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 Bots. A hypothesis is made here that the AI will only be effective in the early stages of the game surviving the rush but 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 not effective and a low average being effective. Whether the AI wins the matches will also be taken into account, a higher average win rate showing the AI to be effective. Upon successful completion of this work, the AI will be submitted to the Artificial Intelligence for Interactive Digital Entertainment StarCraft AI Tournament.**

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 allow students to compete their Bots against other Bots, 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 literature review of the current methods being utilized by the research community in the development of StarCraft Bots and a summary of the hypothesis and other questions considered will complete the section. This is followed by the method used by this paper which includes the tools used, the design and life cycle of the artefact, preliminary results, the metrics used and wrapping up with how the AI will be tested. The results will then be analysed and discussed, with a final analysis of the software, future work and finishing the main body of this work with a conclusion. In Appendix A, there is a reflective addendum, supporting tables and figures are located in Appendix B.

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 Bots 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 this paper, the term Bot which is the standard in the literature will be used rather than AI when referring to AI's that were created specifically for StarCraft. In the StarCraft research community, there are many different methods of Bot 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 Bots 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 Bots tend to struggle with countering this strategy, hence why this type of Bot tends to win.

B. Datasets

A Dataset can be a collection of any data, for a game Bot a dataset can consist of thousands of replays with millions of game frames, and player actions [21]. This information can then be 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 to 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 requires 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 the armies of their opponent. 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 Bot 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 Bot 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 a Bot that controls units individually, they do this by using uncertainty which instead of asking where a unit might be,

it makes a rough estimation 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 a Bot capable of handling all aspects of an RTS [18], [33]–[35]. These Bots tend to take several methods that have been created in other research and combining them to form a new Bot [18]. Another use for the full gameplay Bot is to try and create a "Human-Like" Bot, which can mimic the play-style of an expert human player. Though the current Bots are limited in this as players reported that the Bots 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 Bots.

G. Planning

Planning in StarCraft usually deals with the build order that the Bot 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 Bot 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 the most optimal method of gathering resource and building units [12]. Planning is perhaps the most optimal approach to creating a Bot as there are little real-time calculations to make. Through the use of POSH tools [16], you can iteratively design Bot prototypes and deploy quickly [3].

From looking at the research in the field there are many methods that can be used to create a Bot. The use of replays to train a Bot 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 Bots 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 Bot, 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 Bots

In the StarCraft Bot community there are many Bots that have been created to compete against each other, and in this work, a competition Bot is defined as a Bot that has been entered to the Artificial Intelligence for Interactive Digital Entertainment (AIIDE) conference StarCraft Bot Competition. A yearly competition hosted by David Churchill and sponsored by AIIDE. Examples of the top Bots 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 Bot employs different strategies, for example, the ZZZKBot uses a rush tactic, which 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 7 to 12 minutes of the game [47].

I. Research Questions and Hypothesis

1) Hypothesis:

- **Hypothesis 1** The Bot will survive no more than 14 minutes and Win.
- **Hypothesis 2** The Bot will have a Win rate greater than 0%.
- **Hypothesis 3** The Bot will have a Win rate of 50% or greater.
- **Hypothesis 4** The Bot will successfully counter the Rush tactic surviving past the 10-minute mark.

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

To answer the question that this paper focuses on, this research will be performing an experiment to gather the empirical data required then performing a positivist analysis on that data, interpreting it through reason and logic, then forming a discussion to defend or reject the hypothesis presented.

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 Bots. 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 Bots to communicate with the game. BWAPI only give limited information to the Bot, 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) is kept [16], this means that the Bot is just as 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 Bot to have the ability to scout effectively. This limited information prevents custom Bots from cheating and ensures a fair game, though for the developers of the Bot this could be considered an advantage for the development stage of the Bot as there is no need to be concerned with accidentally allowing their Bot to access illegal information. This does, however, provide a challenge in design as the Bot is dealing with imperfect information, it must be designed in such a way that it almost replicates 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 a Bot as once the behaviours and senses are implemented new Bots can be created quickly.

- **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 Bot, as well as any other functionality the Bot 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 Bot. The launcher also allows StarCraft to be run in windowed mode alongside a BWAPI injector that allows the Bot 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, 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 (VM) will be utilized [44]. For the purposes of this experiment the STM will be used to test the Bot against the competition Bots, once the tournament is over the STM will compile a results table as seen in table 1, in an 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 excerpt of R can be seen in fig 3.

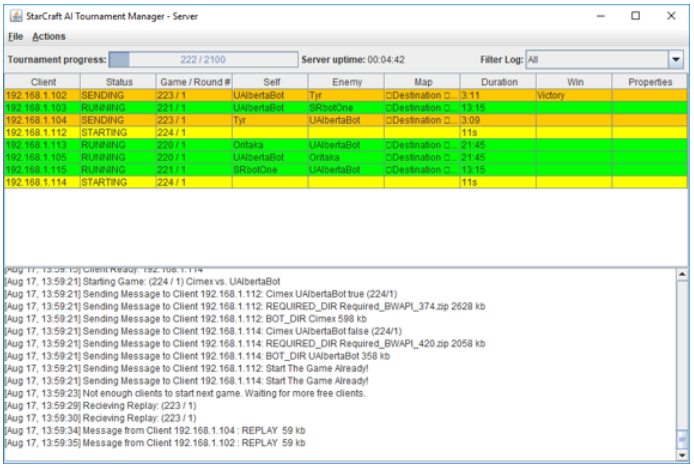

Fig. 2: StarCraft Tournament Manager Server Running.

```
[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 challenge, once the research question was settled on, a large design task required was to obtain several expert strategies that would be compatible with the Bot. 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 [37]. 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.

Another 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 a Bot with pre-built build orders taken from Liquipedia [37], a website dedicated to StarCraft, on 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 excerpt can be seen in Figure 8.

For the creation of the artefact, there were many options for life cycles that could have been adopted. Chief among these were the incremental model and Agile, both supported 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 impractical for this artefact as the strategies and variables in the code need to be constantly refactored and tweaked, an action that the incremental model does not allow.

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 Bot, 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 Appendix B under figure 9. Another reason for AMDD is that the constant small changes to the code would dramatically affect the functionality of the Bot, 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 get 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 Bot to be created, it could either be done in C++ pure,

Java or C# using POSH-tools. POSH-tools in C# were chosen as there was an open source Bot written in this environment available to be iterated upon [3].

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 issue that came up after attempting to compile BWAPI, there were missing Dynamic-link libraries (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, which would inject BWAPI into the game to allow the Bot to work correctly within the game. 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, the 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. In this work Protoss will be chosen, this choice was made as Protoss provides a middle ground to build upon where rushes are difficult to achieve as well as not being too complex to achieve a sound strategy.

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. This functionality would make Probes 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 refactoring needed. An example of this would be the positioning of buildings, 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 lose a builder, as the builder "Morphs" into the structure so each time something is built the Bot 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 whereas 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 Bot 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 Bot 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 Bot to find the choke point and set it as a base location for building its structures, this also opened up an issue with ensuring the Bot can swap building locations. Within BWAPI the Bot 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 Bot 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 Bot would now take advantage of building at both the starting area as well as the natural expansion and choke point. Along with this the Bot 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 Bot to find the enemy base and mark its location, which allowed the Bot to know where to go with its forces.

5) Iteration 4 : Polish: At the end of this sprint, the Bot 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 refactor 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. Due to the nature of the "Action", "Sense" system there was 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 the 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. 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 Bot 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 referred to as a mirror match.

Each strategy focuses on the same goal, producing a unit called Dark Templars (DT's) 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 Appendix B figure 10, which shows the tech tree for the Protoss. Though each strategy gets to this goal in different ways:

- For Protoss VS Zerg the Bot 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 Bot will use a strategy known as "2 Gate Dark Templar (vz. Terran)" [49], the Bot must forego the Zealots and head straight into DT's, that means that the Protoss are prone to early rushes by the enemy, but to defend against this one or two ranged units called Dragoons are trained to defend. Once the Protoss has two DT's they are sent to the enemies natural expansion to harass them and lock them in their base until 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 Bot 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 is defended, and the natural expansion taken the focus shifts to mass producing units, and sending them in to attack the enemy, it is down to how the enemy plays which dictates which unit to produce.

Each strategy was implemented separately and as each action taken by the Bot 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 suit the needs of the player, or in this case the Bot. For example, to fight against 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 remained 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 Bot developed for this research. For the preliminary test, 9 Bots were chosen from the AIIDE 2016 competition, three from the top, three from the middle and three from 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. These results allowed the further development of the metrics within this research, this is explained in the following section.

D. Metrics

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

- **Time Survived** (Average of 13 minutes or above)
- **Endgame Condition** (Whether the Bot 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 Bots vary substantially from 17.21% to a high 83.11% as can be seen in Appendix B Table 7.

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 Bots within the literature. As can be seen in fig 6, the mean win rate for the Bot was 50% and within the literature, the win rate ranged from 54-91% [3], [14], [15], [52]. The literature Bots were faced against both competition Bots and in-built Bots which came with StarCraft.

From the preliminary results obtained from table 1, the mean game time was 14:20, with the longest mean being

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

19:24 and the shortest mean 6:05. No papers that were reviewed for this research focused on average game length, they seemed only concerned with win % or Bot efficiency in a focused area of functionality. For the purposes of this paper, game length will be measured and analysed to measure countering the rush tactic. As defined on the Liquipidiea website ”Mid-game is the period in a match where the strategy of the early game can come together” and ”Mid-game typically starts between 7 and 12 minutes” [47]. This supports the metric that if the Bot survives past the 10-minute mark then it will have successfully countered the rush tactic.

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

E. Testing

The Bot was faced against the open source competition Bots, 400 matchups were simulated, 80 of which the Bot presented in this paper took part in. The environment chosen to test the Bots was the same as the competitions run by D. Churchill, this method of testing was chosen as it provided a ready-made, well-validated method for testing the Bot.

During testing, there was a total of 10 Bots including the Bot that was developed for this research, for the rest of this paper will be referred to as POSH-bot. To choose these Bots, 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 as fair as possible each Bot in each tier will consist of each of the races, where that is not possible, a substitute has been made [53]. AIIDE 2016 was chosen as all the Bots required for testing were available for download. The final Bots that POSH-bot will be facing consist of 4 Terran, 3 Protoss and 2 Zerg, a Zerg was substituted for a Terran as the original Zerg Bot chosen continued to crash throughout the preliminary testing. It was also judged that it would not make a significant difference to the results as one of the Zerg Bots has such a high win rate.

The 10 Bots will be playing on 3 1v1 maps, traditionally there is a selection of 2v2 maps as well. For the purposes of this research, only the 1v1 maps were used as it was judged that on larger maps POSH-bot would have timed out more often, thus resulting in more skewed results. The Bots played for 9 rounds to total 400 matchups with one winner and one looser, the score and game length was also recorded. The three maps used were,(2)Benzene, (2)Destination, (2)HeartbreakRidge, all are available from the D. Churchill’s GitHub page for the tournament manager [44].

To set up the testing environment a Virtual Machine (VM) was required to run several instances of StarCraft, and the StarCraft Tournament Manager (STM). Once the virtual machine was installed a 64-bit operating system had to be installed on the VM, a 64-bit version was required as the STM needed a minimum of 2 Central Processing Unit (CPU) cores to run alongside Starcraft. Then once the VM is set up correctly, the server was launched alongside two clients, one on the host machine and one on the VM. Once the three programs have been launched the user chooses how many rounds the tournament should last, and the rest of the process is automated, the STM will select the correct Bots, send them to the clients and launch the game. After each game the process would begin anew until the tournament was complete, this process took approximately 16 hours.

Race	# Matches	Wins	Win Rate %	AVG POSH Score	Std Dev	Avg Game Length	AVG Oponent Score	Std Dev	AVG Score Diff
Zerg	17	1	5.88%	29370	17778	16:46	33518	19071	-14.12%
Terran	36	6	16.67%	24070	36436	19:41	26276	19881	-9.17%
Protoss	27	1	3.70%	7346	4652	11:48	23168	15537	-215.37%
Total	80	8	10%	19552	27222	16:24	26766	18617	-36.90%

TABLE II: Results from the 80 matches that the POSH-bot described in this paper took part in against the competition Bots on three 1v1 competition maps.

Bot	# Matches	Wins	Win Rate %	AVG Score	Std Dev	Avg Game Length
ZZZKBot	80	73	91.25%	7188	6672	06:25
Iron	80	69	86.25%	36388	35563	13:13
Xelnaga	80	48	60.00%	39295	24793	16:11
POSH-bot	80	8	10.00%	19552	27222	16:24

TABLE III: Results from the top three Bots of each race, ZZZKBot(Zerg), Iron(Terran), Xelnaga(Protoss), compared to POSH-bot

Bot	# Matches	Wins	Win Rate %	AVG Score	Std Dev	AVG Score Diff	Avg Overall Game Legnth	Avg Game	AVG Game Legnth Diff
ZZZKBot	9	9	100.00%	24095	6419	7.24%	06:25	15:35	145.60%
POSH-bot	9	0	0.00%	22350	8795	-7.81%	16:24	15:35	-5.80%

TABLE IV: Closer inspection of the results comparing matches between ZZZKBot and POSH-bot

IV. Results & Analysis

After the implementation of the three chosen strategies, and final testing was done within the tournament manager, the results contained in table 7 located in Appendix B were produced. An example of the full list of results can be found in Appendix B Table 8. From table 7 other tables were produced, 1 and 6 were produced automatically by the STM, while tables 2, 3 and 4 were manually created.

Investigating POSH-bot facing each race individually, from the results in table 2, under the average game length, the shortest games were against the Protoss with the average game lasting 11:48s and a 3.7% win rate. POSH-bot also proved less efficient in scores with a -215.37% difference. This shows that the strategy used by POSH-bot was flawed, as the average game time was 11:48s this indicates that it was the early-mid game, meaning that the early game strategy for the enemy was coming into fruition and POSH-bot failed to counter it. This resulted in the low 3.7% win rate and substantial difference in score. Though the games ended earlier, the average game length was above 10 minutes, which would indicate that any early rush attempts made by the opponent were countered.

When faced against the Zerg POSH-bot had a similar win rate as to when it faced the Protoss of 5.88%, but did considerably better with its scores, with a lesser difference -14.12%, however, the game length on average lasted for 16:46s. Though POSH-bot still failed to achieve a high win rate against the Zerg it did, however, make the matches last longer, past the 15-minute mark. This indicates that the Zerg rush was completely countered, as at this point any early game strategy would have finished and that game was well into mid game. Which also suggests that the strategy employed against the Zerg was successful up until mid game. Another indication of POSH-bot fairing better against the Zerg is the score difference.

The win rate when against Terran was a higher 16.67%, all 6 of these victories were against the same Bot which had a win rate of 18.52%. The score difference was lower with -9.17% and the longest average game length of 19:41s. This suggests that the strategy employed to counter the Terran faired better than the others, likely due to the game lasting past the 15-minute mark allowing the strategy to be fulfilled. The score also supports this as there is a smaller difference meaning that the POSH-bot managed to counter any rush attempt and produce units to fight.

When inspecting the differences between POSH-bot and the three top Bots for each race in table 3, POSH-bots average game length was close to the same as its counterpart Xelnaga with both having 16:24s and 16:11s respectively. This indicated that POSH-bots strategy succeeded in countering any early rush attempt. Their win rates were considerably different, with POSH-bot having 10% and Xelnaga claiming 60%, and coming fourth overall in the testing. Iron, a Terran race has a win rate of 86.25% and an average game length of 13:13s, both of which were better than the results POSH-bot obtained. ZZZKBot who came out on top during testing managed to secure a 91.25% win rate, and an average game length of 6:25s. ZZZKBot likely achieved such a low average on game length due to its rush tactic, Iron also follows suit on this though from the replays, it is evident that Iron takes longer than ZZZKBot to build its forces.

Upon closer inspection of the results table 4 was created which contains the results obtained from the matches between ZZZKBot and POSH-bot. ZZZKBot had a win rate of 100% the average game length when matched against POSH-bot increased by 146.6%, both Bots scores were close with a 7% difference. Though with Iron the game length was decreased by 37.7% and by 11.98% by Xelnaga suggesting that POSH-bots early game rush counter was successful against any Zerg attempt, but was hard countered by a Terran vulture and factory rush. Xelnaga also being stronger earlier than POSH-bot attributed to the faster games. Through performing a left-handed t-test on all the results for game length for all matches

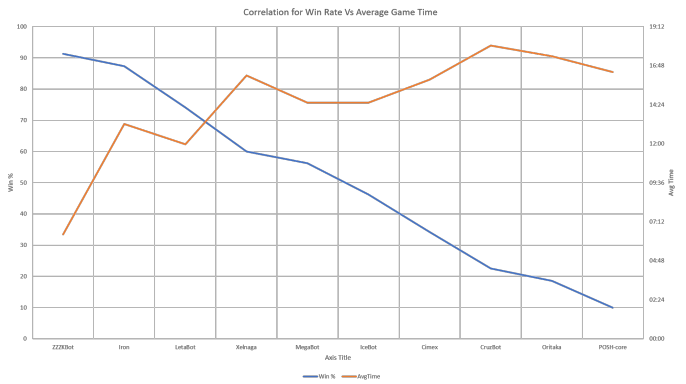

Fig. 7: A line graph showing both the win rate and game length for all the Bots, starting with the highest win rate on the left, based on the results obtained from table 8 in Appendix B.

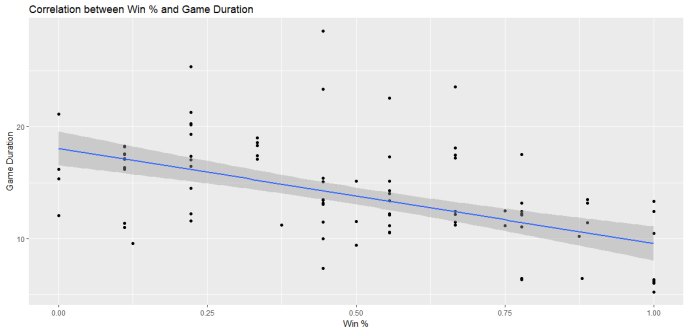

Fig. 8: A scatter plot showing a negative correlation between the win rate and game length, based on the results obtained from the preliminary experiment. Each dot represents an average win rate and average game length for each bot during each round.

involving POSH-bot figure 5 was obtained.

```
> t.test(POSHTimes, mu=13, alternative="greater", conf.level=0.95)

        One Sample t-test

data:  POSHTimes
t = 1.9127, df = 79, p-value = 0.02971
alternative hypothesis: true mean is greater than 13
95 percent confidence interval:
 13.41704      Inf
sample estimates:
mean of x
 16.21225
```

Fig. 5: A left sided t-test performed within R Studio for all game lengths on all matches involving POSH-bot.

Bot	Matches Won	Avg Game Length	Avg POSH Score	Avg Oponent Score
POSH-bot	8	33:50	52978	23790

Fig. 6: An overview of POSH-bots victories over its opponents.

As seen in figure 6 the p-value is less than 5% which indicates that the null hypothesis can be rejected, in this case, it was to filter out the matches against Iron, by setting the μ to 13 for the time that a game transits into mid game and in general a rush can no longer be considered a rush.

During the matches where POSH-bot won as shown in figure 6, on average the games lasted 33:50s, with an average score much higher than that of the opponent. These were most likely timeouts, where the games exceed their maximum frame count quota as stated within the STM readme file [44], from the timeout times the maximum time a Bot can spend in a game is around 59:31s. Though victory is awarded to the Bot with the highest score.

After some investigation, it was discovered that there may be a correlation between game length and win rate as shown in figures 7 and 8. Figure 8 shows that there may be a negative correlation between these two values.

V. Discussion

It is clear from these results that the Bot was not by the metrics defined in this paper, effective in its use of the implemented strategies, thus confirming the null hypothesis of hypothesis 3 that the Bot cannot achieve above a 50% win ratio in its current state. This was not entirely unexpected as during early testing there was evidence that the strategies provided from Liquipidia were not going to be very effective, as they relied on the Bot scouting and building immediately as the game started. As the system could not compute its actions instantaneously the Bot could not execute the strategies in time before the enemy Bot had built a larger force. The framework was too slow and needed altering, as this was not the focus of this research it would have been an improper use of resources.

Though the Bot did achieve a win rate greater than 0% and therefore supporting hypothesis 2. Upon further inspection of the replays of the games that POSH-bot won against Oritaka a Terran Bot, it became clear the victories accounted for the ability to counter a Terran Marine rush. Due to its similarity to the Zerg rush (sending in weak cheap units) it was easily countered by POSH-bot and it became apparent that this was the main focus of Oritaka and once POSH-bot acquired some DT's it easily overpowered Oritaka and destroyed its main base. Though most of the games ended in a timeout, victory was awarded to POSH-bot due to the substantial score difference.

As evidenced by the consistent increase in times for the ZZZKbot's games and POSH-bots ability to survive past the 10-minute mark, as well as the t-test, hypothesis 4 can be supported.

As shown in figure 6 the average game length for POSH-bots winning matches is much greater than 14 minutes, thus rejecting hypothesis 1 and supporting the null. This was likely due to the lack of scouting abilities within POSH-bot, along with a present bug in the software where the units would not attack correctly. This caused the units to not move after reaching the centre of the enemies base thus timing out the game.

A. Analysis of final Software

Though two of the hypothesis can be supported, the rearranging two were rejected, this is due to issues that could be improved on in the software. This section will identify those issue and present potential fixes for future work.

1) Framework: Once the system was complete and all the behaviours needed were present, creating plans was a relatively quick process, though it has one major disadvantage. The Bot can only execute one action at a time, for example, when it 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 Bot as it must be created in such a way that it can be effective while being limited to only one action at a time. A potential workaround for this would be to use parallel plans, this would involve having multiple plans working in unison on separate actions to drive the Bot.

Another issue presented by this software is that it is CPU intensive and at times can slow down the game simulations considerably, this makes testing time increase, 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. Ensuring that ”while loops” and ”foreach loops” are kept limited in their use and only used when absolutely necessary, would help to alleviate this issue.

When compiling the bot everything must be run in administrator mode, a minor inconvenience when developing the bot as the bot would launch in this mode when Visual Studios was run with administrator privileges. The software required this as without these privileges the Bot was unable to communicate with BWAPI ad function. The major inconvenience of this requirement came when attempting to launch the Bot using a custom batch file, as the batch file had to launch in admin mode, to achieve this the windows user setting had to have security disabled, so admin privileges were given automatically and the operating system would not request permission. If this was not done then the tournament manager would be unable to launch the Bot and all testing involving the Bot would have to be done manually.

The results for the mirror matches were not unexpected as during development these matches proved the most difficult to implement a strategy for, as both factions can create units at the same rate. As the framework had issues communicating with the game this slowed down the speed of the bot, giving an advantage to any other Protoss the Bot faced.

2) Tournament Manager and Virtual Machine: When using the STM an issue that arose was that only one instance of StarCraft could be run at a time, which required the use of a VM. The issue with this was that if the VM was not set up exactly as needed, the client would not connect to the server or other clients, which caused the whole system to not work. This was mainly an issue with the operating system and networking set-up in the VM, but the issue could have been avoided by allowing the STM to launch multiple instances of StarCraft on one machine. StarCraft and the client only require one CPU core each to run plus another for the server, most modern computers have an octa-core CPU, which would allow for 5 core to be used by the clients and StarCraft.

The STM when running can only allow clients with different Internet Protocols (IP’s) to join each other in-game, otherwise, the game will not recognise that there is a game to join. If the VM was not set up correctly it would share the same IP address as the host machine, which to a user who is unfamiliar with VM software could prevent the use of the STM entirely. A fix that enabled the software to work in this instance was to reinstall the VM with admin privileges which seems to install extra networking adaptor options allowing for the use of a separate IP address to the host machine, which in turn allows the clients to connect.

When running the STM all firewalls have to be turned off and the registry altered to allow seamless networking between the clients and server, this posses a very high security risk to the users system and could potentially be solved if the STM has a local version which runs off a single host machine.

3) Plans and Behaviours: As the Bot works through the plan, it works from top to bottom, working through the drives and stepping into their functions as needed if the sense is satisfied. Each of these drives can be delayed and skipped to ensure that all drives are triggered, this should work in theory. In reality, if the delay is more than a few microseconds the change in speed for the behaviours being triggered is noticeable, as the bot must trigger a command several times as BWAPI does not always work. If there is no delay put on the drives, the system will crash as it is running faster than the other two systems, POSH and BWAPI, this causes NULL exceptions to be thrown, so there must be a time delay present. The designer can, however, plan the delays in such a way that the plan not only works its way down the drive list but also works back up. This helps with any issues pertaining to the plan getting stuck inside a particular drive but does not solve the issue, for this, it is best to ensure that the senses are set up correctly so the drive is not triggered at all unless it can be resolved. This means that when creating a plan all drives and senses must be precise, as one incorrect number or misplaced drive can break the Bot entirely, so the plan must be checked and tested several times after each change. An example of the code output from the plans can be seen in Appendix B figure 11.

Another issue that arose several times was the placement of buildings, as stated earlier, BWAPI does sometimes not accept a command from the Bot so the command must be sent again. Which caused issues with building placement as the Bot would register that the building was placed but in the game, it was not, this meant that the placement and build functions had to be triggered multiple times.

The Bot currently uses a Fibonacci spiral to locate a suitable build location which should always return a viable option, which is why the Bot currently builds its structures in the immediate vicinity of the main base. This needs to be improved upon as the structures currently interfere with resource gathering, a simple solution would be to implement another strategy that takes advantage of the choke points, this would move the build location to another area. Also as the POSH-bot is Protoss the buildings can only be placed near a Pylon, so another option is to change the build location to each Pylon, though this can cause issues if a Pylon is destroyed, so a list of Pylons would have to be maintained. This solution would force the Bot to build around any Pylon so the only structure that would need changing in its placement would be

the Pylon, rather than changing the entire placement system.

Currently, there is a large delay between when the game begins and when the Bot can trigger certain actions, for instance, scouting, a pivotal part of an opening strategy. This delays the Bot from being able to perform certain functions like building at its natural expansion early, or harassing the enemies base immediately as the game begins. This has caused issues when designing the plan, as the strategies required immediate scouting but the final design had to work around that by triggering the scouting later in the strategy.

VI. 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 game length. 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 Bots to facilitate fast strategies, and though the Zerg implements rush tactics which is inherently fast, it would be interesting to see how and Bot 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 Bots 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 Bot 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 Bot to build up their base, scout and manage their army at the same time. This 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 a Bot. The Bot could have multiple plans, each focusing on a separate component of the game, these could be combat, base building, resource management or defence. Creating a Bot 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 an order of priority or a threat calculation, so the defence can take over from the combat plan if the Bot is losing its base.

A potential improvement to this work could be to investigate the scores of the Bot as in many cases the POSH-bot may not have won but may have performed well as reflected by its scores. As this was not the focus for effectiveness in this paper it could provide a more granular method of evaluating its performance.

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

VII. Conclusion

In games, there are many ways to create AI and many games to create AI in, for this work StarCraft was chosen as the platform in which to create such AI. Creating an effective AI or Bot, for StarCraft can be a challenging task, to which much research has been done, this paper reviewed the literature in this area to analyse the methods used for such a task. After this analysis, several questions and hypothesis were presented and a method was created appropriate to solving these questions and supporting the hypothesis. In this work a bot for StarCraft was presented which used expert strategies sourced from the community surrounding the game, that was tested against several other bots which have participated in the annual AIIDE StarCraft AI competition. The tools that were used were presented as well as the life cycle involved with the Bots creation, once complete the Bot was then tested in a competitive environment against Bots that were entered into the 2016 AIIDE StarCraft AI competition. The results were then recorded presented and analysed, after which a discussion was held on the implications of the results, followed by an analysis of the final software used and potential future works. Though the Bot did not fair as well as was hoped, it did prove an interesting challenge, as the design of the Bot was the greatest challenge, as the limitations of the frameworks speed and the way the plans operated had to be circumvented to produce a viable artefact. In this case, the Bot presented was not effective in its use of expert strategies, that being said, the Bot did achieve a greater than 0% win rate, plus the initial rushes were countered, supporting two of the hypothesis presented, showing potential for improvement. For any future work, it would be recommended that strategies are kept as simple as possible while maintaining an effective strategy or employ the use of parallel plans to make the Bot more efficient in carrying out that strategy.

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]

ACKNOWLEDGEMENTS

I would like to thank Swen Gaudl for his help in the creation of this dissertation, without his guidance and patience this work would not have been produced.

I would also like to thank my peers within the BSc for their continuous support throughout this course.

Also, a thank you to Dr. Ed Powley and Dr. Michael Scott for their unyielding dedication to this course ensuring that it continues to be developed to provide the highest quality education.

APPENDIX A
REFLECTIVE ADDENDUM

Whilst I am proud of this research there are many lesson that I should have learnt earlier and several key issues that need to be addressed. If there is one thing though that I have learnt through writing this paper and creating the Bot is that it requires patience and determination. When I began this journey I knew nothing about POSH plans, StarCraft Bots, BWAPI or even any C#, after taking this project on I realised that it was a bigger deal than I first anticipated, but support from my peers and tutors helped me stay persistent and stay patience as I progressed through this project. This section is to critically reflect on the journey I took in this projects creation, looking at what went wrong and what went well. This is broken down into section

A. Priorities

During the first semester my primary focus was the literature review and attempting to make a start on the other projects presented. This lead to me neglecting learning the software involved with this project, which caused major issues in the following semester.

B. Learning New Systems

Producing this text and the accompanying artefact allowed me to learn outside of a games engine, and helped me understand further

C. Research Question

D. Targets for the Next Project

APPENDIX B
SUPPORTING FIGURES AND TABLES

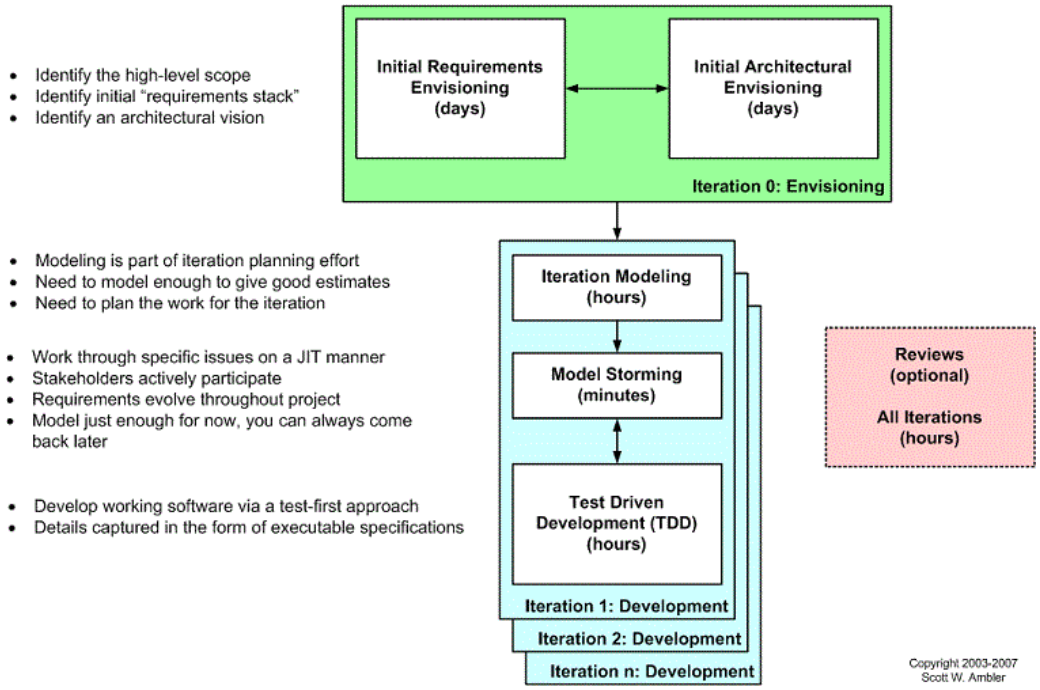

Fig. 9: The Agile Model Driven Development (AMDD) lifecycle [54]

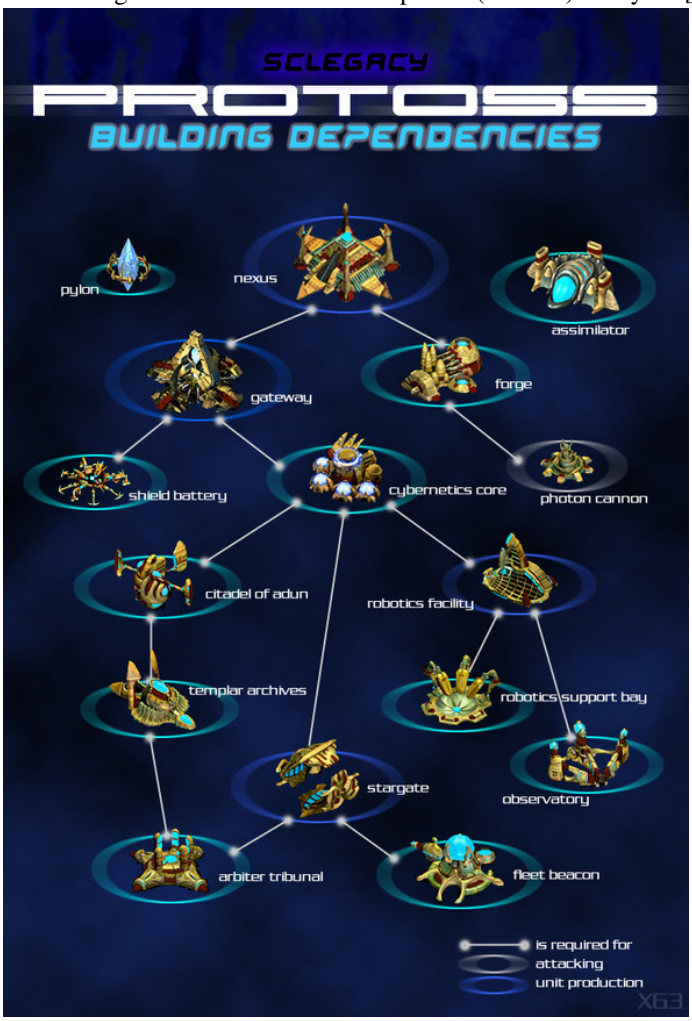

Fig. 10: The Tech tree for Protoss, to be able to produce Dark Templar's from the Gateway, Protoss must first have the Templar Archives. To produce Dragoons the Cybernetics Core must be present, Zealots can be trained as soon as the Gateway is built [55]

Bot	Games	Win	Loss	Win %	AvgTime	Game Timeout	Crash	Frame Timeout
ZZZKBot	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 V: The HTML results table produced by the StarCraft Tournament Manager [44]. Blue represents Terran, Purple represents Zerg, and Yellow represents Protoss

```
(SDC EighteenNexusOpening (goal ((GameRunning 0.0 =)))
    (drives
        (       (defend (trigger ((HasUnits 1.0 ==)(EnemyDetected 1.0 ==)(BaseUnderAttack 1.0 ==))) defendLocation(seconds 0.2)))
        (       (scout (trigger ((ArchivesCount 1.0 >=) (NaturalFound 0.0 ==))) exploreLocal (seconds 0.001)))
        (       (harvest (trigger ((IdleProbes))) probeHarvesting(seconds 0.0003)))
        (       (retreat (trigger ((ForceIsLosing 1.0 =))) flee(seconds 1.0)))
        (       (researchObserver (trigger ((ObservatoryCount 1.0 >=)(HaveOberverUpgrade 0.0 ==))) ObserverResearch(seconds 0.0007)))

        (       (attackZerg (trigger ((DarkTemplarCount 4.0 >=)(ZealotCount 5.0 >=)(ObservatoryCount 0.0 ==)(EnemyRace 0.0 ==))) attackEnemy(seconds 0.0002)))
        (       (attackZerg (trigger ((DarkTemplarCount 10.0 >=)(ObservatoryCount 0.0 ==)(EnemyRace 0.0 ==))) attackEnemy(seconds 0.0002)))
        (       (attackZerg (trigger ((DarkTemplarCount 10.0 >=)(ObservatoryCount 1.0 >=) (ObserverCount 1.0 >=)(EnemyRace 0.0 ==))) attackEnemy(seconds 0.0002)))
        (       (researchZerg (trigger ((CitadelCount 1.0 >=)(NeedResearch 1.0 ==)(EnemyRace 0.0 ==))) OneBaseSpeedZealotResearch(seconds 0.0007)))
        (       (createForceZerg (trigger ((GatewayCount 1.0 >=) (AvailableSupply 3.0 >)(EnemyRace 0.0 ==))) OneBaseSpeedZealotForce(seconds 0.0010)))
        (       (buildZerg (trigger ((NeedBuilding 1.0 =)(EnemyRace 0.0 ==))) OneBaseSpeedZealotBuild(seconds 0.0001)))

        (       (attackTerran (trigger ((DarkTemplarCount 4.0 >=)(EnemyRace 1.0 ==))) attackEnemy(seconds 0.001)))
        (       (researchTerran (trigger ((ForgeCount 1.0 >=)(NeedResearch 1.0 ==)(EnemyRace 1.0 ==))) TwoGateDarkTemplarResearch(seconds 0.0015)))
        (       (createForceTerran (trigger ((GatewayCount 1.0 >=)(AvailableSupply 3.0 >)(EnemyRace 1.0 ==))) TwoGateDarkTemplarForce(seconds 0.002)))
        (       (buildTerran (trigger ((NeedBuilding 1.0 ==)(EnemyRace 1.0 ==))) TwoGateDarkTemplarBuild(seconds 0.0001)))

        (       (scoutProtoss (trigger ((NaturalFound 0.0 ==) (EnemyRace 2.0 ==) (ForgeCount 1.0 >=))) exploreLocal (seconds 0.001)))
        (       (attackProtoss (trigger ((DarkTemplarCount 4.0 >=)(EnemyRace 2.0 ==))) attackEnemy(seconds 0.001)))
        (       (researchProtoss (trigger ((ArchivesCount 1.0 >=) (IsForgeResearching 0.0 ==) (NeedResearch 1.0 ==)(EnemyRace 2.0 ==))) FastExpandResearch(seconds 0.0015)))
        (       (createForceProtoss (trigger ((GatewayCount 1.0 >=)(AvailableSupply 3.0 >)(EnemyRace 2.0 ==))) FastExpandForce(seconds 0.002)))
        (       (buildProtoss (trigger ((NeedBuilding 1.0 ==)(EnemyRace 2.0 ==))) FastExpandBuild(seconds 0.0001)))

        (       (wait (trigger ((Success))) idle))
    )
)
```

Fig. 11: Example of the code that a POSH plan outputs, this code represents the drives located to the right hand side of each line, before the delay which is counted in seconds. The senses are to the left of the drives, an example of which would be DarkTemplarCount, meaning that the number of Dark Templars must be above that number to trigger the drive.

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

Round/Game	Winner	Loser	Crash	Timeout	Map	Duration	W Score	L Score	(W-L)/Max	W 55	W 1000	W 10000	L 55	L 1000	L 10000	Win Addr	Lose Addr	Start	Finish
0 / 00000	CruzBot	POSH-core			Benzene	00:05:47	8932	2515	0.71835	0	0	0	0	0	0	/192.168.56.1	/192.168.1.214	20180503_015733	20180503_015908
0 / 00001	MegaBot	POSH-core			Benzene	00:08:12	19050	7650	0.59839	11	0	0	0	0	0	/192.168.1.214	/192.168.56.1	20180503_015919	20180503_020207
0 / 00002	Xelnaga	POSH-core			Benzene	00:13:36	31986	10157	0.68243	0	0	0	0	0	0	/192.168.1.214	/192.168.56.1	20180503_020400	20180503_020557
0 / 00003	IceBot	POSH-core			Benzene	00:10:29	22955	7725	0.66344	0	0	0	0	0	0	/192.168.1.214	/192.168.56.1	20180503_020607	20180503_020753
0 / 00004	Iron	POSH-core			Benzene	00:09:15	18219	4549	0.75027	1	0	0	0	0	0	/192.168.1.214	/192.168.56.1	20180503_020804	20180503_020931
0 / 00005	LetaBot	POSH-core			Benzene	00:09:29	22353	7395	0.66914	0	0	0	0	0	0	/192.168.1.214	/192.168.56.1	20180503_020942	20180503_021110
0 / 00006	POSH-core	Oritaka			Benzene	00:59:31	66936	16095	0.75954	0	0	0	0	0	0	/192.168.56.1	/192.168.1.214	20180503_021121	20180503_021909
0 / 00007	POSH-core	Cimex			Benzene	00:22:57	54879	34845	0.36505	0	0	0	0	0	0	/192.168.56.1	/192.168.1.214	20180503_024729	20180503_025059
0 / 00008	ZZZKBot	POSH-core			Benzene	00:11:41	20801	14902	0.28358	0	0	0	0	0	0	/192.168.1.214	/192.168.56.1	20180503_025111	20180503_025246
0 / 00009	CruzBot	MegaBot		MegaBot	Benzene	00:17:13	32647	53346	-0.38801	0	0	0	320	0	0	/192.168.56.1	/192.168.1.214	20180503_025255	20180503_030118
0 / 00010	Xelnaga	CruzBot			Benzene	00:59:31	70337	10933	0.84455	0	0	0	0	0	0	/192.168.1.214	/192.168.56.1	20180503_030129	20180503_030846
0 / 00011	IceBot	CruzBot			Benzene	00:14:52	57936	19890	0.65668	1	0	0	0	0	0	/192.168.1.214	/192.168.56.1	20180503_030856	20180503_031122
0 / 00012	Iron	CruzBot			Benzene	00:10:05	25928	7200	0.72228	0	0	0	0	0	0	/192.168.1.214	/192.168.56.1	20180503_031132	20180503_031302
0 / 00013	LetaBot	CruzBot			Benzene	00:19:09	81036	28429	0.64917	0	0	0	0	0	0	/192.168.1.214	/192.168.56.1	20180503_031313	20180503_031627
0 / 00014	CruzBot	Oritaka			Benzene	00:06:57	11765	3960	0.66335	0	0	0	0	0	0	/192.168.56.1	/192.168.1.214	20180503_031638	20180503_031755
0 / 00015	Cimex	CruzBot			Benzene	00:33:37	155011	85071	0.45119	0	0	0	0	0	0	/192.168.1.214	/192.168.56.1	20180503_031806	20180503_032518
0 / 00016	ZZZKBot	CruzBot			Benzene	00:04:14	5288	2377	0.55039	0	0	0	0	0	0	/192.168.1.214	/192.168.56.1	20180503_032529	20180503_032617
0 / 00017	MegaBot	Xelnaga			Benzene	00:59:31	17840	15904	0.10851	0	0	0	0	0	0	/192.168.56.1	/192.168.1.214	20180503_032626	20180503_033117
0 / 00018	MegaBot	IceBot			Benzene	00:20:46	74001	48060	0.35054	0	0	0	2	0	0	/192.168.1.214	/192.168.56.1	20180503_033127	20180503_033406
0 / 00019	Iron	MegaBot			Benzene	00:12:27	38356	14959	0.60998	0	0	0	0	0	0	/192.168.1.214	/192.168.56.1	20180503_033415	20180503_033613
0 / 00020	LetaBot	MegaBot			Benzene	00:19:34	80900	29063	0.64075	0	0	0	0	0	0	/192.168.1.214	/192.168.56.1	20180503_033624	20180503_034010
0 / 00021	MegaBot	Oritaka			Benzene	00:07:55	17641	4819	0.72679	1	0	0	0	0	0	/192.168.56.1	/192.168.1.214	20180503_034019	20180503_034145
0 / 00022	MegaBot	Cimex			Benzene	00:56:33	109599	61420	0.43959	1	0	0	0	0	0	/192.168.1.214	/192.168.56.1	20180503_034153	20180503_035440
0 / 00023	ZZZKBot	MegaBot			Benzene	00:04:09	5215	2510	0.5186	0	0	0	0	0	0	/192.168.1.214	/192.168.56.1	20180503_035451	20180503_035540
0 / 00024	Xelnaga	IceBot			Benzene	00:16:15	61760	37226	0.39724	0	0	0	2	0	0	/192.168.56.1	/192.168.1.214	20180503_035549	20180503_035803
0 / 00025	Iron	Xelnaga			Benzene	00:13:58	46692	19624	0.5797	0	0	0	0	0	0	/192.168.1.214	/192.168.56.1	20180503_035812	20180503_040020
0 / 00026	Xelnaga	LetaBot			Benzene	00:15:33	51424	31855	0.38053	0	0	0	0	0	0	/192.168.56.1	/192.168.1.214	20180503_040031	20180503_040230
0 / 00027	Xelnaga	Oritaka			Benzene	00:19:20	74261	31993	0.56917	0	0	0	0	0	0	/192.168.56.1	/192.168.1.214	20180503_040239	20180503_040603
0 / 00028	Xelnaga	Cimex			Benzene	00:13:33	45830	27980	0.38947	0	0	0	0	0	0	/192.168.56.1	/192.168.1.214	20180503_040612	20180503_040821
0 / 00029	ZZZKBot	Xelnaga			Benzene	00:04:18	5961	2638	0.55736	0	0	0	0	0	0	/192.168.1.214	/192.168.56.1	20180503_040830	20180503_040918
0 / 00030	Iron	IceBot			Benzene	00:21:47	115698	44829	0.61253	1	0	0	0	0	0	/192.168.1.214	/192.168.56.1	20180503_040929	20180503_041410
0 / 00031	LetaBot	IceBot			Benzene	00:05:16	4048	1528	0.62238	0	0	0	0	0	0	/192.168.1.214	/192.168.56.1	20180503_041420	20180503_041512
0 / 00032	IceBot	Oritaka			Benzene	00:14:19	54594	21978	0.59742	0	0	0	0	0	0	/192.168.56.1	/192.168.1.214	20180503_041522	20180503_041750
0 / 00033	IceBot	Cimex			Benzene	00:16:11	64401	21386	0.66791	0	0	0	0	0	0	/192.168.1.214	/192.168.56.1	20180503_041759	20180503_042034
0 / 00034	ZZZKBot	IceBot			Benzene	00:03:56	4806	1885	0.60766	0	0	0	0	0	0	/192.168.1.214	/192.168.56.1	20180503_042043	20180503_042129
0 / 00035	LetaBot	Iron			Benzene	00:19:27	87807	55218	0.37114	0	0	0	0	0	0	/192.168.1.214	/192.168.56.1	20180503_042140	20180503_042413
0 / 00036	Iron	Oritaka			Benzene	00:07:58	15875	3808	0.76008	0	0	0	0	0	0	/192.168.56.1	/192.168.1.214	20180503_042422	20180503_042538
0 / 00037	Iron	Cimex			Benzene	00:10:30	26473	8511	0.67848	0	0	0	0	0	0	/192.168.56.1	/192.168.1.214	20180503_042547	20180503_042718
0 / 00038	ZZZKBot	Iron			Benzene	00:14:11	11545	2269	0.8034	0	0	0	0	0	0	/192.168.1.214	/192.168.56.1	20180503_042727	20180503_042904
0 / 00039	LetaBot	Oritaka			Benzene	00:11:54	42684	18767	0.56031	0	0	0	0	0	0	/192.168.56.1	/192.168.1.214	20180503_042916	20180503_043117
0 / 00040	LetaBot	Cimex			Benzene	00:14:00	51223	24012	0.53122	0	0	0	0	0	0	/192.168.56.1	/192.168.1.214	20180503_043127	20180503_043349
0 / 00041	LetaBot	ZZZKBot			Benzene	00:10:30	23327	3833	0.83565	0	0	0	0	0	0	/192.168.56.1	/192.168.1.214	20180503_043359	20180503_043528
0 / 00042	Cimex	Oritaka			Benzene	00:59:31	139830	91692	0.34426	0	0	0	0	0	0	/192.168.1.214	/192.168.56.1	20180503_043538	20180503_044403
0 / 00043	ZZZKBot	Oritaka			Benzene	00:03:56	5555	2073	0.62671	0	0	0	0	0	0	/192.168.1.214	/192.168.56.1	20180503_044413	20180503_044501
0 / 00044	ZZZKBot	Cimex			Benzene	00:03:35	3890	1713	0.5595	0	0	0	0	0	0	/192.168.1.214	/192.168.56.1	20180503_044511	20180503_044557
1 / 00045	CruzBot	POSH-core			Destination	00:59:31	85766	24653	0.71255	0	0	0	0	0	0	/192.168.56.1	/192.168.1.214	20180503_044609	20180503_045523
1 / 00046	POSH-core	MegaBot	MegaBot		Destination	00:06:46	4117	10342	-0.60186	0	0	0	1	0	0	/192.168.1.214	/192.168.56.1	20180503_045533	20180503_045751
1 / 00047	Xelnaga	POSH-core			Destination	00:14:39	29404	6935	0.76412	0	0	0	0	0	0	/192.168.56.1	/192.168.1.214	20180503_045803	20180503_050024
1 / 00048	IceBot	POSH-core			Destination	00:09:42	18845	6578	0.65091	0	0	0	0	0	0	/192.168.1.214	/192.168.56.1	20180503_050034	20180503_050206
1 / 00049	Iron	POSH-core			Destination	00:08:10	15159	4847	0.68021	0	0	0	0	0	0	/192.168.56.1	/192.168.1.214	20180503_050215	20180503_050342
1 / 00050	LetaBot	POSH-core			Destination	00:08:27	16425	5733	0.65092	0	0	0	0	0	0	/192.168.56.1	/192.168.1.214	20180503_050353	20180503_050517

TABLE VII: An excerpt from the full table of results produced by the StarCraft Tournament Manager