**University of Essex School of Computer Science and Electronic Engineering**

**Tactical Level Game AI with Natural Language Input**

**Final Report**

**Piers Williams**

# Abstract / Summary

This is a project that is all about bringing human and computer Artificial Intelligence (AI) interaction close within a Real Time Strategy (RTS) computer game. I’m doing this after years of frustration with the single minded and un-cooperative nature of many RTS AI systems in games. Many tout having teams and alliances and trading however they are little more than token gestures that only work human-human. I wanted to create an Artificial Intelligence that could use those features through English input.

To achieve this I decided that Natural Language capabilities were the answer to solving this problem smoothly and efficiently. Communication with the AI is done through a text box that appears when the game is paused, and instructions can be entered. These instructions will be interpreted by the AI and then assessed and implemented if successful.

This report is split into 5 main sections. They are as follows:

1. The Project - This is a detailed description of what the project was designed to achieve, as well as the specifications of the program and some description of the history and meaning of the main ideas I was merging in this project.
2. The Program - This is a detailed description of how to use and what the resultant program can do. This includes a basic User Manual as well as some feature showcasing.
3. Implementation Details - This is a detailed description of the internals of the program that I developed as well as including external libraries that were used in the program.
4. Discussions and Conclusions - This is a section for discussing things about the project such as the problems and challenges I faced as well as a section for the kinds of improvements that could be added to the program to turn it from a technical demonstration of what can be achieved to a workable and playable computer game with the technology.
5. Project Planning - This is a section all about how I worked and planned the project, including some of the tools I used to help keep the project running smoothly. This also includes a small section on what I learned about project management in general.

Table of Contents

[Abstract / Summary 2](#_Toc354305221)

[Glossary of Terms 5](#_Toc354305222)

[Section I. The Project 5](#_Toc354305223)

[What is an RTS? 5](#_Toc354305224)

[What is Natural Language Processing? 6](#_Toc354305225)

[Aims and Objectives 6](#_Toc354305226)

[Functional Requirements 7](#_Toc354305227)

[Technical Specification 8](#_Toc354305228)

[Where does this fit in? 8](#_Toc354305229)

[Section II. The Program 9](#_Toc354305230)

[Instructions 9](#_Toc354305231)

[What this project can do 9](#_Toc354305232)

[Section III. Implementation Details 9](#_Toc354305233)

[Collision Detection 9](#_Toc354305234)

[Influence Calculations 9](#_Toc354305235)

[Aspect Orientated Entities 10](#_Toc354305236)

[Strategic Planning Language 11](#_Toc354305237)

[Natural Language Pipeline 11](#_Toc354305238)

[Part of Speech Tagger 11](#_Toc354305239)

[Tagged text to SPL 11](#_Toc354305240)

[Vector2D and Co-ordinates 11](#_Toc354305241)

[Vector2D Cache 12](#_Toc354305242)

[Mutable or Immutable? 13](#_Toc354305243)

[Extensible Mark-up Language (XML) 13](#_Toc354305244)

[External Libraries 13](#_Toc354305245)

[Light Weight Java Game Library (LWJGL) 13](#_Toc354305246)

[Themable Widget Library (TWL) 13](#_Toc354305247)

[Stanford Tagger 13](#_Toc354305248)

[Section IV. Discussions and Conclusions 14](#_Toc354305249)

[Problems Encountered 14](#_Toc354305250)

[OpenGL 14](#_Toc354305251)

[TWL 14](#_Toc354305252)

[Building an RTS on your own 14](#_Toc354305253)

[Future Work 15](#_Toc354305254)

[Voice Recognition 15](#_Toc354305255)

[Negotiation 15](#_Toc354305256)

[Trading 15](#_Toc354305257)

[Improved Graphics 15](#_Toc354305258)

[Wider Diversity 15](#_Toc354305259)

[Improved AI 15](#_Toc354305260)

[Section V. Project Planning 16](#_Toc354305261)

[Work Methodology 16](#_Toc354305262)

[Adapting To Change 16](#_Toc354305263)

[Methodology 16](#_Toc354305264)

[Risk Management 16](#_Toc354305265)

[Source Control 16](#_Toc354305266)

[Microsoft SkyDrive 16](#_Toc354305267)

[Why use two systems at once? 17](#_Toc354305268)

[What I Have Learnt 17](#_Toc354305269)

[Acknowledgements 17](#_Toc354305270)

[Perry Monschau 17](#_Toc354305271)

[Richard Astbury 18](#_Toc354305272)

[Bibliography 19](#_Toc354305273)

[Appendices 19](#_Toc354305274)

# Glossary of Terms

|  |  |
| --- | --- |
| Term | Definition |
| AI | Artificial Intelligence - The notion of getting computers to behave intelligently |
| POS | Part Of Speech - The role a particular word has in a sentence e.g. Jump is a verb |
| RTS | Real Time Strategy - A computer game that is played from a top down or angled view over the whole battlefield in real time |
| Vector | An object composed of an X and a Y value for Cartesian co-ordinate systems. |
| NL(I) | Natural Language (Input) is what humans use to communicate with. For the context of this report, it is taken to mean English. |
| AAA | A AAA game is one that is released to the public by a major publisher and is very popular / well known. Usually they are funded by large budgets in the millions of dollars |
| OpenGL | OpenGL is a cross-language and multi-platform API for rendering 2D and 3D graphics on a computer or other such device. |

# Section I. The Project

The project is all about bringing the ability to co-operate with an Artificial Intelligence and a human player in the context of a Real Time Strategy game. In order to do this I have written a simple RTS game with 4 players, one of which is controlled by the human, one of which is friendly to the human and two others that are friendly with each other but in competition with the human team.

### What is an RTS?

A Real Time Strategy game, or RTS for short, is a game where players control individual or groups of units in order to defeat the other players. They usually also have the option of constructing new units or buildings using accumulated resources. These mechanics often differ in style, but are fundamentally the same in most RTS games. RTS games also often offer the ability to be allied with one or more other players, played by either other humans or computer controlled AI systems. Originally RTS games were two dimensional, however an increase in available computing power allowed them to evolve into fully 3D worlds where you can zoom in and rotate around individual units smoothly. This type of game has never proved as popular as the more common First Person Shooters or Role Playing Games that dominate the market now, they have however been a favourite of mine with it being the first type of game I played on a computer and the only type I played for a while.

### What is Natural Language Processing?

Natural Language processing is the science of taking Natural Language, such as English or some other such language used by humans on this earth, and devising methods, algorithms and processes to cause computers to understand the Natural Language.

This idea began very early into the computing era, in the 1950's with the Georgetown experiment. This was a fully automatic translation of more than sixty Russian sentences to English sentences. Predictions about machine translation being solved within 3-5 years from that point were, however, optimistic and after a decade of research funding was largely cut. The next big success was the ELIZA project in 1964-1966 by Joseph Weizenbaum. This was one of the earliest chat bots, and was very simply made and largely un-intelligent but was surprisingly effective.

In the late 1980's, hand written rules and programs were the normal however a revolution in the field was the introduction of machine learning techniques. Machine learning and other statistical methods now dominate the field of Natural Language Processing, although their main drawback is the need for large corpora of data.

More recently, huge breakthroughs have been made with respect to Natural Language Processing through projects such as IBM Watson [[1]](#footnote-1). Watson is a computer program that managed to take part in and compete against two world champions in an American quiz show "Jeopardy!". "Jeopardy!" is an immensely hard game show, and Watson was not only able to compete but able to win against the best humans that had ever played "Jeopardy!".

### Aims and Objectives

For the project, I defined a set of goals that I would try to achieve. These are very broad and abstract statements however they were useful in guiding me towards what I was trying to accomplish. They are as follows:

#### To demonstrate that it is possible to merge NLP with RTS Game AI

* Create a RTS AI that can co-operate and collaborate with a human player smoothly
  + Assist player in attacks and defence
  + Trade with player to achieve goals or assist
  + Request help from player when necessary
* Create a RTS AI that can understand and process orders in NL
  + Execute a player's request accurately
  + Can attack given targets with a defined force
  + Can defend given targets with a defined force
* Create a 2-Dimensional (2D) game world that:
  + Has minimal complexity
  + Is large scale
  + Has Reasonable variety for game units

#### To demonstrate that Game AI doesn't have to be as poorly made as many AAA games

* Create a convincingly good game AI that can:
  + React well to new information
  + Be smart with its attacks and defences
  + Interact with the player as a single unified entity

### Functional Requirements

I have written a series of Functional Requirements for the project, in order to truly check if everything has been achieved. These are much less abstract that the goals I set above, and much more verbose as well:

1. The project should function as a basic RTS Game
   1. Look and feel
      1. The game will have basic 2D graphics
         1. The game will render at a fixed size different to the game's map size
         2. Units will be colour coded by owner
         3. Units will be square in shape
         4. There will be a blank black background
         5. The game view will be able to navigate the full game map
      2. The game will have a basic UI
         1. The UI can be configured from an XML file detailing the locations and purpose of buttons and labels
   2. Game Functionality
      1. The Game World is empty of everything but the units and buildings
      2. Units
         1. Units will be able to move to a destination without getting stuck 90% of the time or greater
         2. Units will be able to shoot targets at a regular pace to deal a regular amount of damage
         3. Units will have a fixed amount of starting health based on their unit type
         4. Unit health will not be repairable
      3. Factions
         1. Factions will exert their own independent Influence on the map
         2. A single faction will represent a single player in the game
         3. Factions can be allied together and will not attack each other within an alliance
      4. Artificial Intelligence
         1. The AI will be autonomous
            1. Each player will expand continuously if not interfered with
            2. Each player will attack other players until there are none left if not interfered with
            3. Each player will attempt to defend itself unless interfered with
         2. The AI will be omnipresent with full access to the game world
         3. The AI will use the Strategic Planning Language (SPL) internally
            1. SPL will be placed into one or more queues, for processing later, by either the AI's analytical half or the result of Natural Language Input
2. The project should demonstrate that it is possible to merge NLP with RTS Game AI
   1. The AI will use SPL for its internal communications with agents of the game
      1. AI will be split in half, one side analytical generating SPL, one half operational that consumes SPL
   2. The project will be able to translate Natural Language Input that is in English into SPL
      1. The project will achieve 80% or greater accuracy on understood instructions
      2. The project will be able to output a failure to understand instructions in a message to the user
      3. The project will perform conversions of a single sentence of instruction in under 10 seconds on modern hardware
      4. The project will inject NL derived SPL into the AI's SPL Queue for consumption once fully approved
3. The project will demonstrate that co-operation with game AI and human players is possible
   1. The AI must be able to evaluate and accept or reject human orders based on some internal criteria as to the suitability of an order
   2. The AI will have the ability to learn to trust the human based on a number of criteria:
      1. Success of human planned operations
      2. Value of traded items from human
      3. Assistance from human to AI
   3. The AI will be able to follow accepted human orders precisely

### Technical Specification

#### Platform

The project has been implemented in the Java programming language. I chose this language primarily because it is the one that I am most proficient in, but also due to a number of reasons:

* The Stanford Parser I planned to use was available nicely as a Jar file and sample java code
* Java is a high level Object Orientated language
* Nice and easy OpenGL support
* Some game based code from previous modules were done in Java by me

## Where does this fit in?

This project fits into two main communities, that of the Natural Language enthusiasts and that of the Game Developers.

Natural Language enthusiasts will be delighted to see some of their hard work forming the backbone for something that could really bring more and more people into the subject area, and getting its outreach out into the open with the average Joe using it regularly.

Games Developers will like this feature and ability in their games as it gives them a really strong Unique Selling Point against other or previous games if they manage to get it implemented correctly. With the much bigger budgets and developer power, a AAA games studio could really make the technology shine.

# Section II. The Program

This section details the end result of the project which is the program. It describes how to use the project as well as what it is capable of doing.

## Instructions

### Running the program

### Using the program

## What this project can do

### Attacking

### Defending

# Section III. Implementation Details

This section is split into sub sections that each independently detail a single module or idea that is implemented in the whole program.

## Collision Detection

The game is capable of handling a huge quantity of entities colliding at once smoothly through the use of a clever idea I was shown by fellow student Perry Monschau[[2]](#footnote-2). The basic premise is simple, to partition the world into a series of grids, you could check every entity within its grid against every entity in its grid, the next grid to the left, the next grid down and the next grid to the left and down. This would guarantee that it checked every entity that could be near the entity provided a simple rule is followed : the grid size is larger than any single entity. This stops an entity crossing over more than two grids, thus allowing the algorithm to massively reduce the amount of collision checks needed.

In tests I found that whilst the naive O(n2) complexity algorithm started to lag at about 1,500 units the more advanced algorithm didn't start to lag until about 20,000 units. This could perhaps have been fine tuned by altering the grid size parameter, however was quick enough for my needs. Even to this day with all the other code running, the Collision Detection uses very little CPU time and a near constant and small amount of memory. I used the Java JDK Profiler to obtain this information.

## Influence Calculations

The game features a core underpinning notion of unit influence. This is the radiating measure of a unit's power on the battlefield. Influence is made more efficient using a similar method to the collision detection algorithm detailed above.

The world is split into regular sized grids and entities find themselves put into a grid. Each entity has a double[][] that is their influence values with the x and y columns representing cells in the main board. Every time the influence map updates, it clears the board and fills it with the zero value. It then loops through every single entity, and adds that entities grid to the board. This operation is surprisingly quick, and runs without lagging although is one of the more intensive things the game has to process. This can be seen by running a profiler[[3]](#footnote-3), which shows it to use a good amount of a single core on my laptop.

The board is a 3D double array per faction, which corresponds to a double buffered 2D double array that represents the cells of the influence map. The reason for the double buffering was that the constant allocation of new Double arrays were causing the garbage collector to do lots of work clearing them out so instead I kept two arrays in memory throughout execution. This also solved some issues I had with drawing influence, as the previous method would flicker as it drew an array that got cleared half way through. Now I draw the slightly out of date buffer, allowing the influence thread to alter the alternative buffer without complications.

Because the board is split between factions, there are methods that allow factions to obtain the influence calculations for every enemy faction summed. this can be a slightly expensive calculation, and is not calculated regularly and not wasted once done.

The influence calculations are useful to the AI for working out suitably weak targets in the enemy camp, strong locations to avoid and even for improving their own bases by building influence exerting constructions near to weakly influenced buildings.

## Aspect Orientated Entities

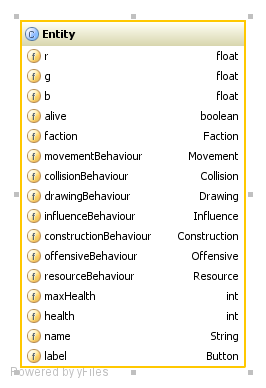
The game allows different units to be constructed through a technique that was inspired by a programming paradigm known as Aspect Orientated Programming [[4]](#footnote-4) and similar technique often known as Component Based Software Engineering.

The basic premise is that an Entity is defined as being a collection of components rather than actually consisting of code and data about itself. This is similar to how real world objects tend to be: with a car being made from a chassis and an engine and a body and exhaust and so on.

In my implementation, these components are concerned with one aspect of the parent entity with which to control. I gave each entity a single instance of each type of behaviour that I wanted. It would have been really simple to convert my implementation from having just one of each behaviour type to using lists of them, allowing layered drawing or multiple weapon types. This could also have been achieved by writing new behaviours that contain multiple behaviours themselves and delegate code based on its own internal state. In all I had these types of behaviour:

* Movement - concerned with the location of the entity and the movement of that location
* Collision - concerned with the details of the collision boundary for the entity
* Influence - concerned with providing the entity's influence onto the battlefield
* Drawing - concerned with drawing the entity when called upon
* Offensive - concerned with performing attacks on other entities
* Constructive - concerned with creating new units or buildings
* Resource - concerned with providing a resource stream from the entity

The end result of this architecture was that every Entity on screen was an instance of the same Entity class, but with different behaviours installed. I used the Factory Method[[5]](#footnote-5) to create entities, filling in the fields based on what had been asked for. This meant it was really easy to alter the behaviour of entities, or create new types of entities by re-using some previous behaviours and adding new ones. A UML class diagram for the entity class looks like this:



Here you can see the large part of the class is the Behaviours, with some other simple fields for simplicity.

## Strategic Planning Language

Early on in the project, I knew I would need to clearly define the types of behaviour that could be controlled and generated from the result of the natural language pipeline. This behaviour was written down in a reasonably formal specification that roughly resembled pseudo-code implementations of functions, with inputs allowing for some degree of re-use. An example of one function is:

Assassinate - Orders the AI to take out a single structure or unit without regard to others. Should withdraw after completion

2 Arguments (Object, Severity, After)

Object - The Target

Severity - How important it is to succeed

After - Optional list of Orders that must be completed first

This language was named the Strategic Planning Language (SPL), and it was also going to be used by the Artificial Intelligence internally. This meant that at all times, the AI would generate SPL and consume SPL, making the Natural Language part of the project a simple case of performing the conversions and then adding the resulting instructions to the same queue that the AI is using. Then it simply takes time for that order to become the next one being implemented by the AI. An additional result of the AI using SPL internally was that the Natural Language was capable of absolutely anything that the AI was and vice versa. This seemed like a good idea, as it is often frustrating when games implement clever features that are little more than token gestures due to them being limited to a small sub-set of the things that should have been done.

## Natural Language Pipeline

### Part of Speech Tagger

For the POS tagging section of the natural language pipeline, I decided that rather than implementing my own system I would make use of a well built and highly accurate tagger. For this I tested a few different taggers in my Interim Report. The first tagger I tried was QTag, which proved difficult to integrate with Java and had some odd ideas about what POS tag to give words. The second tagger I chose to test was the Stanford Tagger, which proved much easier and accurate. It had some little niggles but I've worked around those.

### Tagged text to SPL

The next section of the Natural Language Pipeline was to convert the nicely tagged text into our SPL orders. This was accomplished through the use of some simple code that involved being able to search for strings of tags in the list of words, and only getting the first instance of those tags from the text. Then simple switch statements were used to double check that some keywords were present before making the conversion. In all cases if something couldn't be found, it would simply return null and thus the pipeline would spit nothing out. This was quite nice as it meant the application didn't suffer any crashes. It would also be nice and easy to get it to spit out some form of error message to the end user.

## Vector2D and Co-ordinates

One major class of the project is a representation of a traditional mathematical Vector albeit one that has been limited to just two dimensions. I did this because the addition of more dimensions really wasn't necessary and would have also added massively to the amount of work I had to do. A class diagram for the Vector2D is presented below in Figure 1:

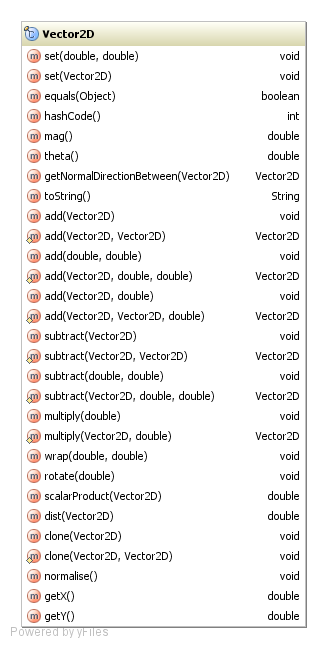


Figure : UML Class Diagram for Vector2D

With everything needing co-ordinates to represent accurately where the entity is, Vector2D's are rather fundamental. A whole raft of helper methods exist for calculating and returning or modifying these Vector2D's to help simplify the remainder of the code. This was largely based on the approach used in a previous module[[6]](#footnote-6) at University, which was inspired by Microsoft XNA's approach [[7]](#footnote-7). Having a single good representation helped give me code I could rely on, rather than using two floats each time and manually coding everything every time I needed it.

### Vector2D Cache

With so many Vector2D's being used in the program, I realised that many of them were actually being created and used as keys to dictionaries or references and were never being altered. They were however being destroyed every frame, meaning my program grew in its size rapidly before having to pause to let the Garbage Collector work its magic clearing them all out. In order to solve this problem, I created a cache that any class could access to get a Vector2D. This cache would only create a new instance if it hadn't issued one before. This meant that the CollisionBoard and InfluenceMap wouldn't cause such massive memory fluctuations because after a while most grids had been occupied and had a Vector2D in the cache already.

### Mutable or Immutable?

Near the end of the project, I decided that Vector2D's needed to be either mutable or immutable and the fact that they were mutable was rather un-nerving with so many of them being entered into the cache and shared it would be catastrophic if they were modified in any way. For this reason, I converted the class to support the creation of mutable Vector2D's and left all defaults to create immutable Vector2D's. I did this so that I had to explicitly ask for changeable Vector2D's. Any attempt to modify the data variables of an immutable Vector2D would cause an exception to be thrown, allowing me to hunt down lines of code that was violating the rule. I could also protect the cache, because the cache creates its own Vector2D's it could make sure they were always immutable. There is no provided way of changing from Immutable to mutable other than the cloning operation, which of-course protects the original.

## Extensible Mark-up Language (XML)

I have used the vastly common technology of XML[[8]](#footnote-8) for a number of configuration duties within my program. For a small period of time I gutted out a large portion of the hard-coded sections and re-wrote them to be read from XML documents instead. This change was largely inspired by the initial use of XML for arranging the User Interface, though it soon proved useful for various things such as the arrangement of buildings in the blueprints and the details of the buildings themselves.

## External Libraries

I used a number of external libraries of code written by other people in order to achieve my project. These libraries performed a wide variety of functions, and may have been mentioned elsewhere in the document however this is a more in-depth account of its use.

### Light Weight Java Game Library (LWJGL)

The LWJGL is a library that makes it easy to create games in Java using OpenGL graphics. Primarily it was the access to OpenGL that prompted me to use it. Knowing that my project would need lots of CPU power I wanted to try to avoid using Java2D for the graphics, with past experience showing it to be slow and clunky when given lots of things to do per frame. OpenGL appeared to have no trouble at all, and even though the graphics were rather simple, the point was to put as little strain on the CPU and in this respect the use of OpenGL worked wonderfully.

### Themable Widget Library (TWL)

TWL is a small library that gives me some basic User Interface elements such as labels, buttons and text boxes when using OpenGL. This was rather handy, as making my own would have been rather troublesome (OpenGL text is rather hard), and convenient. The library was rather hard to use, so I took the code that made one item work and did my best to only have to write that code once. I therefore used XML to configure the UI, with some simple code to place elements based on that XML.

### Stanford Tagger

The Stanford Log-Linear Part-Of-Speech Tagger is a high quality POS tagger developed by a team of scientists from the Computer Science Department at Stanford University. Using innovative techniques for the time (2003), the Stanford tagger achieved an initial accuracy on the Penn Treebank WSJ of 97.24%, an error reduction of 4.4% on the best previous single automatically learned tagging result. The tagger in its current state was really easy to use, as it contained with it some good java examples on its use which made it nice and simple to start implementing tagging within my project. The Stanford Tagger is also fast, with a small number of seconds to load the model initially, followed by near instantaneous tagging of single sentences. A final good reason for picking this tagger was that in my tests for the Interim Report, I found it to be the most accurate for the type of things I was going to be doing, requiring the easiest corrections to make it perfect.

# Section IV. Discussions and Conclusions

## Problems Encountered

### OpenGL

When I chose to use OpenGL I didn't think that it would be quite so hard to get some basic things working on screen. Especially after I got Quads (2D Squares) working reasonably quickly, I was confident. Later on though, I ran into trouble implementing features such as User Interfaces or more advanced graphics.

OpenGL lacks any support for nice UI's, and instead it required me to use an external library. This library was rather difficult to use correctly, and required a lot of effort. This was necessary though as even just the task of displaying text in OpenGL was difficult enough, and this hard library was at least simpler than that.

Advanced graphics are something that I have always struggled with, however in OpenGL it was looking like a lot of effort for little gain. The difference between a coloured Quad and a textured Quad in the tutorials was quite a bit longer and much more complex. Therefore I decided that the best thing to do was to simply not worry about adding in better graphics as the effort for gain ratio was too high.

### TWL

The Themable Widget Library was a headache to get working, and quite a bit of work to do much with. I therefore spent a while allowing my code to create the UI from XML specifications, so I didn't have to hunt about the Java code to try and achieve these things. There have however been problems, for example I can't work out how to change the appearance of things or get Labels to work correctly.

### Building an RTS on your own

I knew that building a completely new RTS game on my own with the limited time that I had would be difficult but I never thought it would be so difficult. I encountered problems nearly everywhere I went, with little information available on the internet for this type of game. This is likely due to the fact that it takes a lot more code and effort than other more popular types of game, which is why it gets left out of the tutorials on the internet.

I had huge trouble getting helpful ideas on how to implement the AI systems the most, as the only people that knew how were likely those that had done it for a company and couldn't release that sort of information.

## Future Work

### Voice Recognition

An interesting idea for future work to this project is the addition of Voice Recognition routines. This would greatly speed up order entry to the AI, as well as allow the player to do that whilst actually playing rather than having to pause the game to do that. Voice recognition can nicely improve the emersion and flow for the player, with the fewer interruptions greatly improving the experience. The best kind of experience is one that works so well you don't even notice that it is there.

### Negotiation

It would be great to expand the AI to the point that it can negotiate orders and even suggest changes or alterations that better suit it. This opens up the idea of having the AI start the entire process by sending the human an order to review for the human to then follow. Negotiation would make the application seem much more human and much more intelligent. This would also improve the experience for the user as they feel more immersed within the game.

### Trading

One feature that was supposed to be in the project was the ability to trade units through the Natural Language interface with allied factions. This would have massively helped the feeling of friendship and trust present between the human and the Artificial Intelligence. Trading would also allow a single unified command to wield forces larger than they could singly construct themselves, whilst allowing players to assist each other in new ways other than simply defending or attacking things for each other. This feature was cut out of the project due to time limitations and it not being one of the core concepts needed to make this project work and demonstrate its key aims.

### Improved Graphics

Whilst in this project graphics were not my focus, the project would benefit with a better graphical representation. With a team of programmers and designers, then the game would be much better looking with textures and more interesting shapes. The move to a 3D world could largely improve tactical options and realism, as well as making it easier to distinguish between different types of unit, something that is rather hard at the moment in my version. Improved graphics can also help to improve desirability of the game which would help to put the ideas contained within in the reach of more and more people.

### Wider Diversity

The game I have developed is rather narrow in the amount of options that players and AI alike have when going about their task of world domination. There is a limited choice of building and unit types, which is not typical of modern AAA RTS games. This is something that can be fixed easily with my architecture if I thought that the time spent would provide something of useful benefit. Unfortunately there are more important things I need to do to the game, and so this has been left out. This would however greatly improve the experience for the end user, and so would be a worthwhile future inclusion to the project.

### Improved AI

The AI for the project was built to get the job done and seem intelligent, however it lacks many features of a standard AAA RTS game. For example, there are no difficulty or style modes for the AI - they all perform the same. In addition, the end game performance of the AI in my game is rather lacking, as it takes a lot of knowledge and expertise (Something I don't have) to get right in the industry.

# Section V. Project Planning

## Work Methodology

For this project, I used an agile inspired work methodology. Agile development is all about pursuing simplicity and speed of development . I partitioned the large project into a series of tasks or sprints. These sprints would resemble a single feature or change to the system that had to be implemented. Whilst I initially intended to stick to this idea, it quickly fell apart and became an annoyance. This is due to the common flaw with trying to plan every little thing in advance, especially if you are in-experienced with the particular type of project. The order and even the sprints themselves changed frequently, until I realised it would be better to decide the next step only really after the previous step has been completed. For a large part of the project this was my method, getting something to work and then thinking about what was next. This proved much quicker than spending days designing a plan that was quickly ignored because I found I needed to do something else next.

## Adapting To Change

### Methodology

Initially I adapted to change by keeping an eye on how my project was going compared to the set of sprints I had in my Gantt chart. This resulted in me re-making the Gantt chart a few times, which took time and was often just as wrong as the previous one. The first thing I had to adapt to was my methodology itself.

## Risk Management

### Source Control

One method I used to control and manage risk was to use Source Control. I used the Git system in conjunction with Git-Hub hosting. This gave me revision control as well as a clear and simple to browse log of my activity. I could also be protected from losing my project as a result of hardware failure. Revision control was a nice safety net for developing new features near to deadlines, as in all cases I could roll it back to the last good working version for the deadline and then continue to develop afterwards. The system of putting messages in with every commit also largely gave me a good history of what I had done and when. This was very helpful in keeping track of progress when I got back to the system after a day or so of not working on it, as I could see where I left off and what I had done.

### Microsoft SkyDrive

I already had been using Microsoft's online cloud storage program SkyDrive for my entire set of documents which includes my University work. This protects my data from being lost in hardware failure events by syncing the data in real time to the cloud[[9]](#footnote-9). I have long had a sub-conscious reaction to saving documents when I stop typing something. Whilst this can be annoying on the web, it works a treat when developing or writing reports. I could also really quickly set up replacement computers or work from multiple pc's at different times as changes were synced automatically when I switched new ones on. In all cases, the data was available through the online interface, if for example I needed to use a University Lab Machine to do some work. This technology was actually tested when my laptop decided to stop booting, forcing me to wipe the C drive including my entire collection of work. Upon re-installing I was able to simply re-download everything and get back to working within hours.

### Why use two systems at once?

|  |  |  |
| --- | --- | --- |
| Software | Versioning support | Backup |
| GitHub | Advanced, easy | Manual |
| SkyDrive | Basic, easy | Automatic |

Bringing together both the hardware protection from SkyDrive with the work logging systems of Git meant I had the best of both worlds. Relying on the fact that GitHub's protection was only as good as my ability to remember to push updates to the internet was a little risky. In addition, git provided much better versioning especially for code. This allows me to roll back the system as well as roll forward the system. SkyDrive does have file versioning, but isn't easy for rolling back more than one file at once or rolling them forward at all.

## What I Have Learnt

I have learnt that projects grow and change as you work on them, and never to rely on anything staying constant through the development time. I have also learned that being ambitious can be good, but also very challenging. I set this project to be something I knew would really push the limits of my capability but be a nice tough challenge that could turn out to be hopefully impressive. I have also learned that really thinking about the infrastructure you set up to work in is really important. The absolutely robust system I set up may have seemed over the top, even prompting confusion from some other students. I did however suffer some failures with my laptop meaning they came in handy. I've also seen other students suffer a lot when their drives crash, destroying their only copy of some work.

# Acknowledgements

## Perry Monschau

Perry Monschau is a third year student at the University of Essex who assisted me with implementing the Collision Detection system that is in use in my project. He largely talked me through the theory as I implemented it in code myself until I had a working application. I have since modified the code a lot from that original implementation including turning it into a Runnable object in order to get it working on a separate thread from the Main.

I believe that this should not be an issue of plagiarism as the collision detection was not part of the project goals and was a necessity that I had to have in order to work on those goals. In addition, the modifications I made were extensive and differs greatly from the code I originally implemented with his help.

## Richard Astbury

Richard Astbury is a Senior Consultant at two10degrees[[10]](#footnote-10), an Ipswich based Cloud Computing Company. During my internship there between my second and third year, he introduced me to the idea of using Git and GitHub for managing projects. I also learnt a lot about how JavaScript allows you to inject dependencies into things, which is similar to the Aspect Orientated framework I used.

# Bibliography

|  |  |
| --- | --- |
| [1] | D. K. C. D. M. Y. S. Kristina Toutanova, “Feature-Rich Part-Of-Speech Tagging with a Cyclic Dependancy Network,” 2003. |
| [2] | K. Beck, M. Beedle, A. v. Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, R. Jeffries, J. Kern, R. C. Martin, S. Mellor, K. Scwaber, J. Sutherland and D. Thomas, “Manifesto for Agile Software Development,” Agile Alliance, 2001. |

# Appendices

1. http://www-03.ibm.com/innovation/us/watson/index.shtml [↑](#footnote-ref-1)
2. For more information see the Acknowledgements section [↑](#footnote-ref-2)
3. Standard JDK Profiler - Java VisualVM [↑](#footnote-ref-3)
4. http://en.wikipedia.org/wiki/Aspect-oriented\_programming [↑](#footnote-ref-4)
5. http://en.wikipedia.org/wiki/Factory\_method\_pattern [↑](#footnote-ref-5)
6. CE218 <http://www.essex.ac.uk/modules/default.aspx?coursecode=CE218&level=5&period=SP> [↑](#footnote-ref-6)
7. XNA: <http://msdn.microsoft.com/en-us/centrum-xna.aspx> [↑](#footnote-ref-7)
8. http://www.w3.org/TR/REC-xml/ [↑](#footnote-ref-8)
9. For text edits to code, changes are synced to the cloud in about 60-90 seconds from them being saved to disk [↑](#footnote-ref-9)
10. two10degrees: http://www.two10degrees.com [↑](#footnote-ref-10)