To What Degree can an AI Built With Expert Strategies be Effective Against Competition AI?

James Hellman
Falmouth Games Academy
UK, Falmouth
Email: jh182233@falmouth.ac.uk

***Abstract*—Effective macro-management (the ability to create armies and expand bases), is essential to obtaining victory in Real-Time Strategy (RTS), in the research community many Artificial Intelligence's (AI's) have been created to handle this. One method is to use a design approach to create what is known as a build order, many of these build orders take from expert strategies used by real people in high ranking tournaments. Build orders can be ridged during games leaving little room for adaptation to the opponent's strategy. In this work, a collection of build orders will be used to create an AI and investigates the impact of build orders that effectively counter-strategies used by other AI's. A hypothesis is made here that the AI will only be effective in the early stages of the game and will be outmanoeuvred in late-game stages. Therefore the effectiveness of this AI will be measured its average time survived, with a high average being effective and a low average being ineffective. Whether the AI wins the matches will also be taken into account, a higher average win rate will allow the AI to face a greater challenge. Upon successful completion of this work, the AI will be submitted to one of three competitions.**

I. Introduction

INVESTIGATING the effectiveness of an AI can be done in many ways, in this work will be looking into the degree of effectiveness an AI built with expert strategies can be against a competitive AI. In games, AI has been used in both single and multi-player environments to help create a more immersive, challenging and fun experience. One such area which AI is prominent is in the Real-Time Strategy (RTS) genre and since the call for more research to be made for AI in RTS games by Michael Buro in 2004 [1], research in this area has exploded [2]. This has given rise to the creation of many AI's in RTS games, from AI's that are built with pre-defined build orders [3] to deep Neural Networks [4] that can learn from game-play replays, which will be covered in more detail later on in the paper.

RTS is a great test bed in AI research for its complex systems, involving many areas of interest in planning, dealing with uncertainty, domain knowledge exploitation, task decomposition, spatial reasoning, and machine learning [5]. Unlike turn-based strategy, RTS requires real-time decision making with imperfect information, the information is limited through the use of partial visibility of the map. Unless the AI scouts the map (Sends a unit around the map) and sees what the opponent is doing, then the AI will have no access to any strategic knowledge. This along with the non-deterministic nature of RTS, meaning it may not exhibit the same behaviour on each run, makes RTS one of the most challenging environments in which to create an AI [6].

Since the release of StarCraft Brood War API it has been easier for Academics to research AI in StarCraft, this has also given rise to an educational value as part of AI related subjects in several Universities around the world [6]. One example of this is the University Delft (NL), which for one of its modules the students are required to create a StarCraft Bot [7]. From this three yearly competitions have been created to compete students AI's, the first of which was hosted by the University of California, Santa Cruz in 2010 as part of the AAAI Artificial Intelligence and Interactive Digital Entertainment (AIIDE) conference program [8]. Another hosted at the IEEE Computational Intelligence in Games (CIG) conference [9], and the last one which is an ongoing stand-alone event is the Student StarCraft AI Tournament (SSCAIT) [5]. Upon successful completion of this work, the AI will be submitted to one of these three competitions.

This paper is organised as follows, first StarCraft and what it is will be presented, followed by a review of the current methods being utilized by the research community in the development of StarCraft AI's. With a description on the research that this work will be using, this is proceeded by the method and tools that will be used to create the AI as well the metrics used to measure its effectiveness finishing with the hypothesis.

Look back to this once conclusion is done to ensure they match

II. Related Work

A. StarCraft

StarCraft is an RTS game developed by Blizzard Entertainment [10] and popular for testing AI [6], the game was released in 1998 [11]. Later that year StarCraft: Brood War was released and took hold in the e-sports community and is still popular today. StarCraft 2: Wings of Liberty was released much later in 2010, with a complete visual overhaul, most of the game mechanics remained the same other than balance changes. The premise of StarCraft is to gather resources, build a base, and build an army to then use to destroy an enemies base and army, during playtime, there are also many upgrades available for these units to give them the edge over an enemy who did not spend the time acquiring them. There are many ways of doing this each player with a different order of building their armies/bases commonly referred to as their "Build Order" [12]. Build orders refer to a players macro-management, whereas in StarCraft Micro-management is a huge part of the game, as those with greater control over individual units can better outmanoeuvre their opponent, and thus defeat them. There is a difference in the way units are controlled, in StarCraft:BW you can only select up to 12 units at a time and can not group them for easy selection, so when playing you have to utilise micromanagement skills more than in StarCraft 2 where you can select an unlimited number of units and can group them for easy selection. StarCraft is considered difficult due to its requirement of abstraction level thinking when planning. Strategy selection is perhaps the most important choice any player or AI can make in StarCraft and RTS as a whole, as this will dictate the actions and reactions which they take during playtime. Though a human player can be proficient at choosing their strategy by simply scouting the map, finding the enemy and seeing what they are building. The human player can then counter accordingly, and if they countered incorrectly the human player can simply change their strategy to accommodate. Creating an AI to do the same though can be a huge and complex task [13]–[15], one way to achieve this result without a large commitment of time is to create a library of expert strategies, and allow the AI to select the appropriate one throughout the game. This can be achieved using tools such as Advanced Behaviour Oriented Design Environment (ABODE) and Parallel-rooted Ordered Slip-stack Hierarchical (POSH) reactive plans [16], which will also be covered later in the paper. These tools allow for an iterative design approach for AI's in games and in this work will be focusing on the macro-management with a particular focus on build orders and the selection of strategies rather than micro-management.

In the StarCraft research community, there are many different methods of AI creation. Some focus on micro-management like S. Liu et al [17] that uses a Genetic Algorithm (GA) and others that focus on macro-management looking at the build order like N. Justesen et al [14]. Many of these research methods are cross depended and utilise more than one method, for example, D. Churchill et al [18] created the UAlbertaBot, which was intended to automate both build order planning and unit control. There are also AI's that only use one strategy that has won several times in competitions like the ZZZKBot [19], [20], which only uses a 3pool build and built that uses a rush tactic. This rush tactic involves creating many weak inexpensive units and sending them to the enemy base within 5 minutes of starting the game. Many AI tend to struggle with countering this strategy, hence why this type of AI tends to win.

B. Datasets

A Dataset can be a collection of any data, for game AI a dataset can consist of thousands of replays with millions of game frames, and player actions [21]. This information can then put together to create a full game-state which allows for machine learning tasks [22]. In AI research, datasets can be used in many approaches of development, one such use is to recreate game-states and evaluate them for prediction in realistic conditions [23].

C. Micro-Management

Micromanagement is a fundamental side of StarCraft gameplay and many papers have their own approach to this aspect of RTS [17], [23]–[27]. Micromanagement is the control of each unit individually, for example: if you have 12 units, each with their own ability, during battle you need to activate each ability at the correct time for each of the 12 units in order to utilise them to their full potential. This required you to select each unit during battle and activating the ability, while still maintaining control over the other 11 units. Though this is a slight exaggeration as in StarCraft some units have an auto use of their ability which allows the unit to decide when to use their ability, one such unit being the medic on the Terran faction which will heal any biological unit with less than full health. Also in StarCraft units can be selected by type i.e. you can double-click on a marine, and it will select all the marines on screen (Up to a maximum of 12). Players that perfect multitasking micromanagement skills are most likely to win the battles when playing, as they can outmanoeuvre their opponent much more easily and use abilities effectively to devastate their opponents armies. Many of these approaches tend to use either Genetic Algorithms (GA) or Evolutionary Algorithms (EA) [17], [24], [25], while others observe replays and apply a Monte-Carlo method to create data for practice use [26]. But most of these methods have one thing in common, they all use a version of machine learning [2].

D. Predictive Methods

On a higher strategic level, the prediction of the opponent's strategy is a prominent approach used in research [28]–[31]. This type of research relies on the use of replays and machine learning to help the AI accurately predict a strategy, these do rely on the quantity and quality of replays used for the learning process [28], [29], [31]. Another method for prediction is scouting alongside machine learning, this eliminates the need for replay observation and allows for a more real-time prediction [30]. Though this method does still require several games to be played before the AI can begin to have an accurate prediction.

Bayesian approaches are based on Bayes' Theorem which is another prediction method. Bayes' Theorem is a calculation of probability or also known as a probabilistic model [32]. In papers by G. Synnaeve et al [23], [27] they create an AI that controls units individually, they do this by using uncertainty which instead of asking where a unit might be, it makes rough estimations and acts upon that. Another use for the Bayesian approach is to predict strategies, by creating a probabilistic model that after learning from replays can predict an opponent's strategy and adapt accordingly [29]. A major downfall of Bayesian Approaches is that it can be computationally intense to calculate.

E. Full Game Play

Many papers try to create an AI capable of handling all aspects of an RTS [18], [33]–[35]. These AI's tend to take several methods that have been created in other research and combining them to form a new AI [18]. Another use for the full gameplay AI is to try and create a "Human-Like" AI, which can mimic the play-style of an expert human player. Though the current AI's are limited in this as players reported that the AI's used unusual unit movements or building placement [36].

F. Neural Networks

Neural Networking are computational models loosely based on the functioning of biological brains [4]. Given an input it computes an output by using a large number of neural units, in StarCraft it can be used to predict strategies or in the case of StarCraft 2 with its new architecture it can be used for full game-play. Using a neural network would be impractical for the purpose of this work as it would take many months to train, and even then would not have a great chance of doing well against other AI's.

G. Planning

Planning in StarCraft usually deals with the build order that the AI will use usually only dealing with macro-management. There are several different ways to use a build order, some will use a static build order that will not change throughout the game [3], and the more popular route is to allow the AI to jump between build orders during play-time, another term is Reactive Planning [13]–[15]. there has been some work on creating the build orders on the fly by finding out that most optimal method of gathering resource and building units [12]. Planning is perhaps the most optimal approach to creating an AI as there are little real-time calculations to make. Through the use of POSH tools [16], you can iteratively design AI prototypes and deploy quickly [3].

From looking at the research in the field there are many methods that can be used to create an AI. The use of replays to train an AI to counter strategies can be effective [29], they lack the greater control of the game, the ability to macro-management as there are too many variables to consider. This lack of large-scale control is usually due to the heavy computational requirements of controlling each individual component of the game. Due to this slow process, it is quite impractical to use when there is already a library of knowledge that can be to exploited [37]. Though there are AI's out there with planned strategies already programmed into them [13], [19], their limitation is that they only use a small number of strategies, though these work it can leave a lot of room for the opponent to manoeuvre. A logical step here is to program a larger pool of expert knowledge into the AI, it will then select one and follow it through, with the ability to jump between strategies at key points in-case a counter is detected.

H. StarCraft AI's

In the StarCraft AI community there are many AI's that have been created to compete against each other, and in this work, a competition AI is defined as an AI that has been entered to the AI for Interactive Digital Entertainment conference (AAIDE) StarCraft AI Competition. A yearly competition hosted by David Churchill and sponsored by AIIDE. Examples of the top AI's from the competition include:

- ZZZKBot Winner of the 2017 AIIDE StarCraft AI Competition [19]
- Iron Winner of the 2016 AIIDE StarCraft AI Competition [38]
- UAlbertaBot Winner of the 2013/2011 AIIDE StarCraft AI Competition [39]
- Skynet Winner of the 2012 AIIDE StarCraft AI Competition [40]

Each of these bots employ different strategies, these strategies a

Discuss the other bots and their strategies

I. Research Questions and Hypothesis

1) Hypothesis:

- **Hypothesis 1** The AI will survive longer more than 13 minutes and Win.
- **Hypothesis 2** The AI will survive no more than 13 minutes and Win.
- **Hypothesis 3** The AI will have a Win rate greater than 0%.
- **Hypothesis 4** The AI will have a Win rate of 50% or greater.
- **Hypothesis 5** The AI will counter the Rush tactic.

2) Research Questions: During the initial research into this area for this paper other research questions were reviewed:

- Combining Behaviour Oriented Design and Expert Human Knowledge to create a competitive AI.
- Creating an adaptive AI using predefined expert strategies.
- Is an adaptive AI built with predefined expert strategies a viable competitor against non adaptive AIs?
- How effective can an adaptive AI built with predefined expert strategies be against other AIs?

These questions were rejected as creating an adaptive AI was not the focus of this research.

III. Method

Empirisism

Include some philosophy on this paper (positivist research)

This research employs a positivist approach as all results will be interpreted through reason and logic,

In this work Protoss will be chosen, this choice was made as Zerg have been done many times and usually rely on rushes to win, whereas Terran are too complex for the planned AI. Protoss proved a happy middle ground to build upon where rushes are difficult to achieve but they are not too complex that it is too difficult to execute a sound strategy. So the second logical step is to implement an anti-rush strategy as an opening strategy. From there more aggressive strategies will be implemented and executed at the appropriate times. The challenge here is that it is difficult for Protoss to counter a Zerg rush but if successful it will leave the Zerg open for attack. To support this decision the ratio that the races won were looked into and can be seen in fig 4, here the Protoss clearly loose more often than the other two races.

A. Tools

The tools that will be used in this experiment are; The Brood War Application Programming Interface (BWAPI), POSH tools, specifically POSH Sharp which is an interface that uses c# instead of C++, and the ABODE editing software which uses POSH plans to create Behaviour Oriented Designs for AI's. Other tools include Visual Studio 2010, Chaoslauncher, StarCraft Tournament Manager, and VirtualBox, all of which will be explained next.

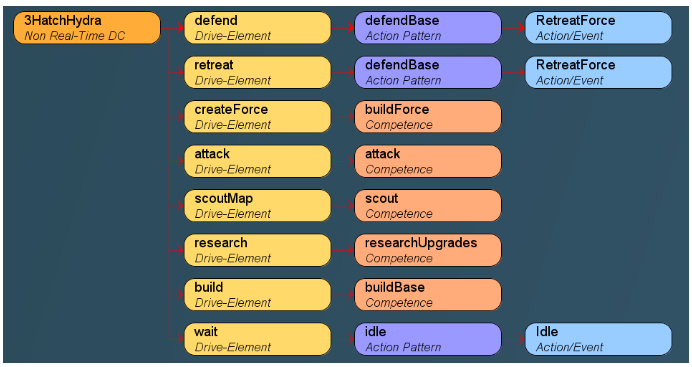

Fig. 1: POSH plan for the Three Hatch Hydra build plan inside the ABODE editor.

- **Brood War Application Programming Interface** [41] is an open source software that creates an interface to allow a custom AI to communicate with the game. BWAPI only give limited information to the AI, which inhibits the bots to have the ability to know what its opponent is doing, this means that the fog of war(the unexplored parts of the map) are is kept [16], this mean that the AI is just a limited in its knowledge as a player would be. The information that is provided is the size of the map and base locations, this allows the AI to have the ability to scout effectively. This limited information prevents custom AI's from cheating and ensures a fair game, though for the developers of the AI this could be considered an advantage for the development stage of the AI as there is no need to be concerned with accidentally allowing their AI to access illegal information. This does however provide a challenge in design as the AI is dealing with imperfect information, it must be designed in such a way that it almost replicated human responses, i.e. scouting, and checking areas already scouted for enemy presence.

- **POSH** plans can be created in the ABODE Environment as seen in Figure 1, these are visual planning tools that allow for a hierarchy of actions with associated triggers. Each plan can be split into three parts, Drive-Collections, Competencies and Action patterns, these three determine when an action is to be triggered. POSH plans use a behaviour library created in the native language of the problem space, see Fig 4. This tool can be a powerful asset to designing and creating an AI as once the behaviours and senses are implemented new AI can be created quickly and with little error.

- **Microsoft Visual Studios 2010(VS2010)** is an integrated development environment(SDK) from Microsoft [42]. It is used to create computer programs, as well as websites, apps, and online services. In this experiment VS2010 is being used to create the behaviours for the AI, as well as any other functionality the AI requires, this includes the framework for the POSH wrapper.

- **Chaoslauncher** is an open source third-party launcher for StarCraft that allows the user to inject any universal plugins [43]. For this experiment the launcher will be used as a debugging tool for the first stages of the AI. The launcher also allows StarCraft to be run in windowed mode alongside a BWAPI injector that allows the AI to communicate with BWAPI.

- **StarCraft AI Tournament Manager(STM)** is a tool that was developed by David Churchill to manage and run StarCraft AI tournaments, it is an open source project available for anyone to use. It runs the tournament by creating a server and allowing instances of its counterpart (the Client) to connect to it, each client runs a single instance of StarCraft and the server will put two clients into a game and record the results. This set-up allows for as many instances of StarCraft to be run on the server as the user wants, unfortunately the current set-up of the STM doesn't allow for more than one instance of StarCraft to be run on a single PC at a time. To solve this a virtual machine will be utilized [44]. For the purposes of this experiment the STM will be used to test the AI against the competition AI's, once the tournament is over the STM will compile a results table similar to fig 2, in a HTML format which can be opened in any browser.

- **Oracle VirtualBox** is a general-purpose full virtualizer for x86 hardware, targeted at server, desktop and embedded use [45]. This allows the user to run multiple instances of an operating system on the same hardware, for this experiment it will be used to run multiple instances of StarCraft on the same PC.

- **R** is a language environment for statistical computing, using R along with R studio the user can compute statistical equations and produce the appropriate graphs. For the purposes of this research R will be used to create some of the figures present in this paper, a code exert of R can be seen in fig 3.

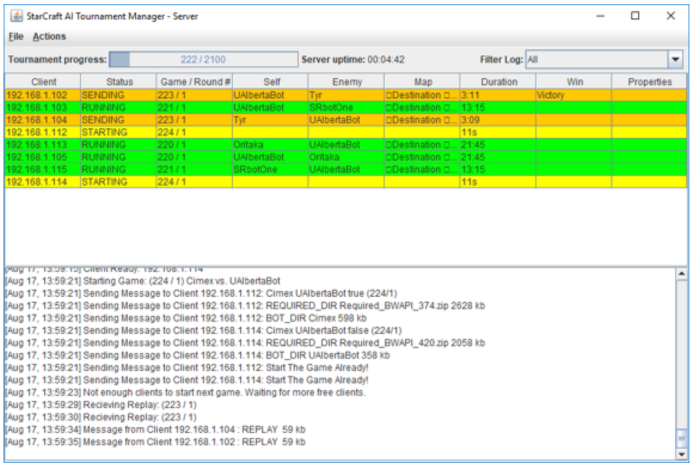

Fig. 2: StarCraft Tournament Manager Server Running.

```
[ExecutableAction("SelectProbeScout")]
public bool SelectProbeScout()
{
    if (probeScout != null && probeScout.
        getHitPoints() > 0)
        return true;

    Unit scout = null;
    IEnumerable<Unit> units = Interface().
        GetProbes().Where(probe =>
    probe.getHitPoints() > 0 && !Interface()
        .IsBuilder(probe));

    foreach (Unit unit in units)
    {
        if (!unit.isCarryingGas())
        {
            scout = unit;
            break;
        }
    }

    if (scout == null && units.Count() > 0)
    {
        scout = units.Last();
    }
    probeScout = scout;
    return (probeScout is Unit && probeScout
        .getHitPoints() > 0) ? true : false;
}
```

Fig. 3: C# executable action for selecting a probe scout, the plan will execute this code when triggered.

B. Design and Research Artefact

Designing this experiment was a simple, yet time consuming task once the research question was settled on, as the only designing required was to obtain several expert strategies. These strategies were obtained from an online source Liquipidiea, this is an online wiki available to the esports community to bring together all the information they can to help each other in their respective sports. This wiki is a valuable source of knowledge when trying to obtain the necessary StarCraft strategies as the ones on this site are used by the experts that play the game.

The challenge was the implementation of these build orders, as the POSH plans have to be precise, meaning the priorities of its actions had to be correct, plus the timing of each action needed to be correct.

This work will be focusing on the implementation of an AI with pre-built build orders and their counters taken from Liquipedia [37], a website dedicated to StarCraft, on the there they have a collection of strategies that are free to use in any capacity. Implementing these build orders are covered in greater detail in Iteration 5 : Implementation of New Strategies.

```
qplot(input$`Win%`, input$Duration, geom = c("
    point",
"smooth"), method="rlm", main="Correlation
    between Win % and Game Duration", xlab="Win
    %", ylab="Game Duration")
```

Fig. 4: R code to create a scatter plot with a line of best fit. The figure created from this exert can be seen in Figure 9.

For the creation of the artefact there were many option for life cycles that could have been adopted. Chief among these were the incremental model and Agile, both supported a short development cycles, but lacked the ability to revisit and refactor code [46]. For the incremental model each development cycle is static and isolated, meaning that when the cycle is over the build will never be revisited again. This approach would be a impractical for this artefact as the strategies and variables in the code need ot be constantly re factored and tweaked.

The Agile development cycle seemed like a better choice as it is more friendly towards an iterative development cycle that produces smaller chunks of code. Though Agile still lacked the focus on testing first, this was required for this artefact as with each implementation it needed testing to confirm if the implementation succeeded. For this artefact Agile will be combined with Test Driven Development (TDD) as there is no use of daily scrums, plus the focus of this research is to have a working AI, and the best method to achieve this is to continuously test and refactor. This method is called Agile Model Driven Development (AMDD) which focuses on iterative development while also being test driven, a more detailed description of this can be found in the appendix under figure. Another reason for AMDD is that the constant small changes to the code would dramatically effect the functionality of the AI, therefore every change needs testing.

For testing the only concern was whether a function within the code did as was designed, there is only a pass/fail did it do the thing it was designed for or not, when testing these only one test is usually required to ge the answer then a slight modification was made each time if it resulted in a fail. The tests had only minor automation as the artefact had to be manually run and the game manually set-up.

1) Iteration 0 : Software Installation And Design: Before any development was carried out, the first decision was how was the AI to be created, it could either be done in C++ pure, Java or C# using POSH-tools. PODH tools were chosen as

The first sprint was to set up the environment. This involved downloading and installing several versions of Visual Studio, and downloading the correct version of BWAPI, Chaoslauncher, and the addition of PoshSharp. An intimidate issue that came up after attempting to compile BWAPI, there were missing .ddl's in the Windows directory, to rectify this the relevant dll's were manually copied into the relevant locations. Once the coding environment was set up, the next step before any code was written was to set up the testing environment. This was done using the Chaoslauncher, BWAPI had to compile correctly to begin with otherwise the AI would not inject into the game correctly. Once StarCraft launched with no issues, a basic POSH plan with no functionality, that came with POSHSharp was compiled and executed, this all had to be done in admin mode as it would not work correctly. By the end of this sprint the correct software was installed, the testing environment was working, and a basic plan ran in the game. The artefact at this point was ready to be designed and have code written in the behaviours.

Before any design could commence a race had to be chosen, in StarCraft, there are three, Zerg, Protoss and Terran, each with their own unique play style. Zerg is a rushing faction, with their units being relatively weak and cheap, Zerg usually focus on overwhelming their enemy with numbers. Protoss are strong but expensive, relying on smaller numbers and taking longer to produce anything, this means they can be weak at rushing and defending from a rush. Terran is a balance of the two, being able to produce strong and expensive units as well as cheap, weak ones, they can effectively rush and defend from a rush.

A rush is a tactic employed in RTS games which involves building up a small force as quickly as possible to harass the enemy base and units. A rush is only considered a rush if it is done within the first five minutes of the game [47].

Once the software was installed, the method of development was chosen and Protoss was picked, the next step was to begin testing and developing.

2) Iteration 1 : Prototype: The artefact needed to have an executable plan for the desired race, in this case Protoss, the reasoning for this is explained in the Method section, under Preliminary Results. No behaviours were changed, this was simply an exercise to ensure the testing environment ran with the correct race, so only basic functionality was present. In this case the Probes would gather resources and build a Pylon. This would be the final step in setting up the tests, as now the Protoss ran with no issues in game and testing would be seamless.

3) Iteration 2 : Alpha: Creating an alpha involved changing the behaviours to suit the race and the behaviours that came with the software were written for Zerg only. Which meant that there had to be a lot of redesigns and code re-factoring needed. And example of this would be the positioning of buildings, as for Zerg they can only build on something called "Creep" this is present at the start of the game at a set radius around the starting base, and can be extended through the use of special structures. For Protoss however they can only build within range of a structure called a "Pylon", these Pylons can be built anywhere but any other structure bar the starting structure has to be built within range. The behaviour for the placement of structures for Zerg worked fine for them however for the Protoss the function had to be refined for more precision when building.

Another example is when building structures the Zerg loose a builder, as the builder "Morphs" into the structure so each time something is built the AI would just select a new builder and remove the previous one from any list it was related to. Protoss on the other hand can have many structures constructing at one time, this means that the build must remain the same unit. Plus the Training of units, as Zerg only trained from one structure where as Protoss would train from several.

This was done over several weeks, every behaviour, action and sense that was modified/written was tested for functionality each and every time there was a change. This allowed the artefact to take small steps with each change and test always progressing. Once a piece of code was complete there was rarely a need to return to it, and if there was then it was a simple matter to make a change and test if it worked. By the end of the sprint the AI was building in the correct places and produced units from another structure.

4) Iteration 3 : Beta: During this sprint the objective was to ensure the AI could build in other locations as well as build an army to attack the enemy. The first goal was to create a method for the AI to find the choke point and set it as a base location for building, this also opened up an issue with ensuring the AI can swap building locations. Within BWAPI the AI has knowledge of all the choke points and base locations, though they are not allowed to access them unless it has been revealed on the map. then once it has the AI can save that location, and once it needs to build there it will use the positioning code from that location instead of from the start base.

Once the functionality for building at the choke was implemented and tested, a plan with greater detail was created. Meaning that the AI would now take advantage of building at both the starting area as well as the natural expansion and choke point. Along with this the AI had to build an army and attack the enemy, using the new method for finding new build locations, it was a simple modification to the scouting function to allow the AI to find the enemy base and mark its location, which allowed the AI to know where to go with its forces.

During this sprint it became clear that the strategies provided from Liquipidia were not going to work, as they relied on the player (AI) scouting and building immediately as the game started. Unfortunately as the system can not compute its actions instantaneously the AI can not execute the strategies in time before the enemy AI had wither rushed or build a larger force. This proves my null hypothesis in these conditions, being the framework is too slow and needs altering, and as this was not the focus of this research would have been an improper use of resources.

To counteract this The AI has been modified to include a tactic that is not present in any of the strategies, which is to build several Protoss Photon Cannons at the natural expansion. This means that the base immediately next the

the AI's starting position will be built up with defensive structures to stop any early game threat, allowing the AI to continue with the original strategy proposed.

5) Iteration 4 : Polish: At the end of this sprint the AI was expected to have all behaviours fully implemented and tested and a completed plan written and tested. This was an opportunity to look back and re-factor any code that needed tuning or any duplicate code that got into the system. Though the point of TDD is to avoid duplicate code, its priority is to get code that works, in the case of this software duplicate code did end up in the system. Unfortunately due to the nature of the "Action", "Sense" system there is bound to be duplicate code, though it can be minimized with internal functions. Tuning the plan took most of the time in this sprint as with each change the plan had to be tested within the game, to ensure that th change was meaningful and effective. This usually involved changing the build order and priority order of actions.

6) Iteration 5 : Implementation of New Strategies: The final sprint was to implement several plans that could work alongside the base plan, these would all be taken from the Liquipidiea site. Unfortunately the current set-up only allows for one plan to be used at a time, though each plan can contain several strategies that are triggered when the AI plays. Each strategy was tailored to face each race as an opponent, three strategies were chosen from Liquipedia, one for Protoss VS Terran, VS Zerg, and VS Protoss, When two races are the same it is refereed to as a mirror match.

Each strategy focuses on the same goal, producing a unit called Dark Templars (DT) as quickly as possible. This Unit is cloaked and has a clear advantage over an enemy that has no detectors (detectors are a unit that can reveal cloaked/hidden units). The process to achieve this can be seen in figure 11 in the Appendix, which shows the tech tree for the Protoss. Though each strategy gets to this goal in different ways:

- For Protoss VS Zerg the AI will use a strategy known as "One Base Speedzeal (vz. Zerg)" [48], the focus is to produce Zealots and as soon as possible produce DT's, once the DT's are in production the Zealots are to move to the Zerg base and harass them until the DT's arrive. At which point the Zerg should either be crippled or defeated, though smart building placement by the Zerg can counter this, then to counter the counter, the Protoss must recover as quickly as possible and renew the attack, or at least continue harassing the Zerg until the Protoss has a large enough army to win.

- When facing Terran the AI will use a strategy known as "2 Gate Dark Templar (vz. Terran)" [49], the AI must forego the Zealots and head straight into DT's, the means that the Protoss are prone to early rushes by the enemy, but to defend against this one or two ranges units called Dragoons are trained to defend. Once the Protoss has two DT's they are sent into the enemies natural expansion to harass them and lock them in their base unit the Protoss are ready to finish them off.

- The hardest strategy to implement is for the mirror Protoss VS Protoss, as they both can produce the same units at the same rate. The AI will use a strategy known as "2 Gateway Dark Templar (vz. Protoss) [50], DT's are again used in this instance. Though this strategy employs tactics from both other strategies, building Zealots and Dragoons for both attack and defence. Photon cannons (A defensive structure) are used to help defend against any attacks. Once the initial attack was defended, and the natural expansion taken the focus shifts to mass producing units, and sending them in too attack the enemy, it is down to how the enemy is player which dictates what unit to produce.

Each strategy was implemented separately and as each action taken by the AI had to be precise and done at the correct time, not in game time, but when a building was completed then another action has to be taken. With each implementation the whole process began again, with a prototype of the plan, followed by a beta, alpha, then polishing and finally implementing the next strategy. The only difference at this point was that all the behaviours were implemented and only the plan followed this development cycle, this allowed for all three plans to be implemented, tested and working in a relatively short period of time.

Also as stated on the Liquipeidia website, these build orders can be changed to suite the needs of the player, in this case AI. For example, to fight againt the Zerg rush, a forge and photon cannons were built at the beginning of the game, even though the build order did not call for it. Changes like that one were made to each of the strategies, though the underlying direction of each strategy remains the same [51].

C. *Preliminary Results*

To prepare for final testing a preliminary test was carried out using the tournament manager, this was done to ensure the software was set-up and working for the final testing for the AI developed for this research. For the preliminary test 9 AI's were chosen from the AIIDE 2016 competition, three from the top, three from the middle and three form the bottom. These bots would then play one on one games on 10 maps, totalling to 360 games, the results of each match were recorded automatically by the tournament manager and compiled into table 1, from this table fig 4-8 were created. These results allowed the further development of this researches metrics, this is explained in the following section.

Bot	Games	Win	Loss	Win %	AvgTime	Game Timeout	Crash	Frame Timeout
Iron	80	68	12	85	13:03	0	0	0
ZZZKBot	80	68	12	85	6:05	0	0	0
LetaBot	80	55	25	68.75	14:09	2	0	0
Xelnaga	80	45	35	56.25	15:53	3	3	0
IceBot	80	43	37	53.75	14:33	0	7	0
MegaBot	80	38	42	47.5	13:28	2	8	7
Cimex	80	20	60	25	17:10	5	2	0
CruzBot	80	14	66	17.5	19:24	10	0	1
Oritaka	80	9	71	11.25	15:20	4	0	0
Total	**360**	**360**	**360**	**N/A**	**14:20**	**13**	**20**	**8**

TABLE I: The HTML results table produced by the StarCraft Tournament Manager [44]. Blue represents Terran, Purple represents Zerg, and Yellow represents Protoss

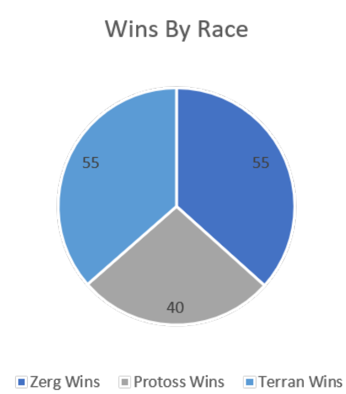

Fig. 5: A pie chart representing the average win rate % for the three races, based on the results obtained from table 1.

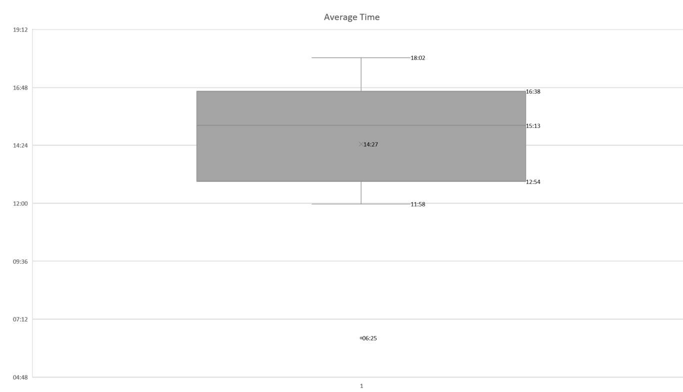

Fig. 6: Box and whisker showing the average game length, with the mean, median and outlier based on the results obtained from table 1.

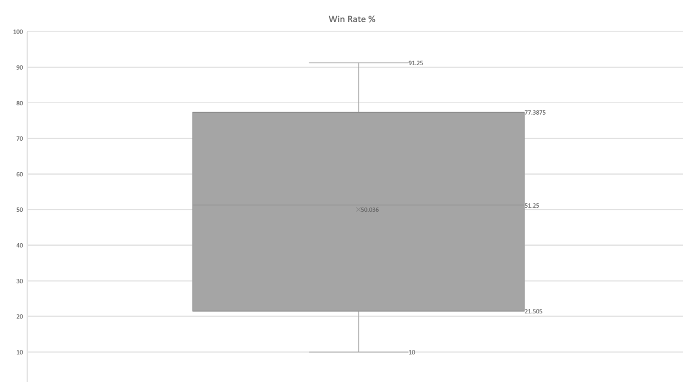

Fig. 7: Box and whisker showing Win Rate, with the mean and median based on the results obtained from table 1.

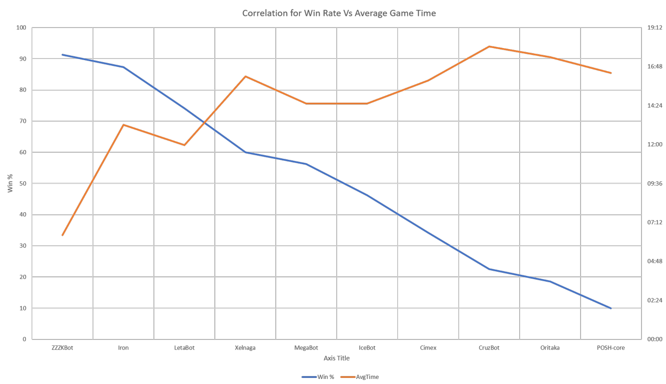

Fig. 8: A line graph showing both the win rate and game length for all the bots, starting with the highest win rate on the left, based on the results obtained from table 1.

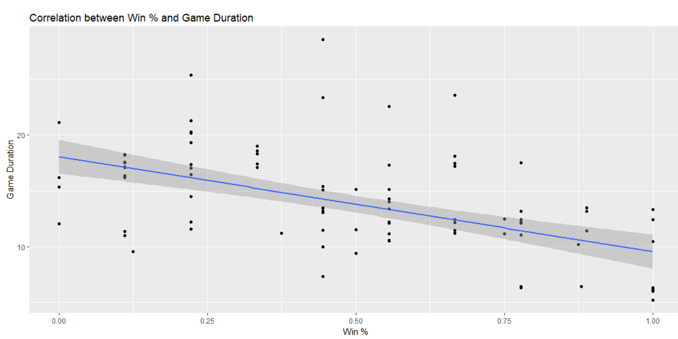

Fig. 9: A scatter plot showing a negative correlation between the win rate and game length, based on the results obtained from the preliminary experiment. Each dot represents an average win rate and average game length for each bot during each round.

D. Metrics

Initially the StarCraft AI was to be measured on its success in these two metrics:

- **Time Survived** (Average of 13 minutes or above)
- **Endgame Condition** (Whether the AI wins the game or looses)

These metrics were obtained from the results from the AIIDE 2017 StarCraft Competition, which found that the average time of each game was 13-minutes, and the quickest average being 8-minutes and the longest being 19-minutes. The win ratio of the AI's vary substantially from 17.21% to a high 83.11% as can be seen in Table 2 [20].

After presenting these findings, it was discussed that the metrics required a greater level of merit, to achieve this, the metrics were modified using the preliminary results and further inspection of the AI's within the literature. As can be seen in fig 6, the mean win rate for the AI was 50% and the mean game length from fig 5 was 14.20, and within the literature the only concern for some papers were win rate, these range from 54-91% [3], [14], [15], [52]. The literature AI's were faced against both competition AI's and in-built AI's, which means their win rates could be inflated.

No papers that were reviewed for this research focused on average game length, though for the purposes of this research it will be. As can be seen in fig 8 and 9 there seems to be a negative correlation between win rate and game length, which supports the metric.

For the final metrics that the AI will be measured on will be as follows:

- **Average Game Length** (Average of 14 minutes or below)
- **Average Win Rate** (Win Rate above 50%)

Through these revised metrics, the effectiveness of the AI will be determined, if the AI obtains an average game length of 14 minute or less, and or has a win rate of 50% or higher it will support the statement that it is effective. Though if the AI fails at achieving these goals it will support the NULL hypothesis.

E. Testing

The AI was faced against the open source competition AI's, and 400 games were simulated, 80 of which the AI presented in this paper took part in. This method of testing was chosen to create a separate data set from that of the annual competitions, and provide the AI an environment similar to that of

During testing there will be a total of 10 AI's including the AI that was developed for this research. To choose these AI's three were selected from the top of the board, three from the middle and three from the bottom. They will be selected from the AIIDE 2016 competition, to keep the experiment af fair as possible each AI in each tier will consist of each of the races, where that is not possible, a substitute will be made [53]. AIIDE 2016 was chosen as all the bot's required for testing were available for download.

The 10 AI's will be playing on a total of

Mention that the testing will be within a competition environment.

Talk about how and what you are testing

F. Disadvantage of Software

Though once this system is complete with all the behaviours needed are present, makes it quick and easy to create new AI plans, it has one major disadvantage. The AI can only execute one action at a time, for example, when the executes the "ProbeScoutToEnemyBase" action it must wait for the entire action to be executed before moving on to the next. This has a negative effect on the design of the AI as it must be created in such a way that it can be effective while being limited to only one action at a time.

Another issue presented by this software is that it is CPU intensive and slows down the game simulations considerably, this will make testing time increase and unfortunate there is not much that can be done about it, as the c# is being translated into c++ for the BWAPI to translate into StarCraft commands.

IV. Results

Describe the results do not discuss

Bot	Games	Win	Loss	Win %	AvgTime	Game Timeout	Crash	Frame Timeout
ZZZKBot	80	73	7	91.25	6:25	0	0	0
Iron	79	69	10	87.34	13:13	1	0	0
LetaBot	81	60	21	74.07	11:58	0	0	0
Xelnaga	80	48	32	60	16:11	4	1	0
MegaBot	80	45	35	56.25	14:31	5	10	2
IceBot	80	37	43	46.25	14:31	2	1	0
Cimex	79	27	52	34.18	15:56	5	2	0
CruzBot	80	18	62	22.5	18:02	9	0	0
Oritaka	81	15	66	18.52	17:22	9	0	0
POSH-core	80	8	72	10	16:24	7	0	0
Total	**400**	**400**	**400**	**N/A**	**14:27**	**21**	**14**	**2**

TABLE II: The HTML results table produced by the StarCraft Tournament Manager [44]. Blue represents Terran, Purple represents Zerg, and Yellow represents Protoss

Race	# Matches	Wins	AVG POSH Score	Std Dev	AVG Oponent Score	Std Dev	AVG Score Diff
Zerg	17	1	29370	17778	33518	19071	-14.12%
Terran	36	6	24070	36436	26276	19881	-9.17%
Protoss	27	1	7346	4652	23168	15537	-215.37%
Total	80	8	19552	27222	26766	18617	-36.90%

TABLE III: Results from the 80 matches that the POSH-bot described in this paper took part in against the competition AI's on three 2v2 competition maps.

V. Discussion

Talk about the results and the implications

VI. Professional Considerations

VII. Future Work

Though the null hypothesis was confirmed, this work provided an interesting find, that there is a possible correlation between the win rate and win time. A proposed question for future work would be if there is a correlation between them. This would be an interesting find as it could be used to alter the design of AI's to facilitate fast strategies, and through the Zerg implemented a rush tactic which is inherently fast, it would be interesting to see how and AI using Terran or Protoss would cope with similar strategies. Altering the artefact to have executable time set as a priority during development would be an advantage for this as the current software can not keep up with the other AI's in its current state.

An alternative option to solve this issue is to create another plan to run in parallel to the original. This will allow the AI to execute multiple behaviours at the same time, rather than executing one and having to wait for it to finish before moving on to the next. This would, for example, allow the AI to build up their base, scout and manage their army at the same time. Unfortunately it would involve precision planning as each plan would have to ensure they do not execute the same action at any point. This was not the primary focus of the artefact, as the framework would have to be modified to allow the use of parallel plans.

Using parallel plans would provide an interesting premiss for designing an AI. The AI could have multiple plans, each focusing on a separate component of the game, these could be combat, base building, resource management or defence. Creating an AI with this capability would provide a challenge in its design as the plans would have to communicate with each other to allow the correct execution times, and it would have to ensure there are no conflicts of interest when managing units. This could be solved with a order of priority or a threat calculation, so the defence can take over from the combat plan if the AI is loosing its base.

A further step that could be taken is to allow the AI to construct its own plans with neural networking. It would allow the AI to learn from pre-built strategies and alter them accordingly as it played a number of matches.

VIII. Conclusion

IX. Acknowledgements

References

[1] M. Bruro, "Call for ai research in rts games," in *In Proceedings of the AAAI Workshop on AI in Games*. AAAI Press, 2004, pp. 139–141.

[2] S. Ontan, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "A survey of real-time strategy game ai research and competition in starcraft," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 5, pp. 293–311, Dec 2013.

[3] S. Gaudl, S. Davies, and J. Bryson, "Behaviour oriented design for real-time-strategy games: An approach on iterative development for sc starcraft ai," *Foundations of Digital Games (FDG)*, pp. 198–205, Jun 2013.

[4] N. Justesen and S. Risi, "Learning macromanagement in starcraft from replays using deep learning," in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, Aug 2017, pp. 162–169.

[5] D. Churchill, M. Preuss, F. Richoux, G. Synnaeve, A. Uriarte, S. Ontanon, and M. Certicky, "Starcraft bots and competitions," in *Encyclopedia of Computer Graphics and Games*, 01 2016, pp. 1–18.

[6] D. Churchill and M. Certicky, "The current state of starcraft ai competitions and bots," in *AIIDE 2017 Workshop on Artificial Intelligence for Strategy Games*, 2017.

[7] U. of Delf. Project multi-agent systems. [Online]. Available: https://www.tudelft.nl/en/education/programmes/bachelors/ti/bachelor-of-computer-science-and-engineering/curriculum/ [Accessed 05-12-2017]

[8] D. Churchill. Aiide starcraft ai competition. [Online]. Available: http://www.cs.mun.ca/ dchurchill/starcraftaicomp/ [Accessed 17-11-2017]

[9] K.-J. Kim and S. Yoon. Cig starcraft ai competition. [Online]. Available: https://cilab.sejong.ac.kr/sc competition/ [Accessed 17-11-2017]

[10] Blizzard Entertainment. Starcraft. [Online]. Available: http://eu.blizzard.com/en-gb/games/sc/ [Accessed 03-11-2017]

[11] ——. 10 years of starcraft. [Online]. Available: https://web.archive.org/web/20080402134120/http://www.blizzard.com/us/press/10-years-starcraft.html [Accessed 16-11-2017]

[12] D. Churchill and M. Buro, "Build order optimization in starcraft," *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2011.

[13] M. Preuss, D. Kozakowski, J. Hagelbck, and H. Trautmann, "Reactive strategy choice in starcraft by means of fuzzy control," in *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, Aug 2013, pp. 1–8.

[14] N. Justesen and S. Risi, "Continual online evolutionary planning for in-game build order adaptation in starcraft," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2017, pp. 187–194.

[15] B. G. Weber, M. Mateas, and A. Jhala, "Applying goal-driven autonomy to starcraft." in *Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. AAAI, 2010.

[16] C. Brom, J. Gemrot, M. B ida, O. Burkert, S. J. Partington, and J. Bryson, "Posh tools for game agent development by students and non-programmers," in *9th International Conference on Computer Games: AI, Animation, Mobile, Education and Serious Games*, Jan 2006, pp. 126 – 135.

[17] S. Liu, S. J. Louis, and C. Ballinger, "Evolving effective micro behaviors in rts game," in *2014 IEEE Conference on Computational Intelligence and Games*, Aug 2014, pp. 1–8.

[18] D. Churchill and M. Buro, "Incorporating search algorithms into rts game agents," *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2012.

[19] C. Coxe. Zzzkbot. [Online]. Available: https://github.com/chriscoxe/ZZZKBot [Accessed 16-11-2017]

[20] D. Churchill. Aiide starcraft ai competition official results. [Online]. Available: http://www.cs.mun.ca/ dchurchill/starcraftaicomp/results.shtml [Accessed 16-11-2017]

[21] G. Synnaeve and P. Bessire, "A dataset for starcraft ai and an example of armies clustering," *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2012.

[22] L. Zeming, G. Jonas, l. K. Vasi, and S. Gabriel, "Stardata: A starcraft ai research dataset," *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2017.

[23] G. Synnaeve and P. Bessire, "Special tactics: A bayesian approach to tactical decision-making," in *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, Sept 2012, pp. 409–416.

[24] I. Zelinka and L. Sikora, "Starcraft: Brood war - strategy powered by the soma swarm algorithm," in *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, Aug 2015, pp. 511–516.

[25] A. R. Tavares, H. Azprua, and L. Chaimowicz, "Evolving swarm intelligence for task allocation in a real time strategy game," in *2014 Brazilian Symposium on Computer Games and Digital Entertainment*, Nov 2014, pp. 99–108.

[26] J. Young and N. Hawes, "Learning micro-management skills in rts games by imitating experts," in *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. AAAI, 2014.

[27] G. Synnaeve and P. Bessire, "A bayesian model for rts units control applied to starcraft," in *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*, Aug 2011, pp. 190–196.

[28] B. G. Weber and M. Mateas, "A data mining approach to strategy prediction," in *2009 IEEE Symposium on Computational Intelligence and Games*, Sept 2009, pp. 140–147.

[29] G. Synnaeve and P. Bessire, "A bayesian model for opening prediction in rts games with application to starcraft," in *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*, Aug 2011, pp. 281–288.

[30] H. Park, H.-C. Cho, K. Lee, and K.-J. Kim, "Prediction of early stage opponents strategy for starcraft ai using scouting and machine learning," in *Proceedings of the Workshop at SIGGRAPH Asia*. ACM, 2012, pp. 7–12.

[31] H. C. Cho, K. J. Kim, and S. B. Cho, "Replay-based strategy prediction and build order adaptation for starcraft ai bots," in *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, Aug 2013, pp. 1–7.

[32] K. B. Korb and A. E. Nicholson, *Bayesian Artificial Intelligence, Second Edition*, 2nd ed. CRC Press, Inc., 2010.

[33] M. Stanescu, N. Barriga, and M. Buro, "Hierarchical adversarial search applied to real-time strategy games," *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2014.

[34] B. Weber, M. Mateas, and A. Jhala, "Building human-level ai for real-time strategy games," *Advances in Cognitive Systems: Papers from the 2011 AAAI Fall Symposium*, 2011.

[35] J. Young, F. Smith, C. Atkinson, K. Poyner, and T. Chothia, "Scail: An integrated starcraft ai system," in *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, Sept 2012, pp. 438–445.

[36] M.-J. Kim, K.-J. Kim, S. Kim, and A. K. Dey, "Evaluation of starcraft artificial intelligence competition bots by experienced human players," in *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. ACM, 2016, pp. 1915–1921.

[37] Liquipedia. Protos strategy. [Online]. Available: http://wiki.teamliquid.net/starcraft/Protoss Strategy [Accessed 03-11-2017]

[38] I. Dimitrijevic. Iron bot. [Online]. Available: http://bwem.sourceforge.net/Iron.html [Accessed 16-11-2017]

[39] D. Churchill. Ualbertabot. [Online]. Available: https://github.com/davechurchill/ualbertabot/wiki [Accessed 16-11-2017]

[40] A. Smith. Skynet bot. [Online]. Available: https://github.com/tscmoo/skynet [Accessed 16-11-2017]

[41] BWAPI Development Team. bwapi - an api for interacting with StarCraft: Broodwar (1.16.1). [Online]. Available: https://github.com/bwapi/bwapi [Accessed 04-11-2017]

[42] Microsoft Corporation. Microsoft visual studios 2010. [Online]. Available: https://www.visualstudio.com/vs/older-downloads/ [Accessed 17-04-2018]

[43] MasterOfChaos. Chaoslauncher. [Online]. Available: http://winner.cspsx.de/Starcraft/ [Accessed 17-04-2018]

[44] D. Churchill. Starcraft ai tournament manager. [Online]. Available: https://github.com/davechurchill/StarcraftAITournamentManager [Accessed 17-04-2018]

[45] Oracle. Oracle virtualbox. [Online]. Available: https://www.virtualbox.org [Accessed 17-04-2018]

[46] P. Isaias and T. Issa, *High Level Models and Methedologies for Inofrmation*. Springer International Publishing, 2015.

[47] Team Liquid. Liquipedia. [Online]. Available: http://liquipedia.net [Accessed 22-04-2018]

[48] Liquipedia, "One base speedzeal (vs. zerg)." [Online]. Available: http://liquipedia.net/starcraft/One_Base_Speedzeal_(vs._Zerg) [Accessed 04-05-2018]

[49] ——, "2 gate dt." [Online]. Available: http://liquipedia.net/starcraft/2_Gate_DT [Accessed 04-05-2018]

[50] ——, "2 gateway dark templar (vs. protoss)." [Online]. Available: http://liquipedia.net/starcraft/2_Gateway_Dark_Templar_(vs._Protoss) [Accessed 04-05-2018]

[51] ——, "Help:reading build orders." [Online]. Available: http://liquipedia.net/starcraft/Help:Reading_Build_Orders [Accessed 04-05-2018]

[52] J. Young and N. Hawes, "Evolutionary learning of goal priorities in a real-time strategy game." in *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2012.

[53] D. Churchill. Aiide starcraft competition 2016. [Online]. Available: https://www.cs.mun.ca/ dchurchill/starcraftaicomp/2016/ [Accessed 22-04-2018]

[54] S. W. Ambler. Agile model driven development (amdd): The key to scaling agile software development. [Online]. Available: http://www.agilemodeling.com/essays/amdd.htm [Accessed 04-05-2018]
[55] Liquipedia, "Protoss tech tree." [Online]. Available: http://www.teamliquid.net/forum/brood-war/226892-techtree-pictures [Accessed 04-05-2018]

X. Reflective Addendum

FOR EXAMPLE: During this dissertation many challenges had to be overcome, in subsection those challenges will be explored.

Appendix

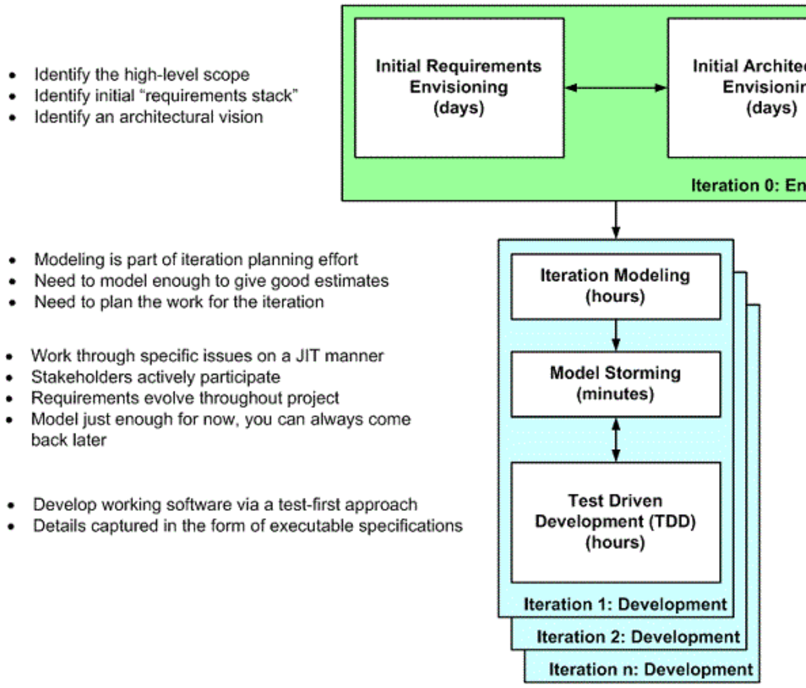

Fig. 10: The Agile Model Driven Development (AMDD) lifecycle [54]

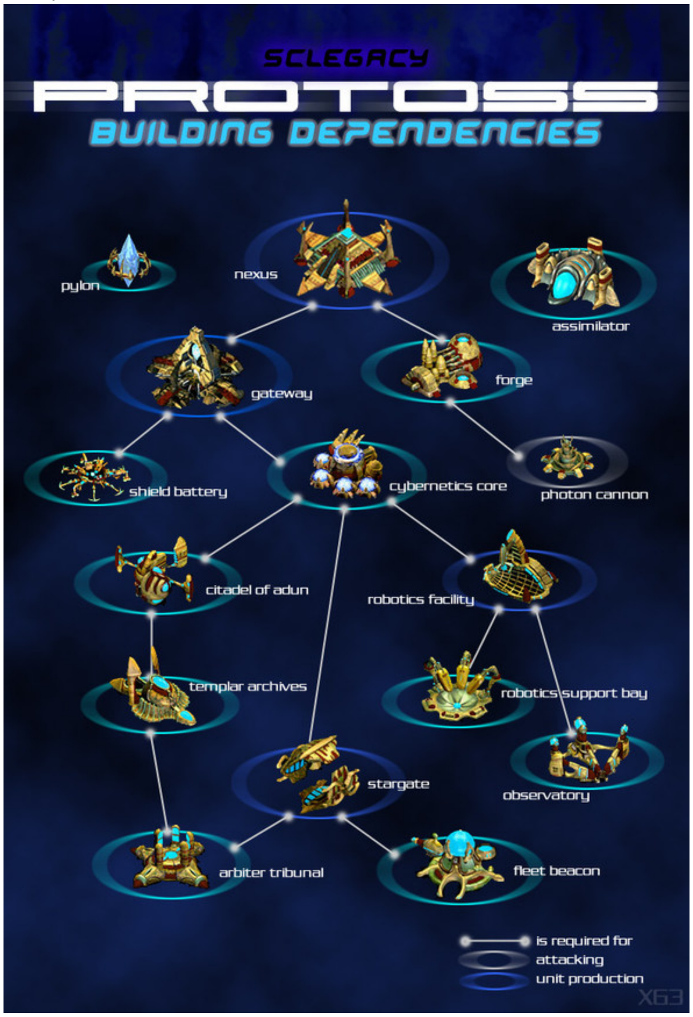

Fig. 11: The Tech tree for Protoss, to be able to produce Datk Templar's from the Gateway, Protoss must fist have the Templar Archives. To produce Dragoons the Cybernetics Core must be present, Zealots can be trained as soon as the Gateway is built [55]

Bot	Games	Win	Loss	Win %	AvgTime	Game Timeout	Crash	Frame Timeout
ZZZKBot	2966	2465	501	83.11	8:00	3	4	0
PurpleWave	2963	2440	523	82.35	13:27	15	25	0
Iron	2965	2417	548	81.52	14:19	117	83	0
cpac	2963	2104	859	71.01	9:45	5	3	0
Microwave	2962	2099	863	70.86	11:34	14	22	0
CherryPi	2966	2049	917	69.08	9:50	7	12	27
McRave	2964	1988	976	67.07	14:35	32	14	0
Arrakhammer	2963	1954	1009	65.95	11:37	11	14	1
Tyr	2966	1955	1011	65.91	13:09	18	13	0
Steamhammer	2964	1901	1063	64.14	10:32	11	4	0
AILien	2966	1729	1237	58.29	13:04	13	216	34
LetaBot	2955	1682	1273	56.92	16:48	119	34	0
Ximp	2962	1605	1357	54.19	18:46	42	205	14
UAlbertaBot	2968	1585	1383	53.4	10:54	59	74	0
Aiur	2965	1496	1469	50.46	13:51	68	53	0
IceBot	2955	1348	1607	45.62	17:16	134	24	0
Skynet	2958	1295	1663	43.78	11:40	29	4	0
KillAll	2965	1276	1689	43.04	10:56	22	4	120
MegaBot	2802	1200	1602	42.83	12:21	52	413	25
Xelnaga	2962	1099	1863	37.1	15:19	121	147	0
Overkill	2958	967	1991	32.69	18:00	128	18	1
Juno	2962	876	2086	29.57	14:07	174	16	0
GarmBot	2961	802	2159	27.09	15:21	40	8	0
Myscbot	2964	769	2195	25.94	13:41	75	4	6
HannesBredberg	2964	630	2334	21.26	14:09	62	8	1
Sling	2963	625	2338	21.09	16:21	147	64	0
ForceBot	2960	532	2428	17.97	15:10	167	9	0
Ziabot	2964	510	2454	17.21	10:08	25	67	0
Total	**41398**	**41398**	**41398**	**N/A**	**13:23**	**855**	**1562**	**229**

TABLE IV: Results of the 2017 AIIDE StarCraft AI Competition Sourced from the official website [20]