the start node, which will be stored in the activities variables, should be added to the visited list to avoid the nodes checking that.

Find Initial Path

```
1:  startRoutes = read(start)
2:  found = 0
3:  ArrayList routeEnds
4:  for startRoutes do
5:      routeEnds add route− > end
6:      if route− > end = end then
7:          Plot Single Route route
8:          found = 1
9:      end if
10: end for
11: if found == 0 then
12:     deepFind routeEnds
13: end if
```

The deep find algorithm will initially be a BFS, a mostly correct version can be seen within the Design Specification however it does not account for being passed it starting nodes. The actual implementation should look close to the following pseudo code.

In this example you can see that the start node is set to visited and the nodes to be cycled through as those passed in as a variable. While the algorithm looks relatively simple set like this due to having to implement loading from the devices memory and other functions it is expected to take time to develop. Initial research has been conducted on loading.

A key line which helps in a later algorithm can be seen on line 11. This helps us move backwards from our destination as we can tell which connecting Node each Node was opened from, essentially plotting the path backwards by tracing the algorithms steps.

In future it is advised that more complex search algorithms are considered, including A*, to

try and find a more efficient algorithm as on a large graph BFS could cause a small delay on older mobile devices.

Find Route

```
1:  start set Visited
2:  Nodes = passedInNodes
3:  while !route found do this
4:      for Nodes do
5:          if Node has been visited then
6:              do nothing
7:          else
8:              set Node visited
9:              get NodeRoutes
10:             for NodeRoutes do
11:                 Set Opened from Node
12:                 if NodeRouteEnd = Destination then
13:                     set route found true
14:                     call Plot Path (visited)
15:                 end if
16:                 if NodeRouteEnd not visited then
17:                     Add to Nodes
18:                 end if
19:             end for
20:         end if
21:     end for
22: end while
```

As can be seen in the above algorithm the visited nodes are passed in as a variable to the function that will plot the path, this way access is provided to the information about which Node was opened from where, allowing the function to work backwards though the Nodes.

This is possible due to the fact that as soon as a single route is found that connects to the end no more are looked for, by doing this we guarantee only one path being painted. However this plan does not work if the search finds multiple paths, something that needs to be noted for future development.

Due to the nested loops the algorithm should always paint all required routes.

Plot Path

```
1:  VisitedNodes = visited
2:  search = destination
3:  for VisitedNodes do
4:      for VisitedNodes do
5:          if search = NodeName then
6:              Plot Single Route 9Node− > name, Node− > from
7:              search = Nodefrom
8:          end if
9:      end for
10: end for
```

Pseudo code for plotting a single route and then painting it can be found in the Design Specification sections 3.3 and 3.4. These have not changed through further design and are expected to suffice as they are. This also applies to the saving of a route file algorithm, found in section 3.5. However after research it has been noted that external memory is not always provided and as such further checks should be completed, following example code from both the Android API and other sources it is expected the following code should suffice.

Plot Path

1: $state$ = $getExternalState$
2: $possWrite$ = false
3: **if** $state$ = $readOnly$ **then**
4: Cannot create files
5: **else if** $state$ = $readWrite$ **then**
6: $possWrite$ = $True$
7: **else**
8: Something is wrong, we only have one desired result which we have checked for
9: **end if**
10: return $possWrite$

An algorithm which has been previously overlooked as it was initially assumed to be fairly simple is the file writer. After further research which resulted in the memory check algorithm it has been decided that its process should be noted due to the various ways it could be completed. A further decision which has been made is to include the saving of a file into the file writer function, this way when a user initially goes to save a file the folder will always be present. Initial design should be as follows.

Save File

1: $root = getMemory$
2: $MyDirectory = root/routes$
3: **if** My Directory exists **then**
4: Do Nothing
5: **else**
6: Make Dir $->$ $MyDirectory$
7: **end if**
8: New File$->$ $root$
9: $out = OutputStream->File$
10: $out->Route->Start$
11: $out->Route->End$
12: $out->Route->Grading$
13: **for** $Points$ **do**
14: $out->Pointi$
15: **end for**

2.6 Interfaces

User Interface was an area in which the developer had little to no experience, while basic GUI's had been developed before they did not truly consider a users habits and motivations. From the initial design specification to the beginning of implementation research was completed on both good design guidelines and the options provided through Android and its use of XML for layouts.

While UI was seen as a less important task than the actual underlying code a set of basic rules were set out from the start, taking influence from other applications and the Google Material Design outline. While the material design outline was not adhered to fully it provided good insight on the vision Google has for the Android platform.

- 1 - Make best use of white space within design, make it act as a barrier rather than having to add an actual barrier in.
- 2 - Bold colours will help cause distinction between background and active UI members.
- 3 - Theme should be carried throughout pages as best possible, little to no room for exceptions.
- 4 - Develop a system which supports the same experience throughout devices. XML allows for relative layouts so no fixed measurements mostly.
- 5 - Interactive elements need to be responsive, users need feedback on their actions. This can be textual or a change in colour. Noise is not practical.

2.6.1 Colour Theme

Colour theme was something that the developer felt was key to a successful UI, without it a good layout would go to waste. Due to this several sets of colours were chosen at the start for trial upon the full completion of the application. Intially design highlighted a yellow and purple theme as the likely choice, due to matching the universities website. However as design furthered and prototypes were developed of basic layouts, it became obvious this may not be the ideal theme. These themes were both developed internally and taken from on line resources. The following were chosen due to them being seen as palettes that would not only provide good distinction between members of the UI but provide an easy to view screen. Using many bright and clashing colours could possibly create a visually offensive UI which is something to be avoided. By using safe, strong colours a higher chance of success was predicted.

1. 

2. 

3. 

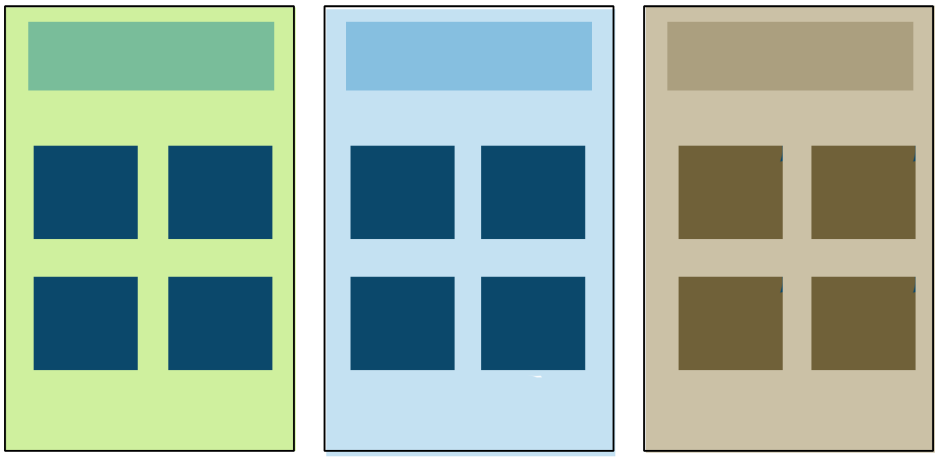
As visible from the above diagram the palettes were based on what seemed to be sequential colours as such, while no actual math was applied to come to this decision it was visually apparent that the themes were a progression of shade. However this provides the developer with a range of possibilities, with the shades being far enough away from each other, text of one colour should always be visible on another due to the lack of overlap. It is advised during development though that text of one colour be on the background of a colour at least one away.

For example, text from the middle of the first palette should only be applied to backgrounds of the two ends, while viable to put on the two connecting colours the impact of the distinct colours will be lost slightly.

As can be seen in the above diagrams the principle of using five progressive colour should provide us with a theme that is not only easy on the users eye but provides us with the ability to give a good distinction between UI members and blank space. With the above three options to be tested it is likely that either 1 or 2 will be the final theme. With 3 being viable the colours seem to be bland, while it could work with the right arrangement it is suggested more effort be put into a functional theme built of 1 or 2.

A similar palette to that of 2 can be seen at outlook.com. In this design the colours are fairly similar with white acting as the background and then a colour similar to the middle acting as interactive elements. Distinction between the two can clearly be seen and as such they stand out as having a purpose, it also helps with the clear display of text while removing cluttering multiple colours may cause. In the design responsiveness is achieved through a brief colour change in the element, something that should be considered within this project. Other considerations to be made in terms of colour design is the implementation of themes for colour-blind users, however this a stretch goal and unlikely to be implemented.

Mock up screens of the application with the prospective colour schemes can be seen below, with these it is is clear to see why 1 and 2 have an advantage over 3. It should be possible in future for the user to define a theme from a set of options, not only will this help users who make frequent use of the application but it will provide a way for the colour-blind themes to be implemented without a separate application.



As it stands theme 1 is seen to be the most likely candidate, it provides a range of sequential yet clearly varying colours that do not look quite as unvarying as the other two options. With this visual representation it is clear to see why option 3 is unlikely, however with the use of white space it could be a viable option. Colour theme is an area that will have to be tested with the help of human users, response to a design is not something we can simulate. By completing tests on actual users not only can we plot out the decision making and adapt around it, we can get feedback on clarity, simplicity and overall operability. A fully working application that a user can easily navigate through is the prime objective of this project.

2.6.2 User location

Display the users location to them is seen as a key element to route finding, displaying a route is only so much use if a user cannot see where they are on it. While the solution to this problem has already been discussed, some additions can be made to it. Mainly the addition of custom graphics to benefit the user.

This is something which is covered on the introduction page to the Maps API. From that and some basic research it is possible to see that custom graphics and a movable camera are both possible. Giving the developer to implement an almost sat-nav feel to the application.

A range of graphics have already been drawn up to discuss possible implementations of the 'arrow'.

As can be seen from the above diagram, 1 which is a representation of the basic user location does not really provide the user with too much information, it will sometimes change based on a users direction but the feedback is not obvious and those with poor vision may not be able to notice it at all.

Possible graphic 2 is the most likely option, along with 3. These two implementations of the arrow system show the user there direction and close location without too much hindrance to view or complications based on confusing graphics. It is advised that if 2 is implemented it be made to be clearly separate from the map, while unlikely to be mistaken for a map feature a bad implementation would still allow for the possibility. Option 3 avoids this problem due to the containing circle.

Option 4 is an adapted version of 1, with the direction being clearer and as such would likely be an improvement. However with the viability of 2 and 3 being much higher it is unlikely that it will be chosen.

The theme for the guidance will follow whichever is chosen by the developer for the overall application. Creating a familiarity between the user and what they are being shown, if all interactive elements are previously shown in one colour then the arrow should follow, making it recognisable as something that will change or provide assistance.

2.6.3 Screen Designs

Some initial screen designs can be seen in the original Design specification, while rough there overall display will not be altered too much for the development. However the theme used and some of the options made will make it hard for the screens to be implemented as they are on various screen sizes, especially on the menu screen.

In the original designs the Menu Activity is shown as a screen with a list of buttons which would then link the user to the respective Activity. However, on multiple screen sizes the buttons could start to look like they were rushed due to them just being in the middle, especially on larger screens. Due to this the decision has been made to implement a grid system on the main menu, by doing this we can make the segments relative to the screen and have them fill it. White space should also be left between the segments to provide clear non verbal instruction that the buttons are separate and interactive.

Further changes include the addition of several screens which were not identified during the start of the project. First of these being the Route Type Activity. In this activity the user is given

a choice between the two types of graph they can choose to find routes through. Following is a representation of what should be a fairly simple screen.

Another addition to the set of activities will be the Route Data Entry activity, this will allow the user to enter information about the route they are planning to take. To accomplish this the user should enter the start and end points, from there they can then move onto the actual logging of the route. Again a simple screen due to it being two text entry boxes, only restriction being that of the theme.

2.7 Other relevant sections

Chapter 3

Implementation

The implementation should look at any issues you encountered as you tried to implement your design. During the work, you might have found that elements of your design were unnecessary or overly complex; perhaps third party libraries were available that simplified some of the functions that you intended to implement. If things were easier in some areas, then how did you adapt your project to take account of your findings?

It is more likely that things were more complex than you first thought. In particular, were there any problems or difficulties that you found during implementation that you had to address? Did such problems simply delay you or were they more significant?

You can conclude this section by reviewing the end of the implementation stage against the planned requirements.

3.1 Location Based Help

Throughout the design phase of the project the Help Activity is referred to as most likely to be the easiest feature to implement, when it came to implementation these assumptions were mostly correct. Due to past work the developer was already aware of the Great Circle function, while research would have quickly led to the function itself it did save some time.

There are a fair amount of changes to the implementation shown in the class diagram, some of this was done to simplify the process and stop code being used twice in two different places. The largest change is that the $calcDist$ function shown in the diagram has been pulled into its own class, this was done after initial development on the activity due to the route plotter also needing a function to calculate distance, it was best practice to only have the code exist once in the application.

Another change is the removal of the $loadHelpPoints$ function, while loading from a file is completed throughout the application in this case problems arose which made the process seem to be a waste of resources during this specific time line. While looking for information regarding people who could be contacted for help, little turned up. Details relating to three buildings were found so for the current iteration they are hard coded.

A small change within the program is also what the $sortHelpPoints$ function returns, in the diagram it returns the sorted array. However in actuality it now returns the index of the closest

location, this removes some stress on the device and is overall simpler yet still effective. A further small change is the removal of the $displayClosest$ function. In actuality this was not needed, all methods are called in the $onCreate$ due to the simple nature of the task, once the right index has been found what the method was meant to do just takes a single line of code so the inclusion of this method was seen as unnecessary.

The location based help function was actually simpler than initially thought, due to its very small size. A users location is easily found using the Location Manager and from there its basic object manipulation mostly. Implementing the great circle function took some time until a fitting one could be used, taken from Stack Overflow. Other proposed solutions seemed overly complex for what was considered, this was due to most of them including a height variable which was omitted in this case due to the small nature of campus.

3.1.1 Updating Help Location

One idea that was experimented with was the idea of having the activity update when it moved into a new 'area' of help. This was done by repeated updating of the user location and rerunning of the methods which calculated the closest position. While it works in practice the major issue was what was seen as a good degree of uncertainty in the calculations.

While the Great Circle function works well, with its results being tested against measurements competed on Google Maps, reliability of the GPS co ordinates were not as certain. While co ordinates give us a very close estimate of our position, they are not always perfect. After some research it was clear that civilian GPS while in good conditions could return results within a metre, other factors that affect the accuracy are present on campus. Looking at figures, most GPS queries return accuracy to within 3.5 metres. However the impacting variables of receiver quality and signal blockage are not something we can plan for, due to these reasons a series of decisions were made.

As can be seen in the figure above, with the boundary between some buildings being so small, an error of 5 or so metres, something that has been seen, would provide false readings that would not be updated until the next cycle. Due to this the implemention of updating values on the Help page was omitted, the text displayed clearly displays the building the help is for, if this is wrong a button to update values has been included.

3.2 Building Display

During implementation a large amount of change was made to the way that the Building Display worked, a lot of this was down to it sharing an Activity with the Route Plotter. Initially this was done as the activities used similar resources and seemed small enough to not cause too many issues. However into development it was decided that the two activities really should have been separate, if nothing else than for clarities sake. However this was not decided until the features were complete, so as the project stands the screens have not been split.

3.2.1 Removal of Render functions

In the program the largest change is the removal of the $renderPlotter$ and $renderDisplay$ functions. In past documentation these were said to be the methods that would actually render what was needed, however these have been removed and the problem handled in the $onCreate$. This is done by the press of either the Building Display Button or the Route Plotter button initially sending through a string which represents the users desire. From there the $onCreate$ is split, if the user has said they want to plot a route then a different part of the method is entered, in both cases the visibility of a set of buttons and other interactive elements is set. By doing this we can view the activity as having two $onCreate$ methods, its just that they are nested within the same parent method.

3.2.2 Other Changes

A change seen previously is the removal of the $calcDistance$ function from this class as well, it now relies on the class that handles all of the distance algorithms. A similar change is the removal of the $loadPlaces$ function. Loading of places is handled by the actual Place class, called locations, in the application. By being moved away we keep the code in this class purely related to the displaying of locations, removing any blur between the classes. $showByType$ is implemented in the suggested way by the class diagram, it is called using the $setOnClickListener$ functions used in the $onCreate$ function. These listeners set the behaviours for an interactive element in the design, in this case the $showByType$ function is called with the buttons type being passed in as the string variable. For showing Lecture Buildings the String Lecture is passed in and the array searched for any location that matches that information. $showByType$ was actually initially implemented all in each $onClickListener$, while no real problem for the user it meant the method was implemented for however many types of building there were. When the design was finally adhered to it vastly simplified the process and created a much better class in terms of code quality. This also means that in the future any addition to the type of buildings will only need a few lines of code instead of the repeated fifteen or so which were initially being used.

Throughout development other applications and documentation were reviewed which showed the use of custom markers for buildings, due to this the feature was also included within the application. With it only taking a small time and the result being a more customised application it was seen as a good time to improvement ratio.

3.3 Route Plotter

Route plotting was an area in the application where a fair amount of issues arose during implementation, while the feature was successful many hard decision had to be made along the way and the result of those will have to be tested. With once again using GPS co ordinates for the positioning, problems arose with accuracy and exactly how a user would want to plot a route. Most of the class diagram relating to this feature is correct, the only difference is the movement of the log point method into the $onClickListener$.

3.3.1 Plotting of points

Plotting of points is an area which had similar issues to the updating of values in the Location Based Help feature. Mainly the testing of automatic point logging, this turned out to be a much bigger problem than in the help section. With the error in gps co ordinates sometimes being over 3 metres the automatic logging of points is not something which could be trusted to return viable results. Tests using automatic logging returned results, which when visualized, returned results similar to below. Any stop in a users movement would cause problems as any change GPS made then would cause a small difference meaning that it looked like the user was taking an erratic route.

As can be seen from the pictures above this was not an ideal way to implement the feature, what is now done is a point is only added when a user asks for it to be logged. Having the feature implemented this way means the user is free to explore paths, yet not plot them. Providing what is seen as a more user friendly experience. Another benefit found was that by using the Google Maps $locationListenerEnabled$ the user would be able to see their position, however the benefit goes further than that, in locations with good signal the point is plotted exactly at the displayed position. It is thought the reasoning is that both Google Maps and $logPoint$ method are both using a Location Manager which will always return the same results to both as its running on the same device.

3.3.2 Visualizing and Cancelling

Visualizing a route was implemented in the way detailed throughout the application, poly lines are drawn between each consecutive point causing a route which displays the users position throughout. Within the application every time a point is logged, except for the first time, the $paintRoute$ function is called, it takes the latest and previous logged point and simply paints a line between the two.

An implemented method which was not expected to be within the class is the $showDialog$ method, this is run when the user wants to cancel a walk. Absence of this method in the class diagram and other specifications is due to what was a lack of research on the possibility of a dialog box. It was wrongly assumed that the displaying of one would be smaller than it actually is, however the method is fairly simple. On the user wanting to cancel a walk, all points are wiped and the map cleared, on choosing no nothing is done. A further decision made here was whether to return the user to the data entry screen to reset the walk, however it is has been currently decided that a user cancelling a walk is more likely to be due to an error than anything else. Due to this route information is persistent unless the user chooses to go backwards to the data entry screen.

3.3.3 File Saving

Saving the plotted route to a file was a task which took much longer than expected, this was due to a misread of the documentation. Through research it was found that an Android device typically has two types of storage, internal and external. It was assumed that external storage would only be possible with an SD card in the device, this was the first mistake which caused the long delay. A second compounding error was the assumption that saving to the internal memory would make the file accessible by the user, something which is only possible in cases where a device is rooted.

These compounding factors caused the process to take a length of time which simply was not expected and set the development off by around two working days.

Chapter 4

Testing

Detailed descriptions of every test case are definitely not what is required here. What is important is to show that you adopted a sensible strategy that was, in principle, capable of testing the system adequately even if you did not have the time to test the system fully.

Have you tested your system on real users? For example, if your system is supposed to solve a problem for a business, then it would be appropriate to present your approach to involve the users in the testing process and to record the results that you obtained. Depending on the level of detail, it is likely that you would put any detailed results in an appendix.

The following sections indicate some areas you might include. Other sections may be more appropriate to your project.

4.1 Overall Approach to Testing

4.2 Automated Testing

4.2.1 Unit Tests

4.2.2 User Interface Testing

4.2.3 Stress Testing

4.2.4 Other types of testing

4.3 Integration Testing

4.4 User Testing

Chapter 5

Evaluation

Examiners expect to find in your dissertation a section addressing such questions as:

- Were the requirements correctly identified?
- Were the design decisions correct?
- Could a more suitable set of tools have been chosen?
- How well did the software meet the needs of those who were expecting to use it?
- How well were any other project aims achieved?
- If you were starting again, what would you do differently?

Such material is regarded as an important part of the dissertation; it should demonstrate that you are capable not only of carrying out a piece of work but also of thinking critically about how you did it and how you might have done it better. This is seen as an important part of an honours degree.

There will be good things and room for improvement with any project. As you write this section, identify and discuss the parts of the work that went well and also consider ways in which the work could be improved.

Review the discussion on the Evaluation section from the lectures. A recording is available on Blackboard.

Appendices

Appendix A

Third-Party Code and Libraries

If you have made use of any third party code or software libraries, i.e. any code that you have not designed and written yourself, then you must include this appendix.

As has been said in lectures, it is acceptable and likely that you will make use of third-party code and software libraries. The key requirement is that we understand what is your original work and what work is based on that of other people.

Therefore, you need to clearly state what you have used and where the original material can be found. Also, if you have made any changes to the original versions, you must explain what you have changed.

As an example, you might include a definition such as:

Apache POI library The project has been used to read and write Microsoft Excel files (XLS) as part of the interaction with the clients existing system for processing data. Version 3.10-FINAL was used. The library is open source and it is available from the Apache Software Foundation [2]. The library is released using the Apache License [1]. This library was used without modification.

Appendix B

Code samples

2.1 Random Number Generator

The Bayes Durham Shuffle ensures that the psuedo random numbers used in the simulation are further shuffled, ensuring minimal correlation between subsequent random outputs [6].

```
#define IM1 2147483563
#define IM2 2147483399
#define AM (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774
#define IR1 12211
#define IR2 3791
#define NTAB 32
#define NDIV (1+IMM1/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0 - EPS)

double ran2(long *idum)
{
  /*-------------------------------------------------*/
  /* Minimum Standard Random Number Generator        */
  /* Taken from Numerical recipies in C              */
  /* Based on Park and Miller with Bays Durham Shuffle */
  /* Coupled Schrage methods for extra periodicity   */
  /* Always call with negative number to initialise  */
  /*-------------------------------------------------*/

  int j;
  long k;
  static long idum2=123456789;
```

```
static long iy=0;
static long iv[NTAB];
double temp;

if (*idum <=0)
{
  if (-(*idum) < 1)
  {
    *idum = 1;
  }else
  {
    *idum = -(*idum);
  }
  idum2=(*idum);
  for (j=NTAB+7;j>=0;j--)
  {
    k = (*idum)/IQ1;
    *idum = IA1 *(*idum-k*IQ1) - IR1*k;
    if (*idum < 0)
    {
      *idum += IM1;
    }
    if (j < NTAB)
    {
      iv[j] = *idum;
    }
  }
  iy = iv[0];
}
k = (*idum)/IQ1;
*idum = IA1*(*idum-k*IQ1) - IR1*k;
if (*idum < 0)
{
  *idum += IM1;
}
k = (idum2)/IQ2;
idum2 = IA2*(idum2-k*IQ2) - IR2*k;
if (idum2 < 0)
{
  idum2 += IM2;
}
j = iy/NDIV;
iy=iv[j] - idum2;
iv[j] = *idum;
if (iy < 1)
{
  iy += IMM1;
}
```

```
  if ((temp=AM*iy) > RNMX)
  {
    return RNMX;
  }else
  {
    return temp;
  }
}
```

Annotated Bibliography

[1] Apache Software Foundation, "Apache License, Version 2.0," http://www.apache.org/licenses/LICENSE-2.0, 2004.

This is my annotation. I should add in a description here.

[2] ——, "Apache POI - the Java API for Microsoft Documents," http://poi.apache.org, 2014.

This is my annotation. I should add in a description here.

[3] H. M. Dee and D. C. Hogg, "Navigational strategies in behaviour modelling," *Artificial Intelligence*, vol. 173(2), pp. 329–342, 2009.

This is my annotation. I should add in a description here.

[4] S. Duckworth, "A picture of a kitten at Hellifield Peel," http://www.geograph.org.uk/photo/640959, 2007, copyright Sylvia Duckworth and licensed for reuse under a Creative Commons Attribution-Share Alike 2.0 Generic Licence. Accessed August 2011.

This is my annotation. I should add in a description here.

[5] M. Neal, J. Feyereisl, R. Rascunà, and X. Wang, "Don't touch me, I'm fine: Robot autonomy using an artificial innate immune system," in *Proceedings of the 5th International Conference on Artificial Immune Systems*. Springer, 2006, pp. 349–361.

This paper...

[6] W. Press *et al.*, *Numerical recipes in C*. Cambridge University Press Cambridge, 1992, pp. 349–361.

This is my annotation. I can add in comments that are in **bold** and *italics and then other content.*

[7] Various, "Fail blog," http://www.failblog.org/, Aug. 2011, accessed August 2011.

This is my annotation. I should add in a description here.