

# **Project 288 – Multi-player Internet Game**

## **Final Report**

Author – Michael Marron C1115215

Supervisor – Andrew C. Jones

Moderator – Matthew J. W. Morgan

Module – CM3203 – Large One Term Individual Project

School of Computer Science and Informatics, Cardiff University

Date Completed 05/05/2015

Abstract – The focus of this project was to develop a networked multi-player game with elements of intelligent opponents, two-dimensional geographical game state and networking to implement a functioning game system for multiple players. This report documents the design, implementation and evaluation of this game system. The system consists of a Java server-client architecture implementing geographical tools to form a game area that can be utilized over a network.

# Table of Contents

1. Introduction.....	4
1.1 Goals.....	4
1.2 Approach.....	4
1.3 Outcomes.....	4
2. Background.....	6
2.1 Turn-based strategy games.....	6
2.2 Networking and other considerations.....	6
2.3 Geotools.....	7
3. Specification and design.....	9
3.1 Design Process.....	9
3.2 Coding Frameworks.....	9
3.3 Structural Planning.....	10
3.3.1 State Structure.....	11
3.3.2 Artificial Intelligence.....	12
3.3.3 Networking Structure and Threads.....	13
3.3.4 UI Design.....	14
3.3.5 Geospatial Data and Potential Features.....	14
3.4 Implementation Planning.....	15
4. Implementation.....	17
4.1 Implementation Process.....	17
4.2 Initial Framework.....	18
4.2.1 Server Framework.....	18
4.2.2 Client Framework.....	18
4.2.3 Game State Framework.....	20
4.3 Code Changes and Points of Interest.....	20
4.3.1 Connection details and Start Locations.....	20
4.3.2 AI solutions.....	21
4.3.3 Turn Progression.....	21
4.3.4 AI Overview.....	22
4.4 Geotools Refactoring.....	23
4.4.1 Geotools Overview.....	23
4.4.2 Refactoring serialization.....	24
4.4.3 Refactoring Interfaces.....	24
4.4.4 Geotools Evaluation.....	26
4.5 External Implementation Obstacles.....	26
5. Results and Evaluation.....	27
5.1 Original Goals.....	27
5.2 Game Testing.....	29
5.3 User testing.....	30
5.4 Test Suitability and Confidence.....	31
5.5 Evaluation.....	31
5.5.1 Project Strengths & Weaknesses.....	31
5.5.4 Evaluation of process.....	32
6. Future Work.....	33
7. Conclusions.....	34
8. Reflection.....	35
9. References.....	37
10. Appendices.....	38

10.1 Client UML.....	38
10.2 Server UML.....	39

# 1. Introduction

## 1.1 Goals

The goal of the project was to create a multi-player game to be played over a network. I chose to implement a turn-based strategy game consisting of players fighting over a two-dimensional area to compete against each other. Reminiscent of classic board games such as chess, risk and checkers and their more contemporary digital offspring of games such as “Civilization” and “Age of Empires”. As these forms of games have a substantial historical appeal and have evolved to expand into a wide range of interesting variants, I felt this would form a suitable basis for the game. Although there are many existing similar games of this type, I also felt it would be interesting to attempt to explore alternative implementations of these themes and attempt obstacles which some of these variants do not.

## 1.2 Approach

As I was given the choice of programming language in which to deliver the project, I chose Java. I considered writing the project in C++ or Python initially as the benefits of python's syntactical simplicity and C++'s powerful memory handling were not explicitly in line with the needs of my project. Although Java's memory overhead “huge compared to C or C++...the run time efficiency has become quite acceptable”<sup>1</sup>, I therefore decided that my experience in the specifics of Java would far outweigh any benefits additional memory might provide in such a small scale project.

The project needed to have support for networking protocols, and a strong set of libraries would be beneficial for the task's completion. Therefore, as the Java programming language possessed these and was the language I was most familiar with, I chose to use it to develop the project. Additionally, as I had experience with similar types of code which the project may possess, such as networking sockets, I felt it may be useful to use my coding history and experience in Java to develop the project. As Java was the language I had the most experience with and it had full support for networking sockets<sup>2</sup>, I felt it was a reasonable choice.

As I was producing a variant of an established genre of games, I also considered using existing engines and game creation software, which I will detail later in this report, but felt that it would be more beneficial for me to write much of the code myself to gain a better understanding of how it works at a more basic level, and allow me to create a more adaptable and versatile foundation of work to alter later on if I felt any larger changes needed to be made to the project. This was a decision which turned out to be very beneficial later in the project when I chose to implement geographical data into the two-dimensional geometry aspect of the game as the code was able to be modified as necessary.

## 1.3 Outcomes

With the final goal in mind, the intended audience could be more narrowly defined as a small number of users (roughly between two and ten) using the project over a small local

---

1 An empirical comparison of C, C++, Perl, Python, Rexx and Tcl for a search/string-processing program – Prechelt L. - <http://page.mi.fu-berlin.de/prechelt/Biblio/jccpprtTR.pdf> – 25/4/2015

2 All About Sockets – Oracle - <https://docs.oracle.com/javase/tutorial/networking/sockets/> - 30/4/2015

network or connecting via the internet to each other. I was also capable of making the assumptions that the users would possess at least basic usage of a computer, a working, networked group of computers and the ability to run the application on their computer. From these assumptions, I derived the expected outcomes of the project:

- A deliverable Java application of the game.
- An assortment of plans, evaluations, and test results documenting the progress of the project, which would then be compiled into this report.
- A personal development of skills from this project, and evaluation and reflection on how it developed and lessons learnt from this experience.

From these final outcomes, the final application was derived and created, and this report represents the knowledge, skills and experience gained from creating this body of work. This report also records the progression of the project's creation, from the background research, the design of the project and the considerations taken into the decision process, the implementation of the project and the obstacles encountered. The report will then explore my evaluations of the project's results and state before proceeding on to conclusions I have drawn from this project and my own personal reflection on what I have learnt from this experience.

## 2. Background

### 2.1 Turn-based strategy games

There are many two-dimensional strategy games in both digital and physical formats, often with wildly differing rules, play-styles and intended audiences. As a result, there is no current “best” form of this type of game and for the game I developed to be of value, I had to design it to be distinctive from these previous games. However, this vast and disparate range of games did provide me with various terms, ideas and established formats to draw from.

For example, for my specific game, I chose to develop a game in which the strategy is produced from the interaction between players' “units”, or individual pieces which they control as part of a larger strategy. Similar to the board-pieces in chess, in which the combination and overlap in possible moves and abilities provides a tactical library for each player to make decisions upon, and to provide constraints and a guide for the development of a strategy. Therefore, in this report I will frequently refer to “units”, single player-specific pieces which a player can control and may possess different abilities and position in the two-dimensional state of the game.

In my research, I also came across certain lessons learnt from the development of previous similar strategy games in regards to the specification of these types of units. For example, the simplest and most efficient way of creating diverse strategical range is to have a cycle of three units, forming a rock-paper-scissors relationship, in that each type of unit is strong against one other, and weak to another. This provides a simple, understandable relationship for the player to understand and to make intelligent decisions upon<sup>3</sup>. As a result, the first three units in my game which a player can use are “Infantry”, “Machine Gunners” and “Artillery” in which Infantry is strong against the immobile artillery, machine gunners are strong against the vulnerable infantry and artillery is strong against the embedded machine gunners. Although this mental mapping of game mechanics to visual imagery is largely aesthetic, it is functional in that it allows the players to more easily remember basic mechanics. It “saves players from having to remember an abstract relationship between button symbols and in-game actions”<sup>4</sup>; a substantial usability tool I found in my research of these forms of games.

### 2.2 Networking and other considerations

In the background research and planning stage of the project, I also had to consider areas such as networking and usability, and transferring the information of these areas to the player quickly and simply. For example, it is vital that each player knows how many other players are connected to the game and which units belong to which player. This then meant that some form of differing user interface or visuals need to be presented to each player, and this would further impact my design and implementation of the project. In response to this specific example, I resolved at the planning stage to send the same data from the server to each client, but to have it interpreted and displayed differently by each client so that each player's visual interface was player-specific.

In this background section it is also important to describe a few areas of methodology which are later used in the project. The first is that of Java's networking functionality. This can be handled without any additional libraries by Java's default TCP

---

3 Sylvester T. 2013 *Designing Games: A Guide to Engineering Experiences* O'Reilly pp180-184

4 Sylvester T. 2013 *Designing Games: A Guide to Engineering Experiences* O'Reilly pp237

and UDP sockets, meaning that very simple data can be transferred between two end-points on a network. This in turn means greater control on exactly what is sent can be developed, and gave me more freedom on what to send. I elected to use this simple data transfer method rather than a more complex external library as it would mean I would have much fewer constraints on what I could or could not send between players. Of TCP and UDP, I chose to use the lossless TCP transfer protocol, as in a turn-based game, I did not have a requirement for rapid communication of data between turns, but did require completely accurate information transfer between players at each turn to ensure the accuracy of the game. The guaranteed packet transfer of TCP filled both of these requirements as it "...provides a reliable, point-to-point communication channel that client-server applications on the Internet use to communicate with each other"<sup>5</sup>.

## 2.3 Geotools

Later in the development of the project, I developed support for geographical data to be implemented as the two-dimensional game state for players to play on, and as a result, I introduced the "geotools" Java code library into the project. Geotools is a powerful library for Java which allows the manipulation, display and analysis of a wide form of geospatial data. This data may range from simple two-dimensional points and their Cartesian distances from each other, to overlapping three-dimensional volumes of different locations. In this respect, the library is very complex and versatile piece of equipment. I elected to use it, rather than try to re-invent a method of geospatial analysis as it was capable of displaying the geographical data I required, whilst still retaining its semantics, such as distance between areas, overlaps of polygons etc. There were alternative Java libraries, such as 'spatial4j'<sup>6</sup> and 'geoapi'<sup>7</sup>, but these largely handle efficient analysis of geographical data and little visual representation, rather than filling the needs I had of very simple data handling and robust display.

Whilst this will be detailed later in the report, it is worth noting at this stage the nature of the geotools implementation into my project and the effects this had on the nature of the project. Initially, the implementation of the geospatial aspect of the project was intended to be an addition to an already functional system to simply change the two-dimensional game state being sent between clients and the server, although this in itself would require a sizeable amount of refactoring. However, during the incorporation of the geotools library, a series of large obstacles were encountered which greatly hindered the progress of this implementation and had wide ranging consequences on many separate aspects of the system. As a result, the UI, networking and AI aspects of the project required some refactoring. These will be covered in detail in relevant areas of the "implementation" section which outline how these obstacles were handled. However, throughout the earlier stages of this report I will evaluate how planned development and aspects of the system functioned and may refer to how they were later affected by the geotools merging, to represent the two states of the project, before and after the integration which distorted the project's progression.

Another point of external background information is that of the project management area of my project. In this matter, I scheduled weekly meetings with my supervisor throughout the duration of my project, to oversee the project's planning, implementation and evaluation. These meetings were very useful for the decision making stages of the project, and were a useful source of information and advice on more challenging obstacles. As a result, they had a strong guiding influence behind each section of the

---

5 All About Sockets – Oracle - <https://docs.oracle.com/javase/tutorial/networking/sockets/> - 30/4/2015

6 Spatial4j – About - <https://spatial4j.github.io/spatial4j/> -30/4/2015

7 GeoApi – Home - <http://www.geoapi.org/> - 30/4/2015

project's progress, and should be mentioned here as a source of third-party checking of the project other than just my personal project management.

## 3. Specification and design

### 3.1 Design Process

When specifying and planning out the project's progress, I quickly decided on the waterfall model for the progress of the project, to establish a week-by-week non-cyclic development progression. In the game design industry, iterative and agile development is a hugely dominant strategy to get rapid feedback on how a game plays and improve it accordingly<sup>8</sup>.

However, as I did not already possess a large number of testers to iterate through the project, and had a relatively small development time period, I felt that this method would take up too much time and had too many potential risks to make it beneficial. Additionally, the final requirements of a single development report also coincides more effectively with the evaluation final period of the waterfall methodology, making it more congruous to this method. Therefore, I chose to plan out my project in a simple chronological week-by-week format and to attempt to follow this plan for the duration of the project. There were some unknown factors, such as the utilization of the geographical tools later in the project, and how much the networking development would hinder the project's completion. These were taken into account during the planning stage and there were specific weeks in which decisions had to be made which could not be made at the start, which were based on the project's progress up to that point. In this way, the plan was not completely rigid in the initial planning stage, but did allow contingencies and alternative outcomes based on unforeseen obstacles.

The first step taken in the specification of the project was that of establishing the requirements of the project (as detailed in the background section of the report). This was done by deciding which aspects of the game I had in mind were key; those of being networked, multiple players and visually representing a two-dimensional game space. These were then reviewed with my supervisor to confirm their value and whether they were accurately defined enough to be measurable, but adaptable enough to be able to change due to the needs of the project, rather than be rigidly defined and immutable to the point of irrelevance. This process was effective as we established a good understanding of what the project would deliver and this created a sturdy set of requirements. These requirements were useful and defined enough that when the substantial geographical systems changes were implemented into the project, the requirements still held true and could be met despite the initial project implementation being vastly different from the final project.

### 3.2 Coding Frameworks

In the planning stage of the project, I also made several decisions about the physical coding process and how it would be handled. I chose to follow a similar waterfall liner progression at the coding level of the project, by planning out each section of code first, implementing it, reviewing it and then making any minor tweaks and bug fixes at the end, rather than other methods of iterating through code. I felt that using a particular framework to base my code off of, or borrowing many libraries from previously established solutions would not only limit what I could potentially complete with my project, but also harm the expected outcome of the project of the development of my own skills. By writing

---

8 Sylvester T. 2013 *Designing Games: A Guide to Engineering Experiences* O'Reilly pp283



the majority of the code myself, I believed I would increase my programming skills more effectively.

With hindsight, I believe that whilst choosing to write the project's code in this way did allow me to have a more clear understanding of what was occurring in my project, and potentially allowed me to avoid any semantically confusing bugs, this unseen benefit may have been countered by the relatively small scale of the project's final form and I would have liked to have developed more final features for my completed project. It may have been more beneficial for me to have a non-functional project of increased complexity, rather than a simpler working project. In addition, I have not much experience in coding in an iterative fashion or in a more agile development format or working explicitly from a rigidly defined framework, as may be necessary later in my professional career. So it may have been beneficial to gain experience in this format and evaluate my progress at the project's completion. Despite this, I feel that the choice of any of the coding strategies mentioned here have their strengths and weaknesses and I believe my choice was justified and beneficial as it was relevant to the specific needs of my project. As this project is partially a learning exercise and test of my ability, I felt that as much as possible, the project should be a product of my work rather than an external tool.

During my planning phase of the project I also decided to make explicit backups on a weekly basis to track my project progress and provide a fail safe in case any data loss occurred. On the project tracking area, I also made weekly notes on what had been completed, what large decisions were made and what bugs were encountered, to enable this report to be as accurate and detailed as possible. This decision seems to have been the correct one, as any time lost due to working on creating backups or documentation has been negligible in the total time-scale of the project.

### **3.3 Structural Planning**

One of the earlier areas of planning I performed was to concentrate the structure I held mentally of the project into a more specific tangible form. I did this by creating a UML model of both the client and server parts of the application to more concretely determine their abstract structure. This then allowed me to define their needed functions and variables at a basic level and use this as a framework around which to aim the project and more accurately plan it. The UML models went under two iterations as after I produced the first model, I felt there were many areas which needed further review and could be handled differently, as a result I possessed a much more accurate model upon its second iteration, that was very useful for defining the project's functions later on. The UML models can be found in the appendices at the end of this report.

Although two iterations did not fit in my initial planning phase of the project, after meeting with my supervisor we established that it was more important to create a resilient plan earlier on than to meet every target planning stage time with the specific deliverables at each project stage. So as a result, at the end of the second week I possessed a very accurate model but had to work slightly harder to meet the target deadline of my initial coding framework in the following weeks. I still believe that this was the correct decision, as the possibility of wasting time constructing an illogical project structure that may then have to be reverted later in the project's development seemed too great a risk to face.

As a result of the planning, I developed an abstract structure of the key areas of interest of the project as follows:

- A server application capable of storing and processing the game state for each turn of the game.
- A client application to receive the sent game state from the server and able to send information to the server via networking protocols

- A user interface for each client to display the game state and record and send their interactions with the game state to the sever.
- A networking section of the server for handling the connection and disconnection of each player as well as broadcasting information in the correct format.
- A simple artificial intelligence aspect of the server to allow it to play against the player if necessary, and also greatly assisting in testing and debugging of the game during development
- A two-dimensional form of game state held on the server. (This would later be adapted into the geographical data of the geotools library.)

This abstract structure of simplified blocks to work on formed early on in the planning stage was relatively strong and survived to the final form of the project. With the benefit of hindsight, I believe that isolating these specific areas to form a foundation of understanding of the project's structure and then applying abstraction to these rather than attempting to understand the entire project at once greatly assisted in constructing the plan for this project.

### 3.3.1 State Structure

The first consideration I took into account was the structure and form of the system when it is in its native or resting state, before considering how each individual function would work. Although this means jumping ahead in the design process to considering a functioning working system without any of its underlying components, I found it to be the easiest way to create a simple model to build towards. So in this example, I considered the server being connected to an unknown and variable number of clients which can be added to and removed as necessary. To store the references to each of these clients, I decided that it made the most sense to store the sockets and various connections to each of these clients in an arraylist. I also could have stored these in a hashset, as each client should only maintain one connection to the server and there should not be multiple of the same client, but I felt that the iteration through an arraylist would be easier and due to the inherently volatile nature of networking, it made more sense to store clients in a simple arraylist. I believed that the greater simplicity of the Java arraylist would allow me to deal with multiple incorrect client connections and lost connections more easily.

The other main aspect of state I was concerned about was that of the game state, the two-dimensional map or board on which the game was played, and the units for each player. As I was unsure as to how far this could be developed and improved upon later in the project, being the most extensible aspect of the project; I felt it would be wise to store all of these in a "Game State" object so that this object could be manipulated separately to the Server and Client-side inputs and outputs to this game state. This offered increased maintainability and extensibility throughout development, at a slight increase in complexity. I believe this decision was justified as I was unsure at the planning stage of the final development of this area of the project. Additionally, throughout the project I was careful to ensure that interaction with the Game State object was limited where possible to reduce coupling and support its modularity. At this stage I can also state that this choice was very useful during the implementation of the geotools library as this effectively meant replacing the entire game state object with a completely new object, and would have been impossible if it had not been as isolated as it had been.

Within the game state object, the game board was planned as a two-dimensional array of "Location" objects (a simple object containing it's own relative coordinates and basic information) with each Location also possessing a unit with all it's relative ownership

and possible actions. These actions were also defined by an “Action” object for each unit so that each unit simply held what actions it could, and would do. This structure at the time seemed the most basic way to define the simple game state of a two-dimensional unit based game. This system did have its flaws however; the use of a two-dimensional array did mean that the size and scale of the board, whilst storing relative distances and adjacency between locations, meant that the size and scale of the board was rigidly designed, and was more difficult to change. Also, each Location and Unit had to be repeatedly checked if it was null when processing possible actions and the actions of each unit, resulting in increased complexity. Having implemented this structure, and then refactoring it into a different form with the introduction the geospatial libraries, I would have designed it differently in future, with possible changes including a more dynamic array structure for multiple sizes of maps, and a less two-dimensional structure as determined by the two-dimensional array.

For example, in the final form of the project, each Location determines its neighbours by the Locations that share borders with the polygonal shape of each Location. This means that the number of possible Locations and their relative distance can vary hugely and share many different distance relationships. In the original two-dimensional array structure of the game, each Location would only have four neighbours and these all shared the same contiguous relationships with each other, greatly reducing complexity, but also resulting in a very limited possible list of interactions and strategies.

### **3.3.2 Artificial Intelligence**

As a product of the multi-player network aspect of the game, the question quickly arose during the planning stage of what to do with disconnected players, or if a player wished to play against an artificial opponent. Rather than have the game session be ruined or permanently halted by a player leaving, I chose to implement two contingencies to handle these scenarios. First was that a player could reconnect to a session and establish control of their own units again if they reconnected to the same session with the same details. The second of these was to implement an intelligent aspect to the game in the form of a simple AI. This simple AI would take over control of the units of a player who had left and give them at least basic orders to allow the game to continue to be played even when a player had left.

At the planning stage, I had not fully decided the extent to which I wished to implement this feature, as I felt it was a relatively simple implementation, having implemented simple AIs in other projects before hand. For example, I did not consider to what depth of strategy the intelligence should use, how defensively it should play or if it should just attack the nearest target. Similarly, I did not fully consider which algorithms or decision making strategies it should take to reach its targets. This then meant that I only implemented a very simple AI throughout the duration of my project. Although it was suggested that I could implement a more intelligent AI to play against, this objective was not fully explored due to the complexity of the geospatial aspect of the project. In hindsight, I also have know how difficult it was to re-factor the AI aspect of the game to negotiate the more complex game state of the geospatial data in order to make decisions. This suggests to me that although I could have designed a more complex and intelligent AI, it would have only been particularly strong at my specific variant of the game and would have been near-impossible to alter to function in a different game format (such as the geospatial data) due to its increased complexity. Therefore, I feel that pursuing the geographical aspect of the project and keeping the AI simplistic not only established a more obvious and visually clear body of work, which may have been hidden in the off-screen background of a more complex AI, but also allowed me to work on a more substantial project area.

### 3.3.3 Networking Structure and Threads

When designing the networking structure of the client-server architecture of the application, I decided that it should have the following interactions:

- A single server which clients can connect and disconnect from
- Multiple clients which can send their interactions with the game state to the server
- The server must also broadcast the game state to all clients at regular intervals

Rather than program the server to receive each change a client decides to make or not make to the game state and then broadcast these to every other client to update as soon as even a minor change is made, I decided to synchronise each broadcast with each turn. Therefore, each client would select their units and plot in their planned actions for that turn. Then, the client would hit a “ready” button, sending their requested changes in game state to the server. Then when every client was ready and had sent a requested change, the server would process all the changes to its local game state, and when complete, broadcast the results of that turn to each player. In this way, each client can interact with the most recent and consistent game state, whilst hugely reducing the workload for the server. Although it is unlikely that an increased server word load would affect the game, it could prevent any complex networking issues from arising and means that a stable and steady game experience could be created on even the worst possible connection.

To plan this out at a design level, I required a structure in which multiple clients can send their information to the server at any time, but with a centralised processing method for the server. After doing some research of simple Java networking applications, I found many applications used a series of threads so that I/O listeners could receive client information at any time, whilst the main server run a single thread of processing<sup>9</sup>. Therefore I chose to implement a thread based architecture for the server, with each thread stored in a dynamic arraylist. Each thread would then contain a client's connection details, sockets and listeners. For example, a server could be running three connection threads, with each connected to a different client and listening for data sent from that client. Then when a thread received data from a client, it would update its single main stored state accordingly. Although this structure is somewhat complex, it fulfilled all the requirements I had of the networking aspect of the application and did grant me a greater understanding and control of the basic data transfer between the server and client at every point in the project's running.

This networking structure did raise some other considerations to be planned for. The largest of these was the issue that there would be multiple different inputs to the server and these needed to be handled in some amalgamated way. In the example of chess, turns are sequential between players, so no player ever tries to move a piece that does not exist, but in this game, each turn for both opposing players is handled at the same instance before the result is broadcast to each player. For example, if both players decide for their units to attack each other at the same time and send their orders to the server, what should happen to each unit. For this particular example, I chose for each order to be processed as much as possible and both results to occur, so for two units attacking each other, both units would receive the damage, potentially resulting in both units being destroyed at the same time and being removed before the next turn. For two units' movement to the same location, this would be handled randomly, with the first processed unit moving to that location and the other unit doing nothing. All this result is non-deterministic and not ideal, I believe it is a rare enough scenario and the result is

---

9 Java Socket Programming Examples – Toal. R - Loyola Marymount University - <http://cs.lmu.edu/~ray/notes/javanetexamples/> – 30/4/2015

tolerable enough that this choice is negligible. If I had more time, I would have considered having the potential for multiple units at each location, stored in a dynamic array, but I believed that the possibility of stacking giant numbers of units in one place would harm both the visual clarity of the user interface system and potentially the desired player interaction with the game. When I progressed into the actual implementation of the area, the amalgamation of each unit's orders became a more tangible and known obstacle and was handled in more detail than in the initial plans of my design, so I will cover this later in the report.

### **3.3.4 UI Design**

The user interface aspect of the design was relatively simple. In this regard I chose to make the game board as the central visible aspect of the application, displayed as a two-dimensional grid with relevant units being represented by letters, numbers or any other placeholder image. Then I would display the other elements of the game (such as networking connection information, unit health and possible moves, and whether the player was “ready”) to the player in the forms of text boxes along one side of the screen. Although I could have designed a more versatile or visually impressive interface, I felt that making the game look as impressive as possible was not as important as getting it working, so decided to place less priority on this.

I felt that the built-in libraries of Swing and AWT were sufficient to create this interface and having some familiarity, I was confident that this would be one of the simpler parts of implementation.

### **3.3.5 Geospatial Data and Potential Features**

During the planning stage of the project, I discussed with my Supervisor the possibility of further extending the project's functionality if progress and completion of the application was smooth. It was established that as long as the possible additional features were established ahead of time and reviewed with my Supervisor, then a more reactive approach could be taken towards the project's final state. When deciding on potential future features to stretch the project's capabilities, I decided on two major areas. The first of these was to increase the complexity of the game board to use geospatial data, resulting in a more diverse and visually impressive game, the second feature was to create an extended artificial intelligence aspect of the game, capable of advanced decision making based on potential moves, and optimising to select the best possible move.

Even during the planning stage, the geotools aspect was given a degree of favour for multiple reasons; producing a “good” intelligence is notoriously difficult as even very complex AIs have difficulty making what a human considers to be an “intelligent” decision and often heavily rely on established problem-specific heuristics. This meant that if I did choose to attempt to implement a more intelligent aspect of the game, it would likely be less visually impressive and is already a very well-explored area of not only game design, but computer science in general. In comparison, the implementation of geospatial data into a game to provide a new game area to play over had the advantages of being a relatively niche market, and allowed further potential for expansion and unexplored material. In addition, it also provided an extra dimension of content to the game and whereas an artificial intelligence results in the same game but better, a game with real world maps as a basis for play lends the game a sense of mental context and significance. Therefore, at the planning stage it was established with my supervisor that the implementation of geospatial data into the game's design would form a suitable end-goal to aim towards.

### **3.4 Implementation Planning**

Towards the end of the planning section came the process of planning how the code would be practically interpreted from a UML model and structured into a functioning piece of software. For this aspect of production, I chose to divide up the software into all its main classes and objects, such as the server, client and game state, and create these objects in very simple forms as defined by the UML model and gradually increase their functionality in small modular functions. This extended to me writing code for each object, and introducing code stubs for all their obvious functions. For example, for the Server object I first introduced the function of broadcast(String Message) and checkTurnsLockedIn() to very simply establish the functionality the server would have at an early stage. I believe planning out the development of the project in this form helped to reduce "feature creep" that may come from attempting to write through each part of the project's objects in a more linear fashion. Additionally, I believed it helped with the planning and documentation stages of the project as it allowed the project to maintain a simple and modular structure in my head.

The planned implementation of the project was then to implement features in a linear fashion and test them as I progressed, evaluating if they were functioning correctly and repeat until all the stubs had been filled. This decision of having very short iterative testing cycles was made due to my experience of Java and knowing that once one had a simple compiled program, the compile and run times are very short and so rapid testing and evaluation can be performed on a short cycle. The alternative of writing rather lengthy sections of code and testing it all at the end was another possible choice, but I felt that writing a lot of long code and then testing it often leads to having to re-write large amounts of it in circumstances in which you are writing code that is not well-established beforehand. For example, if you know exactly what algorithm to write and implement it, then test it and find only a few bugs to fix, then it makes sense to write as much as possible before debugging. However, if you are in the situation I was in, in which you are writing code which has not been established before as having a "correct" solution, you may find the algorithm you chose to write not only has to be debugged, but is also the wrong answer. This was the particular circumstance when writing the game code, as it had no established model. Additionally I believe that writing and testing each individual section also allows a greater understanding of not only what each section of code is meant to be doing, but also knowledge that that is exactly what it is doing, giving the programmer a greater understanding of the application at every level.

The only remaining section of design and planning remaining was to establish rough guidelines on when specific modules of code were due to produce some sense of time-scale for the project. These decisions were largely from programming experience of how long specific sections of code should take and are more difficult to quantify. For example, I had programmed bits of networking in Java before and knew the difficulties that can come with managing and attempting to code and debug from both a server and client perspective at the same time, so gave this the longest coding period of implementation. My decisions can be justified however as with the exception of the implementation of the unknown geotools library, all of my estimates of when code would be completed were within a few days of accuracy. The specifics of these can be viewed in my submitted initial project plan.

Of great focus in this planning period was when to make large decisions regarding when to implement additional features, how to handle other contingencies and aspects of re-factoring and other sections of code management. To effectively define these obstacles, I met with my supervisor to define what time-scales seemed reasonable and when the latest possible decision could be made on whether I should implement geospatial data or

extended artificial intelligence. These planned out time periods turned out to be reasonable and correct, as they did allow me enough time to implement everything that was expected. Although there were aspects of the project I would have liked to pursue, as detailed in the "Future Work" section of this report, it was not expected that many of these would be covered at the time.

It is also worth noting, that contingency time was planned for if any major obstacles were encountered so that other sections could be moved around accordingly. For example, if the networking sections had taken a week longer than expected, the rest of the project had additional space to allow extra work to be done. In this fashion, when working at a steady state, the areas where I finished earlier than expected were balanced out by areas in which I finished with a delay. As an additional result of this cautious planning, when I did encounter difficulties with the geospatial aspect of the application, there was enough time to deal with these, and due to additional time reserved at the end for any refactoring or major clean-up from debugging that needed to be considered, I was able to fully deliver on this area.

## 4. Implementation

### 4.1 Implementation Process

The first aspect of the project's implementation to note is that of how closely the implementation process matched that as specified in the planning stage. For the majority of the project, the implementation closely followed its planned out counter-part. The well-established and simple functions of the server and client that sent and received messages all fit within their pre-defined function names and few additional functions had to be developed to cover the required functionality. For example, the client's functions are covered by three groups of functions: the networking functions such as `sendLockRequest()`, `receiveUpdate()`, `disconnect()`; the user input and unit handling functions such as `getSelectedAction()`, `cellSelected()` and `cancelCurrentUnitOrders()`; and the visual output and unit information functions: `setInfo()`, `updateUnitInfo()` and `drawLocal()`. All of these functions, were expected in the original plan of the project, and so at a functional structure level, the project did not deviate unexpectedly. There are a few exceptions, such as the Server's unit handling functions expanding somewhat to allow more complex intelligent functions than originally expected.

At a more basic level the question of how the implementation process occurred can be framed as whether the individual code was implemented, tested and evaluated as expected. In this regard, the planned process of implementing functions in their simplest form to quickly produce a testable system and rapidly testing each function as it was completed was largely successful. Each function was rarely revisited for re-factoring or debugging once it was complete, and formed a stable foundation to build on.

Similarly, the order in which functions were chosen to be worked on was cohesive to solid build up of code. Evidence of this can also be seen in one of my code comments from week 5:

```
//built basic units first  
//then turn implementation  
//then basic actions  
//then action handling  
//built in an order that supported testing whether current functionality was working  
//note this for future.
```

Meaning that I began with the basic Unit objects, then a turn progression, and then gave them basic actions to be processed at a turn end. In this way, I would not attempt to code a function which would alter a Unit's status before that Unit even had a status to begin with.

Coding in such a linear fashion did have the potential to introduce bugs later in development as hidden bugs may surface due to not being spotted initially and growing when exposed to increased complexity. This possibility could have been equally handled by applying unit testing to the functions at an early stage. However, I chose not to implement this as I felt that as this was primarily an educational exercise rather than a professional one, that if any bugs that could have been spotted by unit testing were created, it would be more important to catch them and handle them manually to gain a greater understanding of the cause of why I was creating them. In some ways, I wanted to create bugs to learn from them rather than rely on unit testing. In this way, I felt that being more involved with the code and being more responsible for each individual function would allow me to prevent similar errors in future. In a more professional environment or an



exercise where delivering functional code is more important than understanding the code itself, such as in a group or pair-programming exercise, I would have chosen to implement unit testing as a greater safety net and faster debugging solution. Coming to the conclusion of the implementation however, there were no bugs caused by failing earlier modules of the code and much of the debugging work was due to re-factoring and adjusting the code due to the implementation of the geospatial aspects of the project. This implies that my construction of the code at a simple level was robust and stable enough to be built upon in this instance.

## **4.2 Initial Framework**

### **4.2.1 Server Framework**

The main framework of the server is outlined as follows:

- A small interface window at set-up that establishes how many AI players to create and what map file to use.
- A server constructor that initialises client threads, stores them in an arraylist and awaits new connections.
- Client threads with separate Input and Output streams, and responses when sent data.
- Game State Functions for when data is received or connections are modified, called by client threads.

When evaluating this structure of networking, it seems relatively robust and it survived the development of the rest of the project from simple string messages to the more complex serialized objects and game state being sent between users. As the networking was the first major established area of the project, it had to be the most stable foundation part. However, this structure does have some flaws, such as it's memory usage and thread based architecture meaning that it does possess increased complexity and the potential for race conditions and out of sync data if the developer is not careful. Similarly, with a much larger user base, this format is unsuitable to ensure efficient and fast data transfer.

### **4.2.2 Client Framework**

The structure of the client is similar:

- An initial constructor to retrieve user variables of requested IP address and player name.
- A function to connect to the server, followed by a "ListenFromServer()" thread to receive and handle data.
- Text area boxes to display connection, unit and possible actions for a unit.
- A two-dimensional visual representation of the "board" taking up the rest of the screen, to display the game state with Java's built-in two-dimensional graphics libraries.
- Mouse and KeyListeners for recording user interaction.
- Functions for sending game state data to and from the server.
- Functions for retrieving specific unit actions and information.

This client structure generally translated well from its initial design into the first working form of the game, resulting in a simple, but functional display of the game. The lightweight and relatively simple Java swing interface seemed to be a good choice and with hindsight I still would have used it. However, it did become problematic during the implementation of

the geotools gamestate. The geotools library, whilst powerful, is particularly inflexible from a visual design perspective. Whereas the Java interface I constructed consisted of a window, with various modular components of a paint component to draw the map, and textareas and buttons as needed, the geotools library had incredibly limited functionality to create or use other Java components. The documentation and tutorials for the geotools libraries insist on implementing a geotools custom "MapPane"<sup>10</sup> window in which a map and toolbar are presented, which can not be displayed inside other Java interfaces or separated in other ways. So my original expectation that the geospatial data may be handled as simply another module to replace the paint component created in the initial working section of the game was no longer possible.

I spent some time trying to retrieve the visual data from the geotools library as it is translated from a simple file into a visual representation, but found that even when reducing it into its most basic form, it was rendered useless without the geotools specific visual rendering, resulting in a mistranslated series of points rather than a cohesive series of polygons. My second approach was to retrieve the geotools functions and objects that were displayed within the unique MapPane window to see if they could be placed into my original client interface in some form, however this was also unsuccessful, resulting in failures to compile. It became clear to me that the geotools library was designed, at least from an interface perspective, to function in a single, unmodifiable application and not in the more adaptable or modular fashion of other libraries I had previously encountered.

As a result, I had to work through the geotools libraries and tutorials to attempt to find ways to gain some control over the visual format of the geotools library and adapt my client to it. Eventually I established a compromise, rather than attempt to get the visual map data into my client, I would try the alternative of introducing my networking client into the geotools system. I therefore created a simple geotools MapPane window, and began to attempt to add my single components onto it. I was capable of introducing the text areas and buttons and their functionality into the geotools program so that they appeared within the map panel window. Although their presentation and alignment was distorted, it did mean that I was able to keep their functionality. I was then able to add in the networking functions, as these largely ran in the background of the client as back-end code. The final component to introduce was that of the Unit's two-dimensional representation from a simple x and y coordinate in their paint component representation to a longitude and latitude form within the map panel. This took considerably more time as I had to find ways to match a specific map polygon to its distorted and translated geospatial form, and then apply these same actions to a single coordinate. This was also managed to some degree and a new functional interface was created within the geotools library, albeit at the cost of visual appeal and control.

Overall, the client structure and interface was one of the more difficult aspects of the project when having to refactor it, however I believe this is in large part due to the non-modular design of the geotools visual library. Despite it being the most challenging aspect however, it did provide the most useful experience in order to test myself and explore previously unknown libraries. This particular challenge was useful as it allowed me to develop another mindset and tool for debugging and development, as my original assumption and mental structure of all the Java swing components in a hierarchical structure within one encompassing window is only a partial understanding of how the user interface could be displayed. It was equally a series of modular components and functions that form a single framework, separate from their exterior window and can be enclosed in any number of different environments.

---

10 Map Style Tutorial – Geotools -<http://docs.geotools.org/stable/userguide/tutorial/map/style.html> - 30/4/2015

### **4.2.3 Game State Framework**

The third element of the framework which was implemented was that of the game state, this closely followed the format as outlined in the planning section of the project, resulting in a simple chess-board type layout. This was also changed during the implementation of the geotools code however, as to retain the semantic information of neighbouring polygons and their relative positions, the units' positions had to be converted from a simple X and Y coordinate pair into their position as an expression of which polygon they were within. This also meant that code written referring a unit's action's range had to be similarly converted into establishing which Locations were adjacent to the unit's location, rather than they distance in terms of X and Y coordinates.

Although this refactoring to ensure the game still had a function was sizeable, it was relatively simple and quick to enact. This was mostly due to the simplistic structure of the game state at it's initial period. For example, when the AI was first introduced, a function "isActionPossible()" was created for the Action class to determine if a Unit's action was possible within its context before creating or submitting it. Although at an early stage this code was rarely called and functioned as an additional safety net, when the game state became more complex, updating the internal code of this function was relatively simple as it still emitted the same boolean output and allowed it to check more complex types of actions to prevent game bugs. These sort of occurrences, where older simple functions were extended to provide greater functionality were relatively common in the refactoring of the game state, and greatly reduced the potential workload.

## **4.3 Code Changes and Points of Interest**

### **4.3.1 Connection details and Start Locations**

One of the areas which was encountered during implementation and had not been fully considered was that of the connection details of each client. Although the normal operation of users connecting to a server and processing turns had been fully planned, the edge-cases of what to do with disconnected players or players with the same details had not. These areas are of some interest, as they required a faster decision and solution to avoid impeding the flow of progress. In this instance I chose to establish an additional arraylist of historical player names, in addition to that of the currently selected players. Then if a player connected, I would initially check if the name already existed in this arraylist so that a reconnecting player is assigned control of their old units and rather than creating a new unit for the player to use. On a similar note, if another player connects with the same name, to prevent both players being given the same unit the new connecting player is given the name with a number appended. So if multiple players connected with the name "Bob", the players "Bob", "Bob 1", "Bob 2", "Bob 3" etc. would be created. These solutions are relatively resilient in preventing the game from being irreparably broken by simple connection and disconnection actions and were chosen in a short space of time. Although the game is not completely resilient, a basic commitment to this area is at least beneficial in catching the majority of failure states.

A similar area of what to do with newly connected players was that of where to place their first unit when they join the game. During testing this was initially random, but when the two-dimensional grid was completed, it became clear that this would create wildly random and unfair games depending on start positions. As a solution, I specified that players on each team would start on opposite sides of the board, and would be spaced reasonably apart up or down each side. This meant that players would start in opposing

corners, the furthest distance apart, and as more players joined each team, they would be given start positions that were closer to the other team as their start position moved down the grid. This solution seemed the most practical, however, when the geospatial game board was introduced, this rigid concept had to be discarded and the random placement solution was re-introduced. Although this seemed a non-deterministic and chaotic choice, in the complex polygon nature of the geospatial data, the concept of absolute distance did not hold as much dominance as in a grid environment, due to the multiple overlapping paths between Locations. Although I would have liked to come up with a more elegant solution, such as specifying that each start location must be at a certain depth from other polygons to ensure a certain positioning, this was a relatively low priority to the rest of the project and had to be disregarded.

### 4.3.2 AI solutions

Another area of implementation of interest is that of the artificial intelligence and the problems encountered in its production. Most of these problems were trivial with simple solutions, but due to their frequency, they required a larger amount of implementation time than expected. Briefly covered, these included AI units attempting to enter locations blocked by other units, always performing the same strategy or quickest action, not finding the correct target or multiple units all targeting the same enemy unit resulting in a state of “overkill” in which every turn consisted of only one action repeated multiple times to defeat a single unit. The solution to most of these and similar issues was simply to create multiple possible options for the AI to choose from, rather than telling it to immediately choose the best action as determined by its heuristic. This further meant the AI was built towards decision making rather than pure calculation. From these possible actions, the AI would then disregard pointless or impossible actions and then either choose the “best” solution, or randomly select amongst equally scoring actions. Although this is the very basic interpretation of the intelligent aspect of the game, it was at least successfully implemented, and further extended the range of the project.

A specific example of the “intelligent” aspect of the game, was that of the first player Unit, the “Barracks”, capable of producing one of the three main combat units per turn. By simply tweaking the heuristic of which unit to produce per turn based on countering which enemy unit was closest, instantly allowed it to determine a very basic strategy to attempt to counter the enemy's choices. This then meant that it won the distinct majority of games against another, randomly choosing AI player.

### 4.3.3 Turn Progression

The easiest way to gain an understanding of the server's functionality is to consider the `CheckTurnsLockedIn()` function:

```
private synchronized void checkTurnsLockedIn(){  
    boolean allLocked=true;  
    for(int i = ClientThreadList.size(); --i >= 0;) {  
        ClientThread ct = ClientThreadList.get(i);  
        if(ct.lockedIn==false) {  
            allLocked=false;  
        }
```

```

    }
    if (allLocked == true){
        advanceTurn();
    }
}

```

This function is called when a client sends a lock request, indicating that it is ready for the next turn to occur. If the `advanceTurn()` function is called, the Server will process a turn before broadcasting this to all the players connected in the client thread array. Processing the turn will update the `GameState` object held centrally on the main Server, which then be referred to in the broadcast function and sent to each client. In this way, a single thread can be the last thread to become ready for a turn to occur and cause all the other client threads to be contacted.

### 4.3.4 AI Overview

The core of the AI aspect of the project can be viewed in the following form.

- AI units are selected at the start of turn progression
- The closest other enemy unit for each AI unit is found and stored.
- If the enemy is within attack range, an attack command is given.
- Otherwise the unit moves to the closest empty location to the enemy within movement range.
- If the unit can not move (in the case of the “barracks”, it attempts to create a unit in the closest empty location.

The “closest empty location” is defined by an exhaustive uniform cost search. The search state is given by the “`getPossibleEmptyLocations()`” function which provides an arraylist of empty locations arranged in breadth first search order. These are then iterated over by the following code to score them by their distance to the target.

```

    Location locationToUse = null;
    temporaryPossibleTargets.clear();
    getPossibleEmptyLocations(unitToUse, targetUnit, unitToUse.unitActions.get(j));

    int ShortestDistance=5000;

    for ( i = 0; i < temporaryPossibleTargets.size(); ++i ) {
        int targetDistance = (int)DistanceBetweenTwoPoints(targetUnit.X, targetUnit.Y,
temporaryPossibleTargets.get(i).xPos,temporaryPossibleTargets.get(i).yPos);

        if ( targetDistance < ShortestDistance){
            ShortestDistance = targetDistance;
            locationToUse = temporaryPossibleTargets.get(i);
        }
    }
}

```

This simplistic code always provides the optimal location for the unit to move to. As the search space is only one-turn deep, it typically consists of no more than twenty to one hundred possible choices, so for each unit this search space is easily manageable in an exhaustive algorithm. I would have liked to implement further depth to the possible moves and unit action choices, but did not have time. Similarly, a non-exhaustive A\* implementation would have been interesting, as it may not always be optimal, and so should respond in a more human fashion, but this would also have taken a large amount of time to implement.

## **4.4 Geotools Refactoring**

### **4.4.1 Geotools Overview**

The standard structure of the geotools' applications is as follows: a "Shape" file is the standard format for storing geospatial data in its simplest form, as a list of semantically encoded point, line or polygon areas. This file is read by an application and translated into a "SimpleFeatureCollection"<sup>11</sup>, a set of "Features", each "a geographic feature composed of geometric and non-geometric properties"<sup>12</sup>. These properties contain all the information about a feature, so for a polygon representing a country, this would detail its name, area and the points representing its border. Whereas a point representing a hospital would list its name, district etc. This collection is then iterated over and the geospatial data is interpreted by the geotools MapPane and applied through a "coordinate reference system" to translate it into a series of geometric points for display on a screen. This data can then be interacted with in various ways, to identify and analyse different features, perform geospatial functions, measure distances and so on.

When integrating this library into the code, I originally believed that it would be a simple matter of replacing and upgrading the GameState object with geospatial data and then modifying the outputs and inputs of this data to display on and interact with the already existing architecture. However this was not possible for a few reasons:

- Shape file data, once encoded, must be read in a special way to retain its order and structure. For example, reading it out through the geotools library allows the output of a constructed map or ordered points, whereas manually reading out the coordinates has them placed in a disjointed and different order.
- Many of the Feature objects used by geotools are not initially serializable, or must be overridden or converted into other objects for this to occur. This means that they can not be sent over a network socket in the same way another Java object, such as the project's GameState object, can be sent.
- The Geotools user interface and MapPane is one of the few ways in which the data can be displayed, and this Panel can not be contained within other windows in the same way as other Swing java interface components.
- Most displays of geotools shape files refer to longitude and latitude rather than simple raster X and Y coordinates as used in a more mathematical graph.

Although these problems were eventually overcome, the reason many of them were

---

11 SimpleFeatureCollection - Geotools Documentation -

<http://docs.geotools.org/latest/javadocs/org/geotools/data/simple/SimpleFeatureCollection.html> - 25/4/2015

12 Feature – Geotools Documentation - <http://docs.geotools.org/latest/javadocs/org/opengis/feature/Feature.html> 25/4/2015

encountered in this way was due to incomplete research of this library. During my background research of geospatial libraries for Java, the number of possible solutions to choose from was somewhat limited as this is not a particularly frequently-developed area.

So for the same reason that this area was chosen to be explored with my project, (it being relatively esoteric and unexplored and interesting to attempt) was the same reason that it was more difficult to accomplish. Due to the lack of Java geospatial solutions, choice was limited, with geotools being the best choice. Despite this, the documentation and support area of the library is somewhat limited. Whereas many other common problems for other libraries have many solutions and suggestions on support websites and forums, geotools is relatively unknown and has very few other developers.

A specific example I found of this during my project was that of a bug found in geotools in 2014 regarding using mouse clicks to select a Feature object as suggested in one of geotools' tutorials<sup>13</sup>. As of the time of the project, this bug had still not been fixed, resulting in an override having to be written for one of the geotools' library classes to fix it. This level of limited support demonstrates the way in which these problems were not only impossible to know when selecting geotools or any geospatial library to use, but also why the project ran into these difficulties.

#### **4.4.2 Refactoring serialization**

To allow the networking section of the project to retain its functionality, I had to develop a work around to geotools' Features being unable to be sent over a network. To do this I considered that I did not actually wish to send the Features details over the sockets, as both the client and server already possessed the Shape file at each end in order for the game to work. What I actually wished to send was the Units' positional details with respect to the Features, but without the Features themselves. Therefore, if I encoded the units into a serializable format, sent them, received them and then reinterpreted them into the Features in the same format, this would solve the problem.

To send data, the client or server would previously send a "NetworkObject" object which contained the entire GameState, of the game which held all the units, Locations and Features. I removed this GameState, and added a function to scan the current GameState of the sender for each Unit and compiled them into an arraylist, giving the Unit an X and Y coordinate representative of the centre point of the feature in which they were held. Then I would send this arraylist in the NetworkObject and at the receiving end I would interpret the X and Y coordinates of these Units so that the Unit was set to the Feature which held these points as its centre point. This meant I could effectively serialize just the Units I wished to send and have them appear at the receiving end and applied into the correct format. Whilst less efficient, and requiring more functions, functionality was restored and development could continue.

#### **4.4.3 Refactoring Interfaces**

The refactoring of the visual interface of the client has already been briefly covered with regards to the geotools implementation. However, the interaction with this interface, the selection of Units and Locations with a mouse cursor and the giving of orders also had to be refactored. In this regard, a cursor interface is implemented in the geotools MapPane interface and is capable of selecting a single point on a geographical map, by creating a small bounding box over the mouse cursor and selecting features within this area. As I

---

13 SelectionLab Bug- <https://jira.codehaus.org/browse/GEOT-4722> -25/4/2015

required more extended functionality of this interface: the ability to locate adjacent Features and their units, I had to establish a geographical query to locate all neighbouring, touching, containing and interior features to a selected Feature. To do this, I used the following code, with explanation highlighted:

```
while (iter.hasNext()) {                                //an iterator to check if each feature is adjacent
    SimpleFeature feature = iter.next();

    //Boundingboxes created around countries or features
    BoundingBox CountryBoxToCheck = feature.getBounds();
    Rectangle screenRect3 = new Rectangle((int)(CountryBoxToCheck.getMinX()), (int)
(CountryBoxToCheck.getMinY()), (int)(CountryBoxToCheck.getWidth()), (int)
(CountryBoxToCheck.getHeight()));
    AffineTransform screenToWorld3 = mapFrame.getMapPane().getScreenToWorldTransform();
    Rectangle2D worldRect3 = screenToWorld3.createTransformedShape(screenRect3).getBounds2D();
    ReferencedEnvelope bbox3 = new ReferencedEnvelope(worldRect3,
mapFrame.getMapContent().getCoordinateReferenceSystem());

    //A series of filters to specify if the two different features are adjacent
    Filter overlapFilter = ff.overlaps(ff.literal(bbox2), ff.literal(bbox3));
    Filter withinFilter = ff.contains(ff.literal(bbox3), ff.literal(bbox2));
    Filter withoutFilter = ff.contains(ff.literal(bbox2), ff.literal(bbox3));
    Filter touchFilter = ff.touches(ff.literal(bbox2), ff.literal(bbox3));
    Filter twoFilters = ff.or(overlapFilter, withinFilter);
    Filter threeFilters = ff.or(twoFilters, touchFilter);
    Filter allFilters = ff.or(threeFilters, withoutFilter);

    //Checking if all the features overlap within a minimum short distance of 30
    if( allFilters.evaluate( feature ) ){
        Geometry geom2 = (Geometry) feature.getDefaultGeometry();
        if (areaHandler.getArea(geom2) >30){
            adjacentFeatureArray.add(feature);
            IDs2.add(feature.getIdentifier());
        }
    }
}
```

This filtering code would check each geographical Feature to check if it fit the filters and was within an established geographical criteria, bordering the Feature selected by the mouse cursor. Similar code was implemented at the server side and when giving orders to check if the target of the Unit was in an adjacent Feature. In this way, the interaction behaviour was clunkily modified to fit the geotools library, but functionality of the game was restored.



#### 4.4.4 Geotools Evaluation

Having implemented the geotools library successfully into the project, many of the difficulties of the implementation could be examined and evaluated. Although many of these were triggered by the integration of the geotools library, this is not the sole cause of them. For example, the issues with attempting to send the non-serializable Features over a network only arose because I relied on a simple and less efficient method of sending the entire GameState rather than just the data I needed. Similarly, the display of the geotools data being prescribed to a single specific output window only caused issues because I had already coded an embedded output format rather than keeping this output and user interface entirely separate to the data being displayed. Many of these problems could have been avoided if I had constructed the code in a completely disjointed and modular way with every single function divided up into a much more decoupled state. However, this would have taken far more time and could have caused considerably more problems and code rewrites in order to create a perfectly extensible system.

From a theoretical point of view, the key underlying fault that caused many of the other obstacles was that I did not recognise that implementing a third-party library into an established system would be considered a refactoring task rather than a simple development task. The already written functionality did not integrate with the less modular and adaptable geospatial system, and this meant that the sections of code that clashed had to be ameliorated. Despite this, the integration of geotools was a very useful educational and experience-inducing exercise as in addition to the other benefits gained from working on a complex project such as this and overcoming the obstacles it created, I also gained a further understanding of geospatial software and the many challenges and intricacies of this area. From this experience I would state that the key lesson I learnt was not to try and make a library do something it is not explicitly designed to do, but to mould expected outputs or functionality to make best use of the library. The geotools library was incredibly powerful when it was tasked to do what it was designed to, but attempting to make it do something else was futile and wasteful.

#### 4.5 External Implementation Obstacles

For the sake of completion, some of the other obstacles encountered during implementation are briefly listed here. The first of these being a more exterior problem created when using third-party libraries. My personal preference when programming is to use a very simplistic development environment and IDE, such as the very simplistic notepad or equivalent, so that I have complete awareness of what code is being written and I find software attempting to predict what I am intending to do to be frequently incorrect and cumbersome. However, to use the geotools library, a large number of external libraries needed to be downloaded and installed into a specific format, along with relevant updates and other housekeeping. To do this, "GeoTools projects tend to use a large number of jars and an automated solution is preferable."<sup>14</sup> so it is suggested to use an IDE with netbeans and its build tool Maven being the most recommended. This caused a small delay in development as I had to move my previous code into the netbeans development environment and organise the effective download of the geotools libraries. However, this did provide me experience of working in another development environment that I may not otherwise have chosen, which is beneficial.

Another obstacle to overcome was that of the pure size of the geotools libraries. As

---

14 Geotools – Netbeans Quickstart - <http://docs.geotools.org/stable/userguide/tutorial/quickstart/netbeans.html>  
26/4/2015

it is a powerful piece of software designed for a wide range of functions, it understandably is quite large, extending up to 200Mb when fully updated. However, this meant that when developing, I had to consider where I could physically program as I would have to re-download many of libraries in order to fully compile the project. Additionally, the libraries would frequently attempt to update themselves, until they ran out of available disk space or attempting to use up an excess of memory. These problems arise with developing in any less-lightweight solution however and are not particularly unexpected, so I consider this a useful exercise for when developing in a professional environment using other, more complex, software solutions.

## 5. Results and Evaluation

### 5.1 Original Goals

Referring back to the original goals of the project, the key points to evaluate against were:

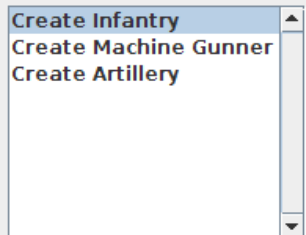
1. A server application capable of storing and processing the game state for each turn of the game.
2. A client application to receive the sent game state from the server and able to send information to the server via networking protocols
3. A user interface for each client to display the game state and record and send their interactions with the game state to the sever.
4. A networking section of the server for handling the connection and disconnection of each player as well as broadcasting information in the correct format.
5. A very simple artificial intelligence aspect of the server to allow it to play against the player if necessary, and also greatly assisting in testing and debugging of the game during development
6. A two-dimensional form of game state held on the server. (This would later be adapted into the geographical data of the geotools library.)

To test goals one, two and four, I ran a series of connection tests to have multiple game clients connect to the game server and progress a series of one hundred turns and sent game states to each client. For each turn I would then check that the turn had progressed for that client as expected, and measure the time taken to send the gamestate from the Server's system clock by outputting the "System.currentTimeMillis()" time taken for each "writeMsg()" operation. I did this in multiple environments to establish the resilience and efficiency of the network, the first of these being the client run on the same computer as the server. The next being the client and server on the same local network, specifically the University Linux labs computers labx03 and labx04. The final test was between a computer from the lab and an off-site computer at my house in Cardiff, connecting by IP via the internet.

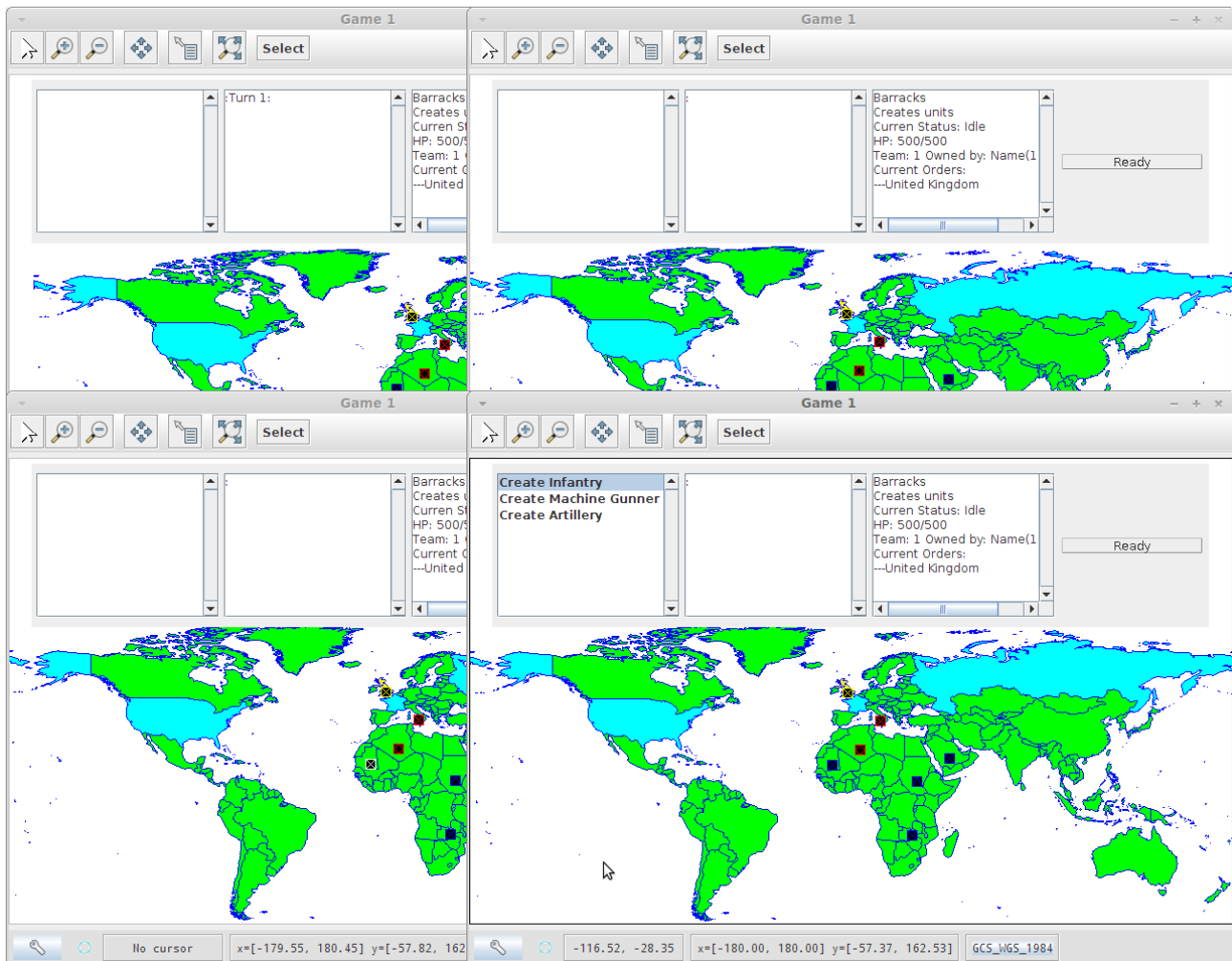
	Local Host	Local Network	Via Internet
Connection Established	Yes	Yes	Yes
Average Transmission Time (ms)	3.5ms	8.2ms	120.5ms

For goals three and six: "A user interface for each client to display the game state and

record and send their interactions with the game state to the sever.” and “A two-dimensional form of game state held on the server.” I arranged a series of tests to check if the relevant elements of the Client user interface display data as expected, comparing the data at the user interface to raw printed data held on the server outputted to command line.

Value to Test	Server Output	Client User Interface Element	Client value
Example Unit “FlavourText” variable	“Barracks”	consoleInfo Textarea	Barracks
Example Unit “currentHealth” variable	“500”	consoleInfo Textarea	500
TurnInfoString	“Turn 1:”	ConsoleOutput Textarea	:Turn 1:
Example Unit Possible Orders List	“Create Infantry Create Machine Gunner Create Artillery”	PossibleActionList JList	
Example Unit – Location – Property - LONG_NAME	“Saudi Arabia”	consoleInfo Textarea	---Saudi Arabia

To further test the output, to ensure both consistency and that each Client was receiving the correct data, I tested running multiple instances of the client on the same computer, with each different Client player only being able to issue orders to their specific units. In this example, the United Kingdom is selected by each Client for four clients, with all the clients returning the correct information for this location and relevant unit, but only the controlling client able to issue orders and therefore possessing a populated possible order list. Additionally in this screen-shot, it can be seen that the unit in Mauritania in West Africa is coloured blue, indicating it is owned by another player in all of the client interfaces except the controlling player where it is coloured white. These test results demonstrated to me that the client user interface both accurately sent the game state to each player and sending the player's interactions with that game state correctly.



To test the fifth goal, the intelligent aspect of the game, I tested both a mixture of having the game played against one of my peers, and playing the game against itself in an automated fashion. In the human versus AI test, the human was careful in deciding which unit to produce and ensuring the optimal routes and targeting for each unit. In this way, the human beat the AI player, as it was able to easily work out the AI's aggressive strategy of always sending a unit towards its nearest enemy. However, the AI did always produce a unit and would at least always attempt to attack successfully, so it did form the very basic process of a programmed intelligence moving towards a goal when given a differing environment. In the AI vs AI test, each game did eventually draw to a conclusion as each AI attacked each other and produced units until destroyed. This could be broken if given geospatial data in which two AIs had no possible route to each other and could not win, but in a gamestate that had a possible victory, the AIs achieved it. Similarly, with two

All players of the same “intelligence” the victory split on each team was approximately 50% with no bugs causing an unexpected dominance.

## 5.2 Game Testing

As a primary round of testing, I formed a check-list of simple game interactions and their expected outcomes, as a formalised system check of the game. The results are shown below:

Action	Expected Output	Actual Output
Initialise Client	Connection details window appears	Connection details window appeared with default fields
Enter Test name and connect by IP	Map file selection dialogue is shown	Map file selection dialogue is shown
Select Map File and select OK	Relevant map file is shown with a starting unit for the player in a randomised location	World map is displayed, with a white-coloured starting unit in Japan
Click on empty feature	Name of location is displayed in Info textarea and the feature is highlighted in yellow with adjacent features highlighted in cyan.	“---Brazil” displayed in textarea. Brazil polygon highlighted in yellow. Larger neighbouring South American countries highlighted in Cyan.
Click on starting feature	Unit Information and Name of location is displayed in Info textarea and the feature is highlighted in yellow with adjacent features highlighted in cyan. Possible Orders Optionlist also populated with possible unit actions	“Barracks Creates units Curren Status: Idle HP: 500/500 Team: 1 Owned by: Test Current Orders: ---Japan” displayed in textarea. Japan polygon highlighted in yellow. Russia, China and USA highlighted in Cyan.
Issue Order to create unit in neighbouring feature	Relevant Order added to Unit info text area	“Barracks Creates units Curren Status: Idle HP: 500/500 Team: 1 Owned by: Test Current Orders: Create Infantry from: 72,22 to 67,26 ---Japan” displayed in textarea.
Select Ready Button	Turn progression: unit info textarea refreshed and relevant unit created at target location. Turn info added to output console.	“:Turn 1:” added to output console. Relevant unit created. Unit info textarea remained the same until feature re-selected.

Following this testing, the only areas that failed the basic testing was a typo of “Current” on

the unit information dialogue, and that the client text area of unit information did not update on turn progression until the next turn. This indicates that for the most part, the game is fully functional and tested at a basic level, but fails slightly in the area of good usability and keeping client data up to date.

### **5.3 User testing**

I performed some basic “smoke testing”<sup>15</sup> with two of my peers to establish if there were any major flaws or problems with the project, by having them play against an AI player and I observed to see if they raised anything unexpected or drastically broken. After gaining a basic understanding of the game, both of the users very quickly made sure their units were always performing an action and were able to beat the simple AI. This rapid understanding of the game may mean that the game is too easy or the AI is not complex enough. However, it does suggest that the game adheres to the basic concepts of usability. It also shows that the system responds to the user's input in such a way that the user's expected output is given successfully and so the user can effectively build on the knowledge they have gained of the system.

The system was not completely without bugs however, and it was raised that one of the units in the game remained on the map after a turn had progressed despite having negative health but was subsequently cleared after an additional turn. Additional smaller bugs or preferred functionality was also raised. For example, with the shape file data I tested on, featuring a map of the world, certain countries were too small to be selected by the game, and certain countries were so large that they bordered many other countries, giving an advantage in strategy and also resulting in some odd AI movements. Although I did not have time to fix these bugs or append additional options for alternative functions, (more details on these will be listed in the future work section of this report) this did at least confirm the effectiveness of the testing.

### **5.4 Test Suitability and Confidence**

Although most of these tests are basic and do not perform the extensive and in-depth verification that regression or end-to-end testing would provide, they form a wide and diverse form of safety net to catch all major and obvious flaws. The addition of user testing as a form of black-box testing also prevents bias and obvious usability issues from being missed due to a pre-disposed mindset. Furthermore, automated testing from unit tests or by creating test scripts would be ideal to catch all the potential bugs created when allowing the user to input data, such as that of the map data selection and name entry, however the time investiture for creating these tests would have drastically cut into the time required to create the project. From a more theoretical perspective, the end-testing of this project is the demonstration of this project to the supervisor and moderator, rather than a public release of this software to a large and unfamiliar user base. As a result, I believe it is more important to test towards the requirements that the software has to fulfil. In this regard, I was testing to see if a basic game that can be used by a non-malignant and minuscule user-base, and I believe that these tests suit this purpose.

---

15 Smoke Testing – Software Testing Classes - <http://www.softwaretestingclass.com/smoke-testing/> - 30/4/2015

## **5.5 Evaluation**

### **5.5.1 Project Strengths & Weaknesses**

From an objective standpoint, the game itself does not compete in the same league of player experience, scale or engagement as that of other games. However, it does have a number of project-specific strengths and weaknesses. When measuring the project against its specified goals, it fulfils most of these to a high degree and was designed, constructed and released in a relatively short time-scale and within its targets. Amongst its strengths are that the game, for its most common functionality: a small scale networked game between multiple players, functions correctly. Additionally the networking code is relatively robust and able to connect a reasonable number of players to each other for a game session. Due to the resilient Java TCP code, and turn-based structure of the game, disconnections or slow connection speeds are not a factor, resulting in a very stable game. The use of geospatial data to create a game board to play on with real-world contextual information provides a degree of novelty and the simple controls are fully functional, providing at least basic engagement and curiosity of the game mechanics for a player. Lastly, as the project when tested fulfilled most of its test criteria, it can be stated that the project does have the strength of being reliable when placed under testing. There are however, many things that could be improved in the project, the specifics of how these can be improved will be listed in the “Future work” section of the report, but what these weaknesses are in their current state can be listed here.

The project does still possess some bugs as demonstrated by the project testing, specifically in the user interface area and in handling more complex geospatial game states. These could be made more resilient to handle more forms of data. Additionally, the AI aspect of the game is primarily basic and can be easily outsmarted by a human player, and is weaker than expected in the original goals. The user interface still has some flaws following the implementation of the geotools library, with the text areas and geospatial data overlapping as a result of the the geotools' libraries limited compatibility with other Java components. Furthermore, the functionality of the game server is particularly basic, with only single game sessions supported and only the very basic of connection protocols (by IP address) being provided.

From a testing and project management perspective, the project has little infrastructure for further maintainability and extensibility, due to lack of formal testing, unit tests, compartmentalised code structure, and official coding standards. Similarly, as much of the code relies on a third-party library and the support of these libraries, it also currently relies on the netbeans and Maven architecture to be run. With more time, a lot of these problems could be overcome, and the project is in a stable state to be updated and modified, without being in an invalid state, but for the moment these factors can be considered weaknesses to be improved upon.

As an overall evaluation of the project against its goals, in light of its strengths and weaknesses, I would state that although the project has a number of flaws and bugs in its design, infrastructure, code structure and final form, it still succeeds in the majority of its requirements and can therefore be evaluated to be a successful project.

### **5.5.4 Evaluation of process**

Although I have defended my design and implementation processes earlier in this report, they are by no measure perfect. Whilst I believe that they have been fit to purpose, for

example: testing towards requirements rather than full regression testing; they should be evaluated. My general process of time management was good as all mandatory goals were met to a good standard within time-scales. However, my research and preliminary checks into the unknown libraries of geotools were a key failing that cost the project many of its further goals and the more complex functionality of the intelligent aspect of the game. Additionally, the geotools implementation required a large amount of code refactoring that could have been avoided if the full extent and capabilities of the geotools had been known.

My knowledge of third-party libraries and solutions is an aspect of project management I wish to improve. It is, however, difficult to build upon in any other way than implementing a number of unknown and distinct libraries with the expectation that some of them are not fit to purpose and will fail in order to fully understand them. This leads firmly back to the key under-lying experience of this project's process and progression, that of responding to a distinct and inflexible external factor. Demonstrating that in terms of the internal aspects of managing a project and the technical skills of creating a project, I am capable. The next skill to learn is to apply this experience and tool-set to working on external challenges. This may be working with other libraries to produce a product, or working on a shifting requirement, with a large team or for a number of distinct clients.

## 6. Future Work

Throughout the course of the project, there were many areas which I would have liked to have progressed further or in a different way. The most drastic areas which need bug fixing or modification are those of the user interface and the game progression. In regards to the user interface, the text boxes which display game information are aligned incorrectly within the MapPane window and it would be preferred to have them either render correctly at an initial stage or have the possibility of hiding and revealing them as needed. Furthermore, the display icons for the game units are very simplistic, consisting of a single box on the screen with varying line patterns and colours to indicate a different type of unit. With more time, I would modify the "Paint()" function of the client to either display these as more complex images or simply iterate on the designs somewhat to make them more distinct and recognisable. A further avenue I wished to pursue in terms of visual clarity would be to modify the Paint() function to display a simple coloured, horizontal bar beneath a unit to represent its health to indicate which units have been attacked, and a line from each unit to its target order location to indicate its movements. These actions would have been relatively simple and quick to implement, but I simply did not have time to introduce and fully test them.

Similarly, with the existing game turn progression there are bugs resulting in certain units not being removed correctly or slight other issues when non-conventional shape files are used to create the game map. With some additional time and restructuring of the advanceTurn() function held on the server, these bugs could likely be found and removed with relative ease.

From the original specification of the system, there were a few features that were quickly cut, but I realised during the project's progression could be implemented into the project if given more time. The first of these would be a chat system, based on the project's original architecture and function. This could be added by adding an additional text area to the client to display this information and a text field to enter text into. Then with a client function this text could be sent to the server, broadcast to all players and then updated within the text area on each client with a slight variance to the receive function.



Although this functionality could be implemented in a few hours, it is off of the original specification and design, and may have unexpected repercussions or bugs; for example, where to display this additional text area on the client or sending incorrect network data.

The key aspects of the game functionality I would like to have improved are that of the starting position distribution of units in the geospatial context of the game. This is currently randomly assigned to compensate for the unknown nature of the map file being chosen for the game, however I would have liked to have changed this to either a choice by the player, or at the server level to allow more control over the form of the game. These modifications, whilst simplistic to describe, would require a substantial amount of work as they would require both new code functionality and user interfaces to implement this functionality.

The final element of future work to mention is that of the turn progression for AI units, currently the choice of how to progress these orders consists of selecting a unit from alternate teams in an order dictated by the way in which the geographical features are stored within the map file. Although this in practice has no impact on how the AI players act and how the game plays out, I feel that if further features were added or if this foundation were built on, this method would affect the game to some degree and ideally should be progressed in a more controlled, deterministic fashion.

## 7. Conclusions

To summarise, this project was to create a multi-player internet game and to do this I initially specified the exact requirements of this project into the main features of:

- A server application capable of storing and processing the game state for each turn of the game.
- A client application to receive the sent game state from the server and able to send information to the server via networking protocols
- A user interface for each client to display the game state and record and send their interactions with the game state to the sever.
- A networking section of the server for handling the connection and disconnection of each player as well as broadcasting information in the correct format.
- A very simple artificial intelligence aspect of the server to allow it to play against the player if necessary, and also greatly assisting in testing and debugging of the game during development
- A two-dimensional form of game state held on the server.

I then designed these components with UML models and into a framework of code stubs in Java, with relatively little external software, libraries and modules to retain more control over the project, learn from it and allow it the most potential to grow.

Implementing the code was a steady and expected process for the first form of the game, but to push the project into unfamiliar territory, the third-party geospatial Java library “geotools” was integrated into the project. This was to modify the two-dimensional game state from a simple grid into a much more powerful and complex spatial data representation. This choice had unexpected effects on the implementation of the interface, networking and many other aspects of the code, but was a beneficial experience and was eventually delivered.

There were some key areas of the project that were disappointing or problematic. For example, I would have liked to have implemented more features and fix more bugs, but all projects must eventually be shipped in some form, and rarely in a perfect state.

Similarly, I believe that some of my coding practices and project management decision making; particularly in regards to research of third-party libraries and unknown frameworks, could use a lot of improvement. However, these skills are improved with experience and hopefully this project has aided in that aspect.

Overall, I believe this project was successful, not just because I was able to deliver fully on a project that matched the original requirements, but also in terms of process, project management and learning experience. In addition to learning a new form of library and area of data management, producing a whole project from a blank state and learning how to overcome many unexpected obstacles; I also was able to appreciate a new development mindset. This was: that in order to overcome the constraints and restrictions enforced by an external library, I had to become more malleable in both my processes and in the skill-sets I was using. Therefore, I could not force the software to accomplish my goals in my specific way, but had to accomplish my goals using the software in the fashion it was designed to be used.

## 8. Reflection

Throughout this project, one of the key areas of change it has had on my personal skill-set is that of the impact the project has had on my assumptions. This can most clearly be seen in my attitude towards libraries and using external or existing frameworks to accomplish a goal. Originally I viewed these pieces of software as tools or existing solutions to save time to complete a goal. This is certainly the case with the incorporated libraries in many languages. However, this is not the whole picture, as many libraries are extremely powerful, but the power comes at a cost of modularity, efficiency or other resources. No software is absolutely perfect, and as a result, these libraries must be implemented in a more careful form. The first libraries I learnt to use were simple tools and so could be often be used as inelegant solutions. More complex libraries are more powerful tools, and must be applied in careful ways and become more fragile if they are attempted to be used for a purpose for which they are not fit.

Many of my other assumptions and experiences were altered by this project as well, such as my avoidance of unit testing, complex IDEs and other project management tools. Although I was able to proceed through this project whilst avoiding some of them, it was clear that if the project were any larger or more complex, these tools would have been vital to its success and so I should operate under the mindset that I will have to use them. This is bound to affect me later in my career if I ever perform any form of programming professionally; either as part of a team or when working on a larger scale, multi-dimensional project. Therefore, I have gained a valuable transferable skill by gradually transitioning into these new areas to gain a smoother introduction to them.

Amongst other new skills I have learnt are those of implementing new frameworks, creating a large project from scratch and independently implementing the entire development cycle from requirements analysis through to release and evaluation. This large body of work and all the smaller areas that come with it have attached a wealth of new tools to my skill-set and thus I feel it has been hugely productive. Many of these skills are transferable, to both a professional setting of programming, but also in many other aspects. For example, the scale of this project and the time management involved in ensuring its completion is of great use to any large project or adult responsibility, and fosters a new level of maturity. In many ways, this project creates many of the transferable “graduate skills”, such as self-management, independence, adaptive thinking, new experience, innovation and creativity that many employers look for, but also help create a more well-rounded person.

In addition to new skills, I have also practised some of my existing skills, to refine them into more versatile and effective tools. For example, I was able to demonstrate my project management and programming on this sizeable undertaking to deliver a large, cohesive and effective application. Furthermore, I have developed my knowledge of networking sockets, client-server architecture, interface design, geospatial systems and other areas introduced during my course to show that I have an understanding of these both in academic theory and in practice.

From an experience and lessons learnt perspective, I have gained a more well-rounded view of creating a project, as this project had areas of both substantial steady work, new unknown areas and areas that were problematic and required more innovative solutions. The design and specification areas of the project provided areas of project specification and requirements analysis, similar to those I performed during my industrial

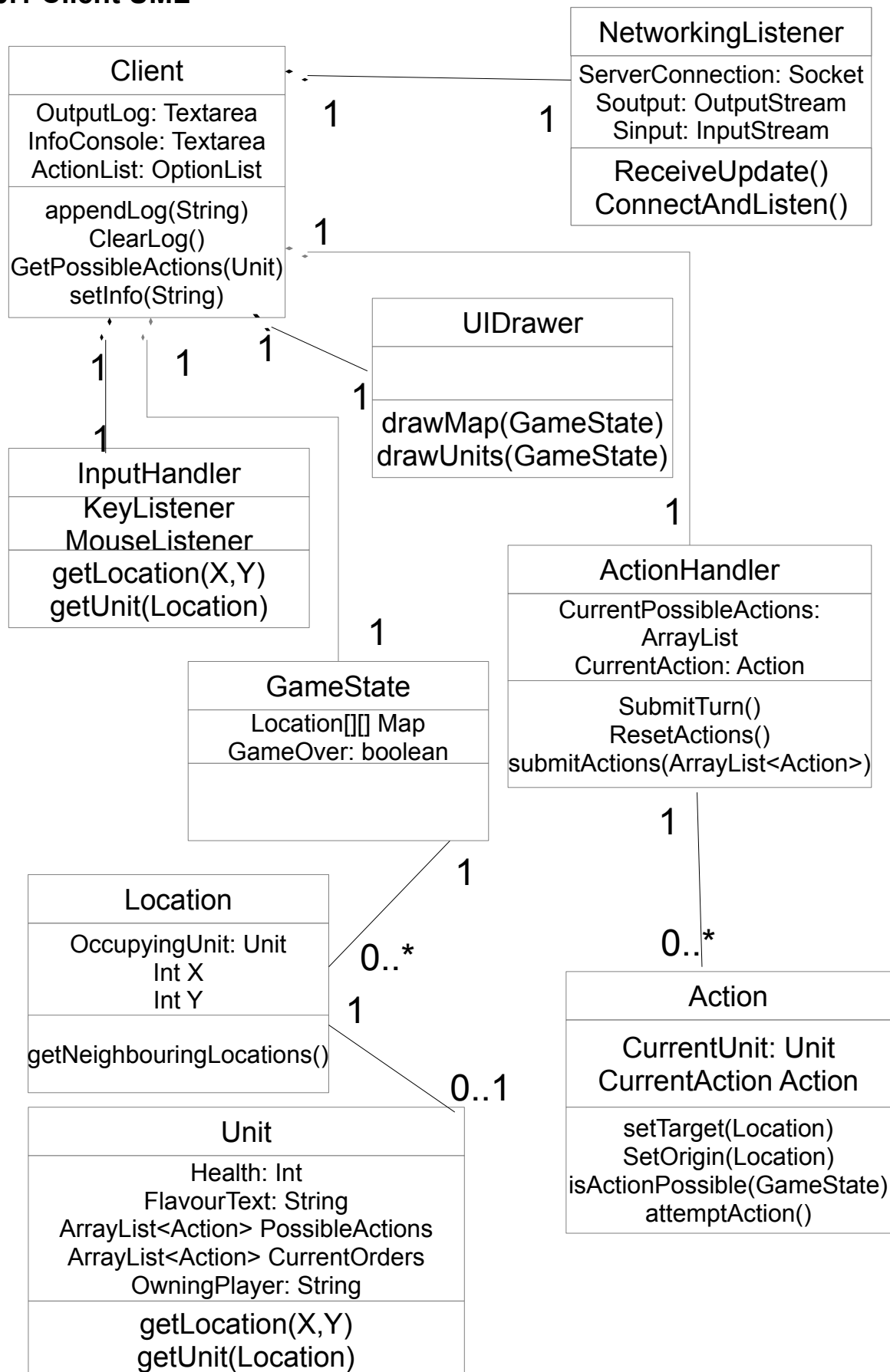
placement last year. The implementation of the project began with a large amount of steady work and expected programming solutions, but then introduced external libraries and required a large number of workarounds, alternative thinking and adjusting of preconceptions to overcome challenges. The final evaluation, testing and documentation stages of this project then provided a useful reflective tool to cement these experiences and store them for later use and application in my career.

## 9. References

- Garnett J. (2014) SelectionLab Bug. Available at: <https://jira.codehaus.org/browse/GEOT-4722> [Accessed 25th April 2015]
- Geotools. (2014). Geotools Tutorials. Available at: <http://docs.geotools.org/stable/userguide/tutorial/> [Accessed 26 April 2015]
- Geotools. (2015). Geotools Documentation. Available at: <http://docs.geotools.org/latest/javadocs/> [Accessed 30th April 2015]
- LocationTech. (2015). About Spatial4j. Available at: <https://spatial4j.github.io/spatial4j/> [Accessed 30th April 2015]
- Open Geospatial Consortium. (2015). About GeoAPI. Available at: <http://www.geoapi.org/> [Accessed 30th April 2015]
- Oracle. (2015). All About Sockets. Available at: <https://docs.oracle.com/javase/tutorial/networking/sockets/> [Accessed 30th April 2015]
- Prechelt L. (2000 March). An empirical comparison of C, C++, Perl, Python, Rexx and Tcl for a search/string-processing program. Available at: <http://page.mi.fu-berlin.de/prechelt/Biblio/jccpprtTR.pdf> Karlsruhe Institute of Technology [Accessed 25th April 2015]
- Software Testing Classes. (2012) What is Smoke Testing? Available at: <http://www.softwaretestingclass.com/smoke-testing/> [Accessed 30th April 2015]
- Sylvester T. (2013 January). *Designing Games: A Guide to Engineering Experiences*. O'Reilly
- Toal R. (2005). Java Socket Programming Examples. Available at: <http://cs.lmu.edu/~ray/notes/javanetexamples/> Loyola Marymount University [Accessed 30th April 2015]

# 10. Appendices

## 10.1 Client UML



## 10.2 Server UML

