.... Beginning of the documentation part .... End of the documentation part .... Beginning of the documentation part .... End of the documentation part ....

MapWars: Location-Aware Multi-Player Mobile Game

Final Report for CS39440 Major Project

Author: Luke Ward (luw9@aber.ac.uk)
Supervisor: Reyer Zwiggelaar (rzz@aber.ac.uk)

25th March 2013
Version: 1.0 (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 ....

Abstract

MapWars is the culmination of an investigation into the feasibility of creating a location-aware, multi-player real-time strategy (RTS) game for a mobile platform. Consideration about which platform to support is presented with the final decision being to focus on Google's Android operating system.

The game combines strategic game play against multiple players in a persistent environment, where game play is based on map of Earth and users are restricted in game to their real world location. Giving each user the capability to create their own army and battle against other players within their local area.

At the same time consideration was made to protect each users privacy and create a fair gaming experience. Thus different techniques on how to prevent crafted requests designed to circumnavigate limits placed in the Android client.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Background & Ojectives

1.1 Background

Real-time strategy (RTS) games have a huge market on desktop environments, but have yet to make the break through into the mobile gaming market. This is mostly attributed to the complex control mechanisms that need to be executed precisely. A mobile devices form factor restricts the number of controls that can be presented to the user at any given time as well as the precision in which these commands can be issued.

RTS games are generally designed around large, expansive maps that cover a large range of environments and landscapes. The game map shapes how the game will be played, how involved the player feels and ultimately the engagement of players. These maps can take years to develop and result in one of the largest costs within the development process. So when coming up with an environment why not take advantage of a ready made one? That of planet Earth.

Building a game where the game play takes place on a map of Earth has a number of benefits. First is the scale of a map covering 510 million square kilometres of varying terrain and features. This scale also contains a huge level of detail that could not be achieved by a team of designers. Along side this there is the feeling of familiarity, unlike when starting a new game and having to teach the player about the environment, the user will already have a well formed model in their own head. Along with highly detailed knowledge about certain areas especially those within close proximity of their current location.

Mobile devices have a whole host of unique features and sensors that set it apart from other gaming platforms. One feature that has been widely adopted in a huge range of different applications is location. A users location can be easily determined with a number of different components present on most modern smart phones, these are a GPS chip and the mobile GSM network. Although location has been used extensively in applications it has yet to be utilized effectively as a key metric within a game. This extra information about the user would enable a game set around the physical world to be able to be integrated with the user. Instead of placing new users randomly on a unfamiliar map they can be placed in their real location. Thus giving them a chance to use their own knowledge of their surroundings to help them within the game environment.

1.2 Market

A worldwide RTS game that combines a map of the world and data gathered about the user combine to create a unique gaming experience and could add an entirely new dimension to the genre. This can also reduce the learning curve and thus shorten the time between installation and engagement. With the sheer number of applications available on a mobile platform combined with the ease and minimal to no cost of installing new applications, this is important to reduce the chance of the user simply finding an alternative.

1.2.1 Choosing a platform

The current mobile platform oligopoly results in their only being two viable platforms, Apple's iOS and Google's Android OS. The Android operating system, as of February 2013, had over 51% market share in the US [6]. Closely followed by the original and most established app ecosystem of Apple iOS. Even though there are other platforms, such as Windows phone and Blackberry, none of these have the market share allow them to be a primary platform of choice. Therefore choosing between the two platforms can be an important decision for mobile developers, both platforms offer their own unique advantages and disadvantages. Android on the one hand has the lead on market share offering a larger audience, although this will not directly affect engagement or return.

This years Developer Economics report [7] looks at all aspects of the mobile app market and is an invaluable insight into both the market and consumer trends. As well as evaluating the current market it also gets feedback from developers as to their experiences with different platforms as well as their success.

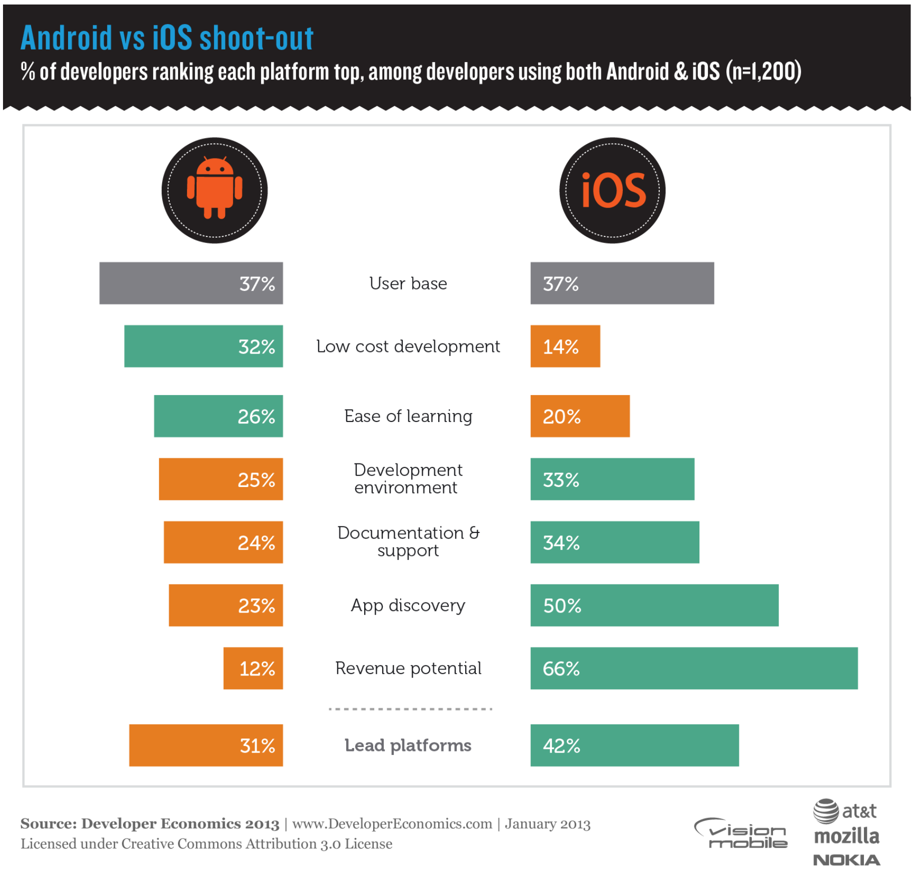

Figure 1.1: Developer Economics report [7] shows iOS as the leading platform among developers

As shown in Figure 1.1 iOS does indeed have a number of advantages over Android. Most importantly is the drastically increased chance of discovery as well as the expected return from applications on the iOS platform. For this project however less emphasis was placed on the success of the finished application and more on the process of creating it. Therefore the metrics of development cost and learning curve were considered to be more important. Aside from the Developer Economics findings there were a number of other more fundamental reasons as to why Android was the preferable platform. There was a certain amount of background knowledge and previous experience both with using and developing for the platform. The development environment had already been investigated and a small application developed prompting confidence in the ability to create and deploy something. Also a reasonable selection of Android devices were available for developing and testing purposes, which was favourable to using emulators and a single test device which would have been available for iOS.

The final consideration, which will be explored further in the following subsection, is that of not just overall market share but the scale of competition. IOS tends to attract the gaming market and as a result has a large selection of highly polished, top quality gaming titles. Whereas the Android market has a considerably smaller selection of games and these tend to be more basic and less graphically pleasing. Although this is changing as game developers migrate to Android it does give an advantage to new developers that might not be able to compete with the quality of iOS.

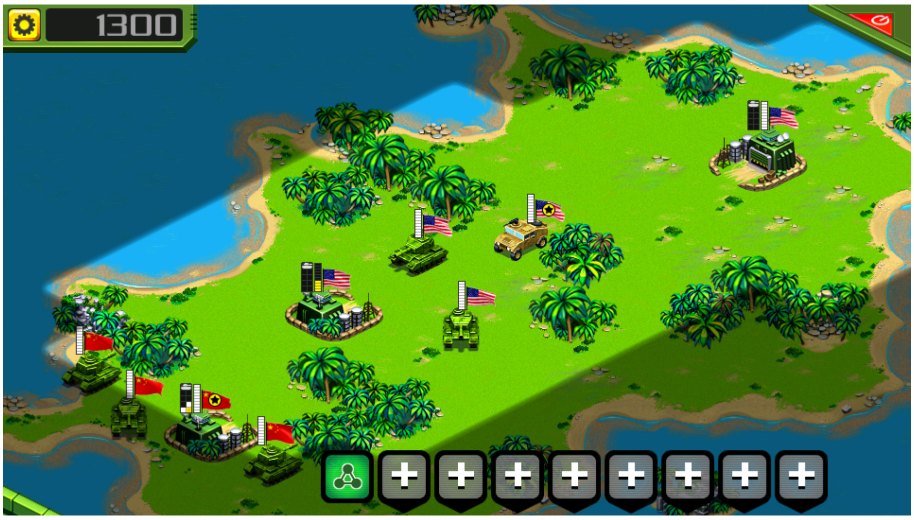

Figure 1.2: Tropical Stormfront from Noble Master Games

1.2.2 Android RTS Games

There is currently only a small handful of RTS games available for Android and of these most are unsatisfactory attempts. That being said there are a few well executed examples with the best examples coming from Noble Master Games (http://www.noblemaster.com/) who focus entirely on making these types of games for both desktop and mobile. The Stormfront series (http://www.operationstormfront.com) consisting of Tropical Stormfront and Desert Stormfront are the closest Android offerings to a full RTS gaming experience. Originally released for Android both were later ported to PC, MAC and Linux then subsequently iOS, this was made possible due to the game being built with OpenGL. These games offered the best chance to evaluate what had been done before and see what is expected and what would work well for MapWars. By examining different aspects from game play to control options it is a good opportunity to pick up ideas for the UI and user interaction.

Tropical Stormfront was their first release for Android in December 2011 and for iOS in November 2012. It took the RTS experience and simplified it to aim it towards the casual gamers that make up a large percentage of the mobile market. Its simple sprite style isometric graphics, shown in Figure 1.2, are pleasing on the eye and make units and structures easy to identify. The uses of an isometric view also removed the complexity presented by 3D graphics, that could it make it difficult to control, while retaining a well designed style.

The game revolves around a series of short predefined missions. Each mission only takes a matter of minutes to complete and don't really require a huge amount of strategy this caters nicely for the casual gamer. There is also an option for multi-player games with a selection of modes ranging from skirmish to capture the flag, with one user hosting the game. The host is able to choose the game type, map and a number of other options, other users can then join the game.

Unit control for example is cutback, users can control where units should move to as well as which enemies to attack but no other complex commands could be issued. A very useful feature was the ability to group units together and assign said groups to hot-keys along the bottom of the screen. Without these it was difficult to command a number of units in a controlled manor. As well as these assigned groups there was also the option to select units individually, by tapping that unit, or multiple units. Multiple unit selection was handled by tapping the green icon to the left of the hot-key buttons. This then enabled the user to drag over an area units selecting all within that area. It was important for MapWars to retain this level of control while retaining its simplicity. It could even be simplified further by removing manual targeting of enemy units, leaving this up to the games AI system.

Another aspect that is missing is the ability to create and position buildings. Each map comes with a predefined set of bases that can not be altered or moved but enemies can capture other players bases. These bases give players the ability to create new units and are also the mechanism for getting money. New units are built in a fairly standard process of selecting a base which in turn gives a range of units to build. Units cost a certain amount of money or resources to be built and the build process takes an amount of time to complete. After the unit has finished being built it exits the base and can be interacted with as usual. The lack of build queue, in which players can make a list of which units they want built and they will be processed in turn, was found to be restrictive. Inclusion of a build queue seemed plausible while still maintaining the ease of use.

As mentioned previously in game currency is generated based upon the number of bases a player controls. Each base earns $50 per minute, so the more bases captured the more money earned which in turn allows for building more units. Resource discovery, gathering and management are key aspects on PC based RTS games but could present problems when being implemented on a mobile device. By removing these aspects the developers were able to speed up game play and in turn try to make the game more exciting. Unfortunately the direct correlation between number of units and victory made game play feel predictable as the player who captured the most bases inevitably won.

Tropical Stormfront takes a good approach that, as with a number of the other compromises, directs the game towards casual gamers by removing some of the complex strategy decisions. They focused on fast based game play by setting the player up with a particular scenario with an established base and units with close proximity to the target. MapWars, although sharing many similarities, will ultimately be seeking a different target audience and game play style. The main aspects to be taken forward into the design of MapWars are:

- Simplify the process of issuing commands to units
- Ability to select multiple units at once
- Clutter free UI
- Cater for the casual gamer while accommodating those who want something more engaging
- Build queues

1.3 Objectives

MapWars is going to attempt to focus on tactical game play with a mobile experience tailored to the users location. All game play will take place on a portion of a world map depending on the players location. WHile taking as many of the tactical aspects from existing PC RTS games with an elevated focus on resource gathering and long term tactical game play. Unlike some games that allow for fast paced ”rushing” (a tactic in which a team will attack an opponent as quickly as possible as to surprise and overwhelm them) MapWars will encourage players to find and gather resources while building a substantial base and selection of units. This is in part due the sparsity of opponents in such a large environment. To try and compensate for the widespread nature of players, as well as the desire to recreate a real world experience, all players will play in a single persistent game. Being persistent all units will be visible even if their respective players are not online increases the emphasis on building a well formed and fortified base.

Problems may be presented with early adopts and those in remote areas who will ultimately not come across other players for considerable portions of their experience. This will undoubtedly lead to players getting bored and leaving the game. To try and combat this possibility game play should try and be enhanced for non-offensive play. The focus on resource gathering, exploration and base development hopefully will be the main combat to this problem, giving players the ability to enjoy the game even without an opponent. Secondly the persistent and location-aware portions will help the game change and evolve as players move around. For example if a player from an unpopulated area spends their time enhancing their units could then travel to a more active area, such as a city, to engage with opponents.

1.3.1 Primary Objectives

The objectives listed are considered to be the minimum required to create a playable game:

- Ability to authenticate users
- Display all units on a map centred on the players current location, limited to a given range
- Location will be determined by all available sensors and the most accurate to be used
- Appropriate portions of the application will stop functioning when minimized, to preserve battery life and minimize data usage. The application should then return to a playable state when returned to
- Resource gathering
- Enable players to create units, within a given range of their current location
- Enable players to move units, within a given range of their current location
- Units will engage with enemy that come into range automatically

1.3.2 Expanded Objectives

If the primary objectives are completed before the hand-in date the following objectives will be considered. Each of these extra objectives are not essential to the project but would add extra depth to the game play and ultimately improve the overall experience.

- Altering the resources generated by each mine based on real world resources in that location
- Introduction of environment variables such as power requirements. Each structure will add to the energy requirements, if these aren't met the structures will not function.
- Ability to upgrade units and structures
- Extra unit and structure types

- Path finding that follows physical routes e.g. roads utilising a directions API
- Offline notifications

1.3.3 Limitations & Evaluation

Working with a mobile platform comes with a number of technical challenges and limitations. Most notably is battery life, modern smartphones are capable of a multitude of tasks that increases the demand on the phones battery. This combined with the lightweight, small factor demanded by consumers the battery life is servilely limited. While developing MapWars it was important to keep this in mind and reduce the battery usage where possible. Certain areas that large power savings can be made, such as reducing the use of GPS sensors, can have a detrimental affect on accuracy or responsiveness. For this application the accuracy of the users location as they move around was not of a particular importance as it was used simply to locate them in the game area and not displayed directly. Therefore, with some small experimentation, the frequency in which the location was queried was tweaked to get a satisfactory balance between power usage and accuracy. Redrawing the screen is another drain on the battery so these should be kept to a minimum. The drawback with reducing the number of redraws is the possibility that the on screen movements of units and animations will become jolty and detract from the experience. Again experimenting with the delays within animation threads and reducing unnecessary redraws helps to keep the action smooth.

With over a thousand unique devices each offering a unique subset of sensors, screen sizes and features it is impossible to predict what features will be available on any device that will be running the application. For this reason it was important to take advantage of all possibilities while being aware of the limitations that may be presented. If a device does not support GPS or has it disabled, for example, it was necessary to find an alternative source of location data. Each source of data offered different levels of accuracy so again it was necessary to take this into consideration when determining which source to trust.

The success of this project will be evaluated primarily by it's ability to perform the tasks set out in the main objectives set out earlier in the document. Unit responsiveness when created and moved must be quick and consistent across devices. When a unit is moved on one screen the accuracy in which this is reflected on all other devices within proximity needs to be examined, with the smaller the difference resulting in a positive evaluation. With such a type of game this responsiveness can negate from the users experience and if it's too slow will frustrate and anger the player. Advantage can not be given to users based on their choice of device and connectivity. The applications consideration for the environment, as explained briefly above, is also a critical point to evaluate. As well as just the battery usage of the application the amount of data sent and received also needs to be kept at a minimum.

These objectives are the only the basis and do not give a complete picture of the desired outcome. As well as these objectives the finished application must play well and be a pleasurable experience. These are difficult requirements to quantify and will rely heavily on user testing and feedback. By distributing the application amongst users unfamiliar with any previous incarnations will be the best measure of it's playability. Their ability to play the game without external interaction and their overall enjoyment are the most valuable measurements available.

Previous metrics are focused primarily on the clients representation of data received from the server, the server it self needs to be evaluated. The server is required to be able to handle multiple concurrent connections each issuing commands that may or may not affect the other connected users. It must also be able to control unit movements and automated targeting and attack mechanisms. All while validating users actions and preventing spoofed requests, crafted to give the user an advantage over other users using the Android client.

Chapter 2

Development Process

Due to a strict deadline and extensive possibilities the project offered it is important to choose a development life cycle that could both quick and flexible. It needed to be able to allow for rapid development in a controlled manor that would reduce the need for rewriting and refactoring of code. Therefore the methodology will revolve around an iterative life cycle. As the project is easily broken down into a small number of well formed sections these iterations are able to be well formed before development beings.

2.1 Introduction

The development life cycle chosen is that of Rapid Application Development (RAD). This process reduces the need to have a detailed specification or design at the beginning of development. Instead specification, design and implementation are all simultaneous. This life cycle follows more closely to the Spiral Model than that of a more traditional waterfall type life cycle. The Spiral Model, as defined by Barry Boehm [5], depicts an evolutionary style of development which still keeps control over the project. Unlike the waterfall model which requires a stringent and detailed design that needs to be followed throughout implementation. Only the highest-priority features are considered for each iteration. The chosen feature is defined, designed and implemented during the cycle. Each cycle results in a prototype of a portion of the final system that can be tested. From this prototype ideas can be tweaked and changed then defined and implemented. Then the next feature can be defined and so on until the desired system is completed.

The RAD life cycle is a further streamlining of the spiral model and only has four distinct stages. Firstly there is the requirements gathering stage where all parties agree on the scope and requirements for the overall project. This is followed by user design where users interact with the developers and create a set of prototypes that are then evaluated by the users. This step is continuous phase that sees the prototypes change and adapt to the users requirements. This phase works in tandem with the construction phase which sees the prototypes be integrated into the final application and tested. With the user still actively involved they can suggest changes and improvements as the application takes shape. After the application is completed the final stage sees its testing, integration and user training.

2.2 Modifications

The spiral and RAD models are defined as an evolutionary method therefore each iteration prototype should be delivered to the stakeholders. These stakeholders can then give their feedback which should be utilized in the next cycle. Seeing as this project does not have any stakeholders this crucial step can not be completed and therefore the model needed to be adapted from a evolutionary to an incremental one. To achieve this the stages were mapped out before implementation began resulting in a slightly increased amount of design. These stages were purposefully left as broad as possible to accommodate necessary changes that would present themselves as development began.

As well as removing the of reworking code by cutting out the feedback loop for each prototype, the user design and construction phases are combined. This single step would see prototypes being worked directly into the current code base. In this way extra development time would not need to be used to integrate the prototype back into the master branch. This combination by itself would slowly reduce the code quality seen across the application as ideas are tested, for this reason spike work is encouraged to be carried out before each iteration. These short spike work sessions should show up any problems and highlight more efficient ways that part of code could be implemented. This will then carry over into the main code when the feature is implemented. It is important to keep this spike work short and focused and full solutions are not the ideal outcome from it.

These modifications were made as a way to streamline development, reducing time spent in design and coding while not reducing code quality or flexibility.

2.3 Development Environment

It was decided to use the Eclipse IDE as the main editor partly due to it's popularity and abundance of information and also it's improved integration with the Android SDK. Google provide an Eclipse plugin called Android Development Tools (ADT) which provide a set of tools to streamline the development process.

Eclipse was only selected for the Android portion of development for the reasons stated above, for the remainder of the coding and other things Sublime Text 2 was picked. It offers syntax highlighting for all the most common languages as well as a vast array of plugins and features to speed up development time.

Version control was handled by Git providing both data assurance and branching. Git was chosen over other systems, such as SVN and Perforce, offering a more versatile workflow allowing for local version control with changes being pushed to the master instead of other more centralized systems. To utilize Git to it's full potential a web-based hosting service was used to both host the code and to add extra features such as issue tracking and wiki-style documentation. Both BitBucket[1] and GitHub[2] were used initially but GitHub was favoured over it's more comprehensive set of tools. Particularly useful was the wiki that was automatically setup allowing for detailed documentation and research. GitHub also offers a number of visualisations to easily view commits history and network showing the different branches and merges.

[1] `https://bitbucket.org/`
[2] `https://github.com/`

The use of version control also helped with the rapid prototyping methodology by utilizing branching effectively. For each new feature being developed a new branch would be made where this development took place. Once development was complete these changes were tested and then merged back in to the master branch.

2.3.1 Hardware

Android offers a huge variety of hardware and is not just restricted to mobile phones. The OS can now be seen on tablets, set-top boxes, smart TVs and netbooks. However, mobile devices still have 96% of the market share with tablets following being with 64%, other devices only account for 20% of the market. To match this and the expected target market development was focused on both mobile phones and tablets. Originally a Motorola Atrix (v2.3) was picked for the main development device but after the first few months a faulty digitizer demoted this to a test device. A HTC Sensation (v4.0) was procured to replace the Atrix as the primary device. Experimentation and implementation for bigger tablet sized devices was done on a Motorola Xoom 2 Media Edition (v4.0). Other testing devices include a Sony Ericsson ST25i (v2.3).

The main development machine was running a 64-bit Debian based OS. Server hardware of consisted a single virtual private server with 128MB RAM running Debian 5 32-bit.

Chapter 3

Design

Stratergy?

3.1 Overall Architecture

3.2 Data Structures

3.3 Class Diagram

hello

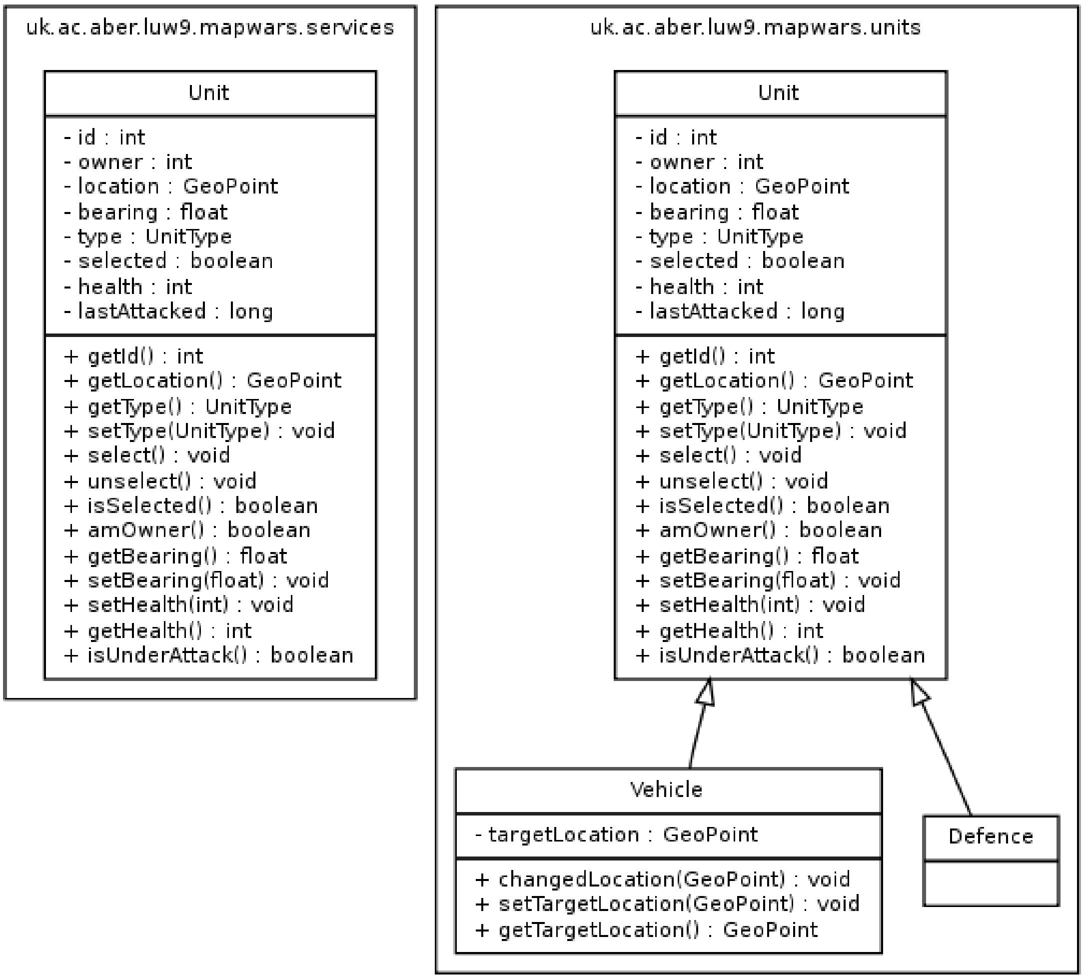

Figure 3.1: Client class diagram

3.4 Use Case Diagram

3.4.1 Language

Developing applications for the Android OS restricts the language choice to just Java combined with an Android framework. This results in code written with Java syntax but are not entirely synonymous with Java. Code is converted from Java Virtual Machine (JVM) compatible code to code that can be run by Dalvik, the vitual machine used by Android. This conversion optimizes the code to be run on devices with limited memory and processing power.

For the server portion of the application it was important to choose a language that would aid in the rapid development methodology outlined for the project. Python is a general purpose high-level programming language that has a relatively small learning curve as well as some personnel background knowledge. It is designed with the express purpose of being highly readable by forcing well formatted code and using English keywords.

As the chosen hardware for the server is limited in processing power and memory it is important keep overheads to a minimum, which Python does fairly well. If Java had been used a much more powerful server would have been needed to support the same number of processes due to the overheads introduced by the JVM. However there would have been a number of advantages to using the same language as the client. These include a greater understanding of the language as it is used more extensively but more importantly would be code reuse. The client and server both perform many of the same procedures and could have sped up development time and increased accuracy and interoperability as the same packages could have been utilized.

The server also required a persistent data store to keep an overview of the users and units between downtime or server migration. The requirements specified matched closely to those given when choosing Python as the programming language.

- Non-proprietary
- Simple, zero-configuration database
- Relational makes things easy
- Lightweight
- Previous knowledge of SQL makes that a preferable language of choice
- Simple python implementation.

A nice looking option was KirbyBase (http://wiki.python.org/moin/KirbyBase) it matched all of the requirements and being purely python based was a plus. It is a flat-file database so was simple to configure and lightweight, it also had the flexibility of running both as part of an application or in a client-server configuration. Unfortunately development stopped in 2006 and the libraries website has since been taken offline.

MongoDB

One long term problem that comes with SQLite is that it could not be run in a client-server configuration restricting the application to a single server. As the popularity and interest in the game increases so would the demand on the server meaning that a more complex, distributed server model would be needed. This is outside of the scope of the current project so this was deemed to be an acceptable compromise.

3.5 User Interface

When developing any application for a small form factor it is important to consider how the user interface can be minimized, and to not clutter the display making precise controls difficult. This is especially important when trying to fit all the functionality of a RTS game within the small form factor of a mobile device. To get around this problem only the essential controls will be included with only a portion of these being accessible from the main game screen.

Figure 3.2: Insert design of game map

Figure 3.3: Insert design of game map menu

Figure 3.3

Chapter 4

Implementation

Each iteration of the project will be described

4.1 Client Implementation

4.1.1 Mapping

The first iteration focused on the central component of the application, the map. It was vitally important that an appropriate mapping solution was used.

Google

Google provides a simple, easy to use interface to their own maps making it the obvious choice for any Android application. Their maps are accurate, up-to-date and very detailed.

Unfortunately there are a number of restrictions in place stopping their use in a number of situations. The most relevant of which is that they can not be used in an application that is not freely available to the public. Therefore restricting it's use in a paid-for application, such as MapWars may become. As the future of the application is uncertain it seemed desirable to steer clear of as many possible restrictions as possible. For this reason it was important to find a comparable alternative.

OpenStreet Map

OpenStreet map is an INSERT DESCRIPTION OF OSM HERE. It's growing popularity means that INSERT STATS ABOUT AREAS COVERED. With an acceptable level of detail combined with it's open SOMETHING(ethos?) made it the next most obvious source.

OSM has an API that allows it to be easily embedded into webpages but no native android SDK. A number of 3rd party libraries are available. The most complete and popular is that provided by MapQuest.

MapQuest are a mapping company that combines proprietary data and OSM data to create their own maps. They offer an Android SDK that gives you the option of which tile source to use. There are obviously restrictions to the proprietary data but if you opt for the free tiles then the same license is used as with OSM. The Android SDK available was designed to mirror the API available for the Google Map SDK. This made swapping out the Google Maps code and replacing it with the MapQuest code was trivial and problem free.

At the point in time of implementing MapQuest the design had called for the option to switch between satellite and road maps. MapQuest's main drawback, and more widely OSM itself, was it's lack of detail. The level of zoom supported was a number of levels less than that of Google Maps. These extra zoom levels would have made unit manipulation easier on smaller devices. Satellite images were the main concern as they were not available at the level of zoom required to make game play comfortable.

MapQuest was used as the mapping solution for a large portion of development and offered a stable platform. Once more of the functionality was in place user testing presented a number of problems with the map tiles being used. Most significant of which was a difficulty in being able to locate units among the details presented with the map. The sprites and colours being used to represent units were experimented with but none were clearly visible. The problem was with the design of the tiles being used and not necessarily the zoom levels present, although this may have helped alleviate the problems.

MapBox

One option available was to use a tile creator and host the map tiles on a server. This would be a costly and difficult solution to the problem. Hosting tiles is not a trivial task and require large amounts of storage and bandwidth.

MapBox offer beautiful hosted tiles. They also have their own software called TileMill which allows the creation of bespoke tiles based on any data source which can then be hosted and distributed via their network. TileMill was based on a CSS style syntax allowing you to customise any visual aspect, from line widths, colours, strokes. It also had the ability to import data from any source giving the ability to build up rich tiles with as much detail as required. For MapWars only the most basic detail was required while using a simple colour pallet. The idea was to make any unit stand out against the map while still presenting all the information required to orientate the user with their surroundings.

Tiles could be loaded from MapBox using a standard URI syntax used by the most tile vendors. This allowed it to integrate easily into any mapping framework.All that was needed was an SDK that allowed custom tile sources. Such functionality was found in OSMDroid. Like with MapQuest, OSMDroid followed the same pattern as Google Maps allowing it to be easily placed into the application without only one substantial problem. OSMDroid was missing one function that was supported by both Google Maps and MapQuest. These function was key in selecting units so had to be reimplemented ... which was not difficult but took time. Assumption was made it would be as effortless as the previous transition. After integration was complete plugging in the URI to my generated tiles was simple and worked straight of the bat.

MapBox did not offer satellite imagery but the beauty and simplicity of the maps being used made up for this. It was also decided that the complexity of such maps would just present the same images as found with the default OSM tiles. Satellite images could always be added to OSMDroid by simply finding a tile source and using that and would have no affect on the functionality of the application.

4.1.2 Location

Detecting the users location accurately is fundamental and is used directly in controlling game play throughout the game.

There are a number of difficulties presented when trying to accurately determine a users location. Firstly is the array of different providers available, each with their own unique characteristics, advantages and disadvantages.

Getting location updates within the application code was straightforward. Android exposes location updates by subscribing to providers via the LocationManager. A LocationListener can subscribe to any number of location providers and specify the frequency of updates required. Updates are sent after a minimum time and distance is reached. For this application battery life was more of a concern than getting high precise to the second data. So the minimum time between updates was set to one minute and the minimum distance sent to 2 meters. The minimum distance did not directly save battery but eliminates noise and recalculations based on small movements.

The network location provider uses the phones network connections to try and determine the users location. If they are connected via Wi-Fi the SSID (Service Set Identifier) is queried against Google's database of known SSID's, returning a location. This database is built by crowd sourcing data from Android handsets. When enabling location services on an Android device the following messages is displayed, Ällow Google's location service to collect anonymous location datä. Periodically the users location data from GPS, Cell-ID and Wi-Fi is broadcast along with details about available Wi-Fi connections to Google. This data tends to be fairly accurate, however when not connected to Wi-Fi the network location provider returns a location based upon Cell-ID. This technique identifies the mobile network providers cell the users is in. The location of each Cell-ID can be located so an estimated can be produced for the users location. Unfortunately cells can be very large making this method very inaccurate, however this tends to be around 1km and even less within rural locations [10] making it acceptable for this application.

When taking location data from a variety of sources it is critical to have a process to determine which is the most accurate to the users current location. The developers guide on location strategies [2] was an ideal resource to understand how each different provider worked and ultimately how to combine them to produce the best outcome. Each provider will gather data at different times and to a different level of accuracy. Whenever this data is processed a judgement needs to be made into whether this is more reliable than the currently known location. This was made based on its accuracy and age.

The process used to determine the most accurate location was adapted from an example provided in the developer guide. As shown in figure 4.1 each new location is compared against different aspects of the previously known location to decide if it is more or less accurate. If no previous location has been noted then the new data is taken to be the most accurate. If there has been a previous location the new location is only used if it is significantly newer else, if it is not significantly older, the accuracy of the two sources are compared. Each time a new location is received it comes with an estimated accuracy. This accuracy is measured in meters and is defined as the radius of 68% confidence [3]. Meaning that there is a 68% chance that the user is located within a circle with a radius of the accuracy, mapped around the supplied location.

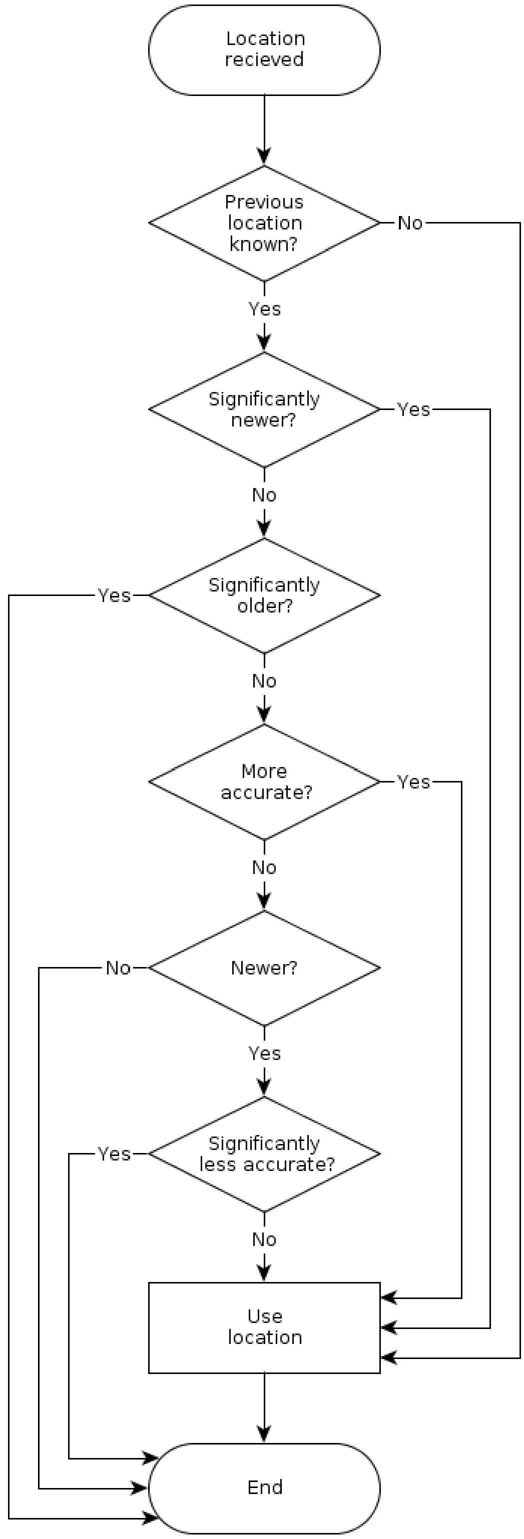

Figure 4.1: Location flowchart

4.1.3 GUI

Device Specific Layouts

Android layouts offer a variety of options to provide different layouts to different devices. The most basic of which is to make a fluid layout that stretches to fill the screen using `wrap_content` or `match_parent` widths. As well as these options there is a variety of dimension units to allow layouts adjust to different devices. The most useful of which are density-independent pixels (dp) which are units based on the physical screen size. These units relates to one pixel on a 160 dots per inch (dpi) screen. So the ratio of dp-to-pixel changes to match a devices dp. There is also a derivative of dp, scale-independent pixels (sp). Sp is the same as dp but also takes the users font size preferences into account.

Although flexible layouts allow for a single layout to be used across devices it does not allow for different layouts to be specified. This is possible using size-specific resources [4]. By placing layouts in a folder `layout-large` these layouts will only be used for devices classified as large, where any device below this category will use layout files in the standard `layout` folder. These size qualifiers are predetermined and static As of Android 3.2 new smallest width based qualifiers were introduced using the devices dp as a reference. Layout files placed in a folder `layout-sw<n>dp` are used only for devices with a minimum width of n dp. It is noted that a typical 7" tablet has a minimum width of around 600 dp, this is the size opted for in this project. As described in the design it was important to differentiate between the phone and tablet layouts as both offer users a different experience.

Figure 4.2: Isometric view of main game screen,
showing separation between components

While working on two separate layouts a large amount of code was being repeated and as features were added both layouts had to be updated interdependently. This seemed like a waste of time so a neater solution was investigated. A simple yet elegant solution was found that allowed different layouts to be combined together using the include tag.

For example the game screen added components to a basic map view, shown in Figure 4.2. The layers A and B are stored in separate files with two versions, one for each device. A simplified version of the file structure can be seen in figure 4.3.

```
res
 ├─ layout
 │   ├─ map_controls.xml
 │   ├─ map_header.xml
 │   └─ map.xml
 └─ layout-sw600dp
     ├─ map_controls.xml
     └─ map_header.xml
```

Figure 4.3: Simplified layout file structure

Device Based Orientation

From the design specifications for the GUI it was decided that devices of different screen sizes should not only have different layouts but also different orientations. Forcing an application wide orientation is straightforward. By setting `android:screenOrientation` to landscape in the `AndroidManifest.xml` file will keep the orientation landscape. Unfortunately there is not such a simple solution for setting the orientation based on device.

It is also possible to set the orientation within the application code. Each activities orientation can be set using `Activity.setRequestedOrientation`. Once it has been determined which device is being used it is then possible to set the relevant orientation, as shown in Figure 4.4. It is to be noted that this changes the orientation for this activity, not the application. As the application is built of multiple activities this needs to run whenever a new activity is created. All activities have access to the MainController and call `setActivity` on creation, so this seemed like the most logical point to do this.

```
if (Utils.isTablet(currentContext)) {
    currentActivity.setRequestedOrientation(
            ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
} else {
    currentActivity.setRequestedOrientation(
            ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
}
```

Figure 4.4: Setting screen orientation based upon device

The code behind isTablet() was based upon an online solution [1] to differentiate between big phones and small tablets. It appears that the best approach is to calculate the screen size by dividing the pixels by the devices dpi. A diagonal screen width of 7 inches was selected as the cut off point between classifications. Anything over this felt more comfortable in landscape orientation and are mainly operated in this manor.

9-patch Images

9-patch

4.1.4 Communication

PUSH/PULL XML/JSON

4.1.5 Units

4.2 Server Implementation

4.2.1 Echo

4.2.2 Units

4.2.3 Storage

4.2.4 AI

Chapter 5

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.

5.1 Overall Approach to Testing

5.2 Automated Testing

5.2.1 Unit Tests

5.2.2 User Interface Testing

5.2.3 Stress Testing

5.2.4 Other types of testing

5.3 Integration Testing

5.4 User Testing

Chapter 6

Evaluation

6.1 Primary & Secondary Objectives

6.2 Critical Evaluation

6.2.1 Known Bugs

6.2.2 Improvements

6.3 Summary

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.

Annotated Bibliography

[1] "Android - detect small tablet vs big phone?" http://stackoverflow.com/a/10080537/794119, 2012.

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

[2] Android, "Location Strategies," http://developer.android.com/guide/topics/location/strategies.html.

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

[3] ——, "Reference: android.location," http://developer.android.com/reference/android/location/Location.html.

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

[4] ——, "Supporting Different Screen Sizes," http://developer.android.com/training/multiscreen/screensizes.html.

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

[5] B. Boehm, "A spiral model of software development and enhancement," *Computer*, vol. 21, no. 5, pp. 61–72, 1988.

[6] comScore, "comScore Reports February 2013 U.S. Smartphone Subscriber Market Share," http://www.comscore.com/Insights/Press_Releases/2013/4/comScore_Reports_February_2013_U.S._Smartphone_Subscriber_Market_Share, 2013.

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

[7] Developer Economics, "Developer tools: The foundations of the app economy," http://www.visionmobile.com/product/developer-economics-2013-the-tools-report/, 2013.

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

[8] Handset Detection, "Where in the World are Android Devices Showing Up?" http://www.handsetdetection.com/blog/where-in-the-world-are-android-devices-showing-up-infographic/, 2013.

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

[9] MapQuest, "Android Maps API: Developer's Guide," http://developer.mapquest.com/web/products/featured/android-maps-api/documentation.

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

[10] E. Trevisani and A. Vitaletti, "Cell-id location technique, limits and benefits: an experimental study," in *Mobile Computing Systems and Applications, 2004. WMCSA 2004. Sixth IEEE Workshop on*, 2004, pp. 51–60.