University of Essex School of Computer Science and Electronic Engineering

Tactical Level Game AI with natural Language Input

Interim Report

Piers Williams

11/30/2012

Table of Contents

Goals of the project

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
  - Understand and generate an in-game representation of a player's request
  - Assess the players request and calculate costs and chance of success
  - Accept or reject or amend the order based on assessments and the trust level between the AI and the player
- 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

These goals form an abstract form of the full System Requirements for the project. They are not however expressed quite as formally as many requirements documents, though I think they clearly communicate the goals and markers that indicate when the project has succeeded or failed. It is this that is more useful to me than page after page of requirements that mention every little functional detail imaginable about the proposed system.

Gantt Chart

My Gantt chart has been through a serious re-ordering since my last report. This was due to the items being planned out in advance and the reality of how things were going to be implemented being radically different. Once I started programming, it was obvious the correct order that I needed to implement the different features. As such, most of the tasks are still the same, but have moved in time. The Gantt chart has however been expanded to include some extra detail, and there is use of a new feature that makes ghost tasks that rely purely on other sub-tasks that make it up. This feature makes the chart clearer, as I tried to implement something previously using pre-conditions however it wasn't a great solution. The full Gantt chart is displayed in the Appendix and included with the online submission in the Microsoft Project format.

Background Reading

Beyond Façade pattern matching [4]

This is a rather useful article that focuses on the Façade NLP system. This system was largely built to improve the state of things after AIML. AIML was XML based and wordy, plus lacked refinement or extensive operators, requiring many rules just to match simple statements. Façade was created around Jess, a java based extension of the CLIPS expert-system language. Façade was a great improvement however its Lisp like syntax made it really ugly to read and write. As such, ChatScript was written to improve these short comings. The syntax was much nicer and concise as well as adding some useful features. one feature caught my eye, and may prove useful for my project

Concepts

ChatScript has the idea of a "Concept" which can be synonyms or a natural ordering of equivalent items. The example given in the article is:

concept: ~meat (bacon ham beef meat flesh veal lamb chicken pork steak cow pig)

Which in my game could be:

concept: ~unit(SimpleUnit, AdvancedUnit, Base, Building)

and so on.

This would allow the NL to pickup references to in game objects easily, with the concepts constructed at game compilation where they are fully known.

Progress To Date

Design approach and methodology

The project is being designed in parts gradually, though a more abstract design exists to guide the process. Each iteration and the tasks that are required to achieve the iteration have been decided in advance and forms a simple guide as to what I need to achieve to count that iteration as complete. The iteration is built from a copy of the previous iteration after it was completed. The first phase is usually preparatory refactoring. This is about preparing the code for the change ahead, and catching any ugliness that the last stage of the previous iteration missed. Secondly, the iteration is completed by altering and adding to the code to achieve the tasks and goals of the iteration. At all times, things are done as well and cleanly as possible, with many mini refactors occurring if and when issues arise that require them, or I spot a neater way to do something. Finally, a full refactor is performed to try to clean up the code afterwards, and perhaps deal with massive obvious performance enhancements.

Software design however is done as and when necessary, as trying to design the entire thing in one go would cause a major headache and more than likely be a hindrance as I deal with errors in the design rather than simply having the free model where I re-design each and every iteration, carefully growing and evolving the structure as needed. That said, a careful eye on future tasks helps me avoid lengthy refactors caused by poor design when future requirements change, as I do know the future requirements to an extent. For example, I know that the units will need to be controlled by an AI, so it makes sense to ensure I don't lock them out of that sort of control.

Objectives achieved

I have so far achieved a number of the items in my Gantt chart, although they have been largely re-ordered since my Initial Report. I have successfully got:

- A concept of teams
- Buildings that belong to a team and produce units for the team regularly
- Groups of units that belong to a team, can have new units added to them and wander around randomly
- Influence calculations for each team including crude drawing of this

This means that I have a game world with some buildings in it that each belong to a different faction. These building continuously build units at a specified rate, with the units being added to a group. The group moves about the world aimlessly. The units that belong to the group use a flocking inspired algorithm to head towards the target of the group, whilst also maintaining separation from the other units. There is no rotation, so the alignment section of flocking is omitted.

Natural Language Research

I have been dabbling with Natural Language Processing for my project as and when I learn about things in the module (CE314) at University. One of the things I did since the Initial Report was to try Part-Of-Speech (POS) tagging on some example NL inputs to the system.

QTag [3]

My first attempt was to use QTag to tag the text, in combination with the PreTokeniser class that was also provided in a CE314 laboratory. Here is a sample from the run that wasn't entirely successful

Construct_NN, a_DT, small_JJ, army_NN, and_CC, when_CS, my_PP$, army_NN, attacks_NNS, the_DT, left_JJ, flank_NN, of_IN, GV-34,_DT, start_VB, your_PP$, attack_NN, on_IN the_DT, right_JJ

Obvious inaccuracies are the fact that "Construct" has been tagged as a Noun, however by the context it is clear that it ought to be a Verb of some form. GV-34 ought to be a Noun however it has been tagged as a Determiner. These could however be altered afterwards to correct them. Nouns at the start of the sentence are more likely to be a Verb in the more declarative style that orders in the system are.

Stanford Tagger [1]

Construct_VB a_DT small_JJ army_NN and_CC when_WRB my_PRP$ army_NN attacks_VBZ the_DT left_JJ flank_NN of_IN GV-34_NN ,_, start_VB your_PRP$ attack_NN on_IN the_DT right_NN

The Stanford tagger has done a much more accurate job, including correctly tagging the un-known entity as a Noun and the word "Construct" as a verb. Unfortunately, the word "right" ought to be an Adjective as it was in the QTagger, however this is fixed if you tag again after appending the word "flank" after it to provide context. However my original example is a good example of what a real gamer or person would write, and is understandable to a human.

Table 5 of [1] shows that one of the Stanford Tagger's main top confusion pairs is confusing Nouns with Adjectives.

Additionally, the Stanford tagger is quite a bit easier to integrate into a Java application which would prove quite useful for my project.

Possible future research

Potential other taggers are the Brill-Tagger or Machinese Syntax to see if any of them improve on my explored options. I should also look into sentence parsers

Plans for next term

Next term I hope to complete the more advanced sections of AI for the game and then move on to working on the Natural Language Processing part of the project and get the AI to properly understand orders and the implement them.

General Design

Game Design

The general design of the game has not changed at all, and is still the same as described in the Initial report. A summary is included here as a reminder.

The game is a 2D Real Time Strategy game, and the game is played using square based units. These units will come with a variety of abilities and in a variety of sizes. The game is set in space, a large expanse with sparse objects naturally. Usual space based technology such as lasers, missiles and shielding exist as well as the possible addition of stealth technology.

Software Design

Thread system

The project is currently structured as a series of threads that handle and process one thing at a time and co-operate together to produce the final result. This approach should improve performance massively on multithreaded systems which are becoming prevalent now. It also reduces the amount of code in any single update loop and is not much harder than the single threaded versions of things.

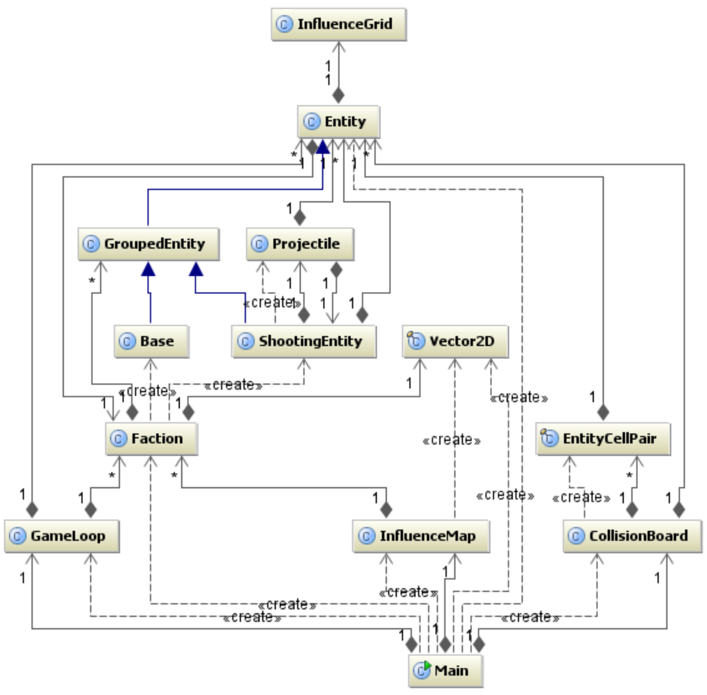
There are a number of threads so far in the system:

- CollisionBoard
    - This thread is there to continuously check for and calculate collisions between objects. It uses very simple calculations to place units inside equal sized cells and therefore only has to check a cell against the cell to its left, the cell underneath and the cell to its left and underneath. Some further optimisations can be applied, however the system is currently capable of quickly detecting collisions between thousands of objects on screen whereas the naive approach was struggling at about 2,000 objects.
- GameLoop
    - This thread is tasked with holding all the game objects and calling their update methods periodically. The objects are available for synchronised access by the other threads and this is the master copy for the game.
- Main
    - This thread is the master thread and after setting up the daemon threads it runs the drawing code in its loop.
- InfluenceMap
    - This thread is tasked with calculating and providing access to the influence maps that the AI will use to make global tactical decisions.

○

Class Organisation

The classes are generally organised in a logical fashion, with appropriate inheritance and aggregation used to achieve the goal as simply as possible. The following diagram is an abstract view of the current prototype for the project, and does not include the full details



As you can see, A huge amount of the work is done with the Entity and Faction classes. This is because most unit operations are only concerned with the functionality stored in Entity or Faction, even if most of the code lies in overriding functions elsewhere. This should allow new functionality to be easily added, making the system extendable. This diagram is the state of the system as it is, not how it will be.

Artificial Intelligence Architecture

The architecture has not changed since the Initial Report though some features have been expanded.

Strategic Planning Language

The Strategic Planning Language(SPL) has been expanded to allow Orders to list other Orders as pre-requisites to the start of an order. The full state of the SPL is included in the Appendix

Plans

A plan is essentially a series of low level orders that can be given to the AI. It is constructed from SPL instructions after the NL stage and has a number of features

Orders

Orders are instructions to either attack or defend in some degree and correlate to the SPL that was defined in the Initial Report. Orders are further decomposed into routes and targets and handed to groups of units.

Triggers

Triggers allow timed events to occur. The plan may say that the left flank should be attack at the same time as the right flank, and so the left player needs a trigger that says the right player has begun so he can carry out the left attack. Additional uses are for actions that must take place in the correct order. Generally every Order will contain a trigger that fires when it is successful, and Orders can list these triggers as prerequisites.

Negotiations

The AI will hopefully be able to negotiate with the player about the orders you have given it, either accepting, rejecting or altering them. Whether the AI will accept the order is based on a number of factors:

- Cost of the order
- Gain to the AI of the order
- Trust of the player by the AI
- Confliction with goals of the AI (possibly)

An example negotiation is as follows:

<Player> **::** I would like you to attack the Blue's front base

<AI> **::** that's too heavily defended, but I'll do it if you take out the flanking base first

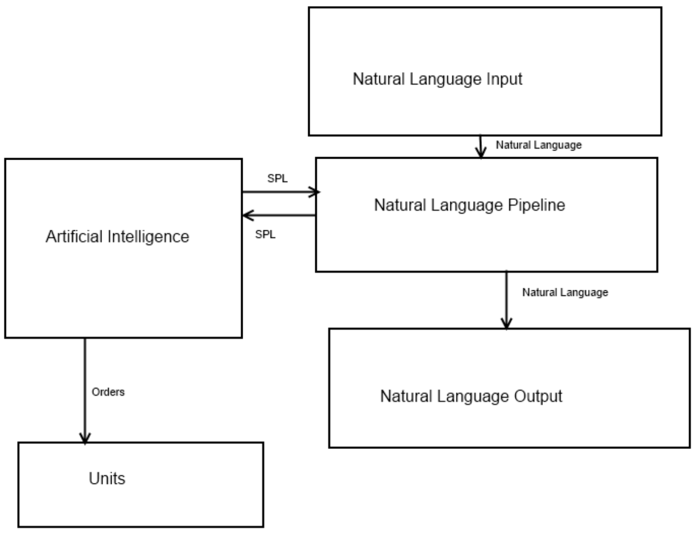
The new plan would hopefully be displayed on screen as well, demonstrating clearly the intentions of the AI. This would show the planned routes, locations and timings that the AI has come up with. It is then up to the player to accept or reject or amend the plan. Upon accepting, each player will be given ghost elements on their screen representing the plan's trigger points etc.

User Interface

The user interface will be split into two halves. The first half is going to be the main game interface, which will be a very cut-down version of what most RTS games feature. This is because rather than waste a lot of time trying to write a smooth OpenGL interface on top of my game, only simple information display and keyboard/mouse controls will be available. A second computer will use Socket programming to communicate with the game and provide a User Interface written in Java Swing. This will use standard buttons and layouts to keep a nice easy to use interface that will allow the user to compose their Natural Language before sending it to the AI. It is expected that the two machines are operated next to or near each other, or perhaps even by two persons.

NLP Architecture

The general modules and their relations are shown here. Natural Language is fed into the pipeline, converted to SPL, fed to the AI and converted into orders that are distributed to the units. In addition, SPL may be fed back to the pipeline to be converted into Natural Language in order to be displayed to the human. This feature however is dependent on time available, as it is reasonably hard and less essential than the other steps. It would greatly improve the realism and "feel" of talking to the AI, especially in negotiations.



Natural Language Pipeline

The Natural Language Pipeline section of the program deserves a section of its own to describe its architecture. The steps for this will be:

1. The input text will be tokenised into individual tokens
2. The tokenised text will be POS tagged
3. The tagged text will be parsed into a sentence tree
4. Annotate Noun phrases with in-game object references
5. Annotate Verb phrases with base/understandable actions
6. These phrases will be analysed to convert them into Functions(Verb phrases) and arguments(Noun phrases, references to in-game objects). This will be a rule based system
7. Finally the SPL functions will be returned to the AI system to execute

Appendix

Strategic Planning Language

```
**************************************************
*** Strategic Planning Language Specification ****
**************************************************

Two Parts to the language.
Query and Order

*************
*** Query ***
*************

A Query is a request of information from the AI

Query
        QueryObject
                HowMany - How many of these do you have?
                        1 Argument(ObjectType)
                                ObjectType - Object type to count
                HowManyGreedy - How many of these and related do you have?
                        1 Argument(ObjectType)
                                ObjectType - Object type to count, as well as sub and parent types
                HowManyAvailable - How many of these to spare?
                        1 Argument(ObjectType)
                                ObjectType - Object type to count
                HowManyGreedyAvailable - How many of these and related to spare?
                        1 Argument(ObjectType)
                                ObjectType - Object type to count, as well as sub and parent types
        QueryTime
                HowLong - How long to complete an order?
                        1 Argument(Order)
                                Order - A previously agreed Order
                HowLongStart - How long till start an order?
                        1 Argument(Order)
                                Order - Either previously agreed Order or OrderWIP

        QueryAssessment
                AttackStrength - Evaluate attack strengths / weaknesses
                        1 Argument(Player/Team)
                                Player/Team - Evaluate the player or team given with all available information to AI's team
                DefenseStrength - Evaluate defense strengths / weaknesses
                        1 Argument(Player/Team)
                                Player/Team - Evaluate the player or team given with all available information to AI's team
                TotalStrength
                        AttackStrength(Player/Team) && DefenseStrength(Player/Team)
                AssessAll - Assess TotalStrength of all players and return matching strengths to weaknesses



*************
*** Order ***
*************

An Order is a request to the AI to carry out a task

Order ::= Order | Order && Order

4 Main Types of order

Attack, Defend, Trade and Explore

Attack - Attack orders are given to direct the AI to attack a location or structure or base.

        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

        Destroy - Orders the AI to completely destroy a base
                2 Arguments (Base, Severity, After)
                        Base - The Base to destroy
                        Severity - How important it is to succeed
                        After - Optional list of Orders that must be completed first

        Neutralize - Orders the AI to keep the enemy out of a region
                3 Arguments (Location, Radius, Severity, After)
                        Location - Location to use as centre point
                        Radius - How wide around the centre point to go
                        Severity - How important it is to succeed
```

- After - Optional list of Orders that must be completed first

Defend - Defend orders the AI to perform defensive tasks

- Grow - Orders the AI to improve its own base defences
  - 3 Arguments (Base, Type, Severity, After)
    - Base - The Base to grow
    - Type - The thing to defend against
    - Severity - How much resources to put into this. Out of 5.
    - After - Optional list of Orders that must be completed first
- Occupy - Orders the AI to construct a base in an area
  - 3 Arguments (Location, Type, Severity, After)
    - Location - The location to use as centre point
    - Type - The purpose of the base
    - Severity - How large/Strong will the base be
    - After - Optional list of Orders that must be completed first
- OccupyForMe - Orders the AI to construct a base for me in an area
  - (Occupy followed by GiveBase)
  - 3 Arguments (Location, Type, Severity, After)
    - Location - The location to use as centre point
    - Type - The purpose of the base
    - Severity - How large/Strong will the base be
    - After - Optional list of Orders that must be completed first

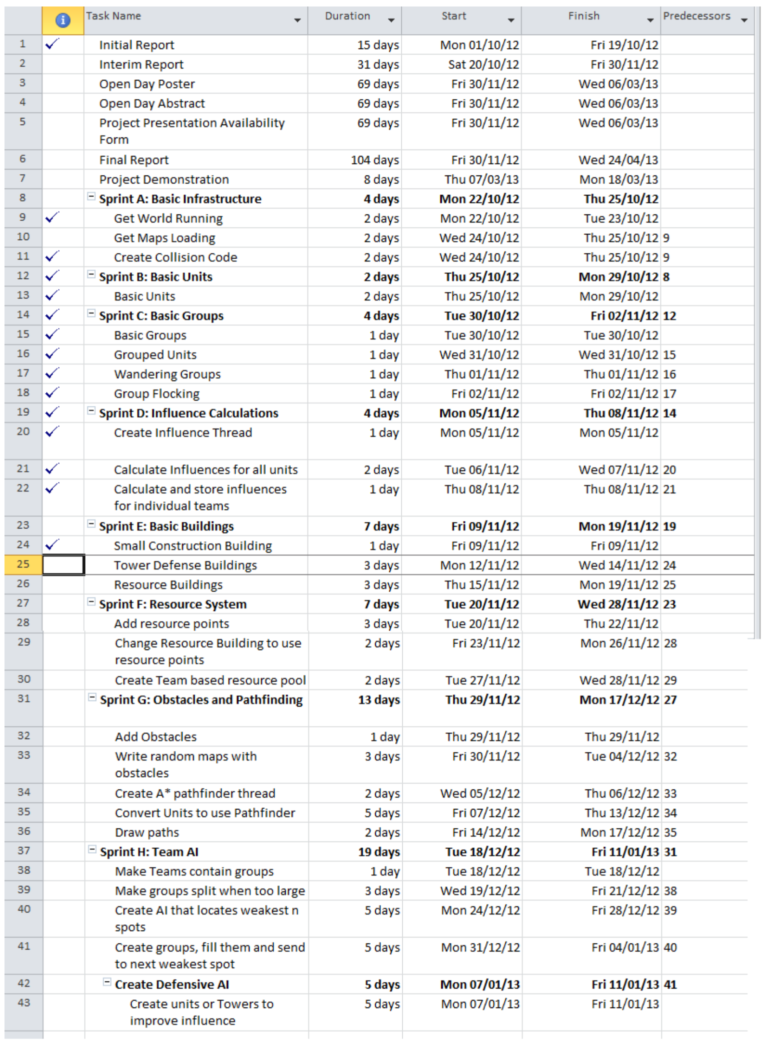
Explore - Explores the terrain

- Explore - Explore a geographical location
  - 3 Arguments (Location, Radius, Severity, After)
    - Location - The Location to use as centre point
    - Radius - How much to explore around the point
    - Severity - How soon should we explore
    - After - Optional list of Orders that must be completed first
- Search - Search for a particular item
  - 4 Arguments (Resource, Location, Radius, Severity, After)
    - Resource - The Resource we are looking for
    - Location - The Location to use as centre point
    - Radius - How much to explore around the point
    - Severity - How soon should we explore
    - After - Optional list of Orders that must be completed first

Trade - Negotiates trade of units/resource/buildings

- Give - Orders the AI to hand over control
  - GiveObject
    - 1 Argument (Object)
      - Object - The Object to be transferred
  - GiveGroup
    - 1 Argument (Group)
      - Group - The Group to be transferred
  - GiveBase
    - 1 Argument (Base)
      - Base - The Base to be transferred
  - GiveAll
    - 0 Arguments
- Receive
  - ReceiveObject
    - 1 Argument (Object)
      - Object - The Object to be transferred
  - ReceiveGroup
    - 1 Argument (Group)
      - Group - The Group to be transferred
  - ReceiveBase
    - 1 Argument (Base)
      - Base - The Base to be transferred
  - ReceiveAll
    - 0 Arguments
- Swap
  - SwapObject
    - 2 Arguments (ObjectMine, ObjectYours)
      - GiveObject(ObjectMine) && ReceiveObject(ObjectYours);
  - SwapGroup
    - 2 Arguments (GroupMine, GroupYours)
      - GiveGroup(GroupMine) && ReceiveGroup(GroupYours);
  - SwapBase
    - 2 Arguments (BaseMine, BaseYours)
      - GiveBase(BaseMine) && ReceiveBase(BaseYours);

Gantt Chart



	ℹ	Task Name	Duration	Start	Finish	Predecessors
1	✓	Initial Report	15 days	Mon 01/10/12	Fri 19/10/12	
2		Interim Report	31 days	Sat 20/10/12	Fri 30/11/12	
3		Open Day Poster	69 days	Fri 30/11/12	Wed 06/03/13	
4		Open Day Abstract	69 days	Fri 30/11/12	Wed 06/03/13	
5		Project Presentation Availability Form	69 days	Fri 30/11/12	Wed 06/03/13	
6		Final Report	104 days	Fri 30/11/12	Wed 24/04/13	
7		Project Demonstration	8 days	Thu 07/03/13	Mon 18/03/13	
8		**Sprint A: Basic Infrastructure**	**4 days**	**Mon 22/10/12**	**Thu 25/10/12**	
9	✓	Get World Running	2 days	Mon 22/10/12	Tue 23/10/12	
10		Get Maps Loading	2 days	Wed 24/10/12	Thu 25/10/12	9
11	✓	Create Collision Code	2 days	Wed 24/10/12	Thu 25/10/12	9
12	✓	**Sprint B: Basic Units**	**2 days**	**Thu 25/10/12**	**Mon 29/10/12**	**8**
13	✓	Basic Units	2 days	Thu 25/10/12	Mon 29/10/12	
14	✓	**Sprint C: Basic Groups**	**4 days**	**Tue 30/10/12**	**Fri 02/11/12**	**12**
15	✓	Basic Groups	1 day	Tue 30/10/12	Tue 30/10/12	
16	✓	Grouped Units	1 day	Wed 31/10/12	Wed 31/10/12	15
17	✓	Wandering Groups	1 day	Thu 01/11/12	Thu 01/11/12	16
18	✓	Group Flocking	1 day	Fri 02/11/12	Fri 02/11/12	17
19	✓	**Sprint D: Influence Calculations**	**4 days**	**Mon 05/11/12**	**Thu 08/11/12**	**14**
20	✓	Create Influence Thread	1 day	Mon 05/11/12	Mon 05/11/12	
21	✓	Calculate Influences for all units	2 days	Tue 06/11/12	Wed 07/11/12	20
22	✓	Calculate and store influences for individual teams	1 day	Thu 08/11/12	Thu 08/11/12	21
23		**Sprint E: Basic Buildings**	**7 days**	**Fri 09/11/12**	**Mon 19/11/12**	**19**
24	✓	Small Construction Building	1 day	Fri 09/11/12	Fri 09/11/12	
25		Tower Defense Buildings	3 days	Mon 12/11/12	Wed 14/11/12	24
26		Resource Buildings	3 days	Thu 15/11/12	Mon 19/11/12	25
27		**Sprint F: Resource System**	**7 days**	**Tue 20/11/12**	**Wed 28/11/12**	**23**
28		Add resource points	3 days	Tue 20/11/12	Thu 22/11/12	
29		Change Resource Building to use resource points	2 days	Fri 23/11/12	Mon 26/11/12	28
30		Create Team based resource pool	2 days	Tue 27/11/12	Wed 28/11/12	29
31		**Sprint G: Obstacles and Pathfinding**	**13 days**	**Thu 29/11/12**	**Mon 17/12/12**	**27**
32		Add Obstacles	1 day	Thu 29/11/12	Thu 29/11/12	
33		Write random maps with obstacles	3 days	Fri 30/11/12	Tue 04/12/12	32
34		Create A* pathfinder thread	2 days	Wed 05/12/12	Thu 06/12/12	33
35		Convert Units to use Pathfinder	5 days	Fri 07/12/12	Thu 13/12/12	34
36		Draw paths	2 days	Fri 14/12/12	Mon 17/12/12	35
37		**Sprint H: Team AI**	**19 days**	**Tue 18/12/12**	**Fri 11/01/13**	**31**
38		Make Teams contain groups	1 day	Tue 18/12/12	Tue 18/12/12	
39		Make groups split when too large	3 days	Wed 19/12/12	Fri 21/12/12	38
40		Create AI that locates weakest n spots	5 days	Mon 24/12/12	Fri 28/12/12	39
41		Create groups, fill them and send to next weakest spot	5 days	Mon 31/12/12	Fri 04/01/13	40
42		**Create Defensive AI**	**5 days**	**Mon 07/01/13**	**Fri 11/01/13**	**41**
43		Create units or Towers to improve influence	5 days	Mon 07/01/13	Fri 11/01/13	

Abbreviations

RTS - Real Time Strategy

2D - Two Dimension/Dimensional

SPL - Strategic Planning Language

AI - Artificial Intelligence

NL - Natural Language

NLI - Natural Language Input

NLP - Natural Language Processing/Pipeline

POS - Part Of Speech

AIML - Artificial Intelligence Mark-up Language

References

[1] Kristina Toutanova and Christopher D. Manning, "*Enriching the knowledge Sources used in a Maximum Entropy Part-of-Speech Tagger"* located at "http://nlp.stanford.edu/pubs/emnlp2000.pdf"

[2] Dr Udo Kruschwitz, *"CE314 Natural Language Engineering"* located at " http://courses.essex.ac.uk/ce/ce314/"

[3] Oliver Mason *"Qtag"* located at " http://phrasys.net/uob/om/software"

[4] Bruce Wilcox, *"Beyond Façade: Pattern Matching for Natural Language Applications"* located at " http://www.gamasutra.com/view/feature/134675/beyond_fa%C3%A7ade_pattern_matching_.php?print=1"