Thesis no: MECS-2015-20

Cooperative Behaviors Between Two Teaming RTS Bots

in StarCraft

Robin Karlsson

Faculty of Computing
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Game and Software Engineering. The thesis is equivalent to 20 weeks of full time studies.

Contact Information:
Author(s):
Robin Karlsson
E-mail: robin.ph.karlsson@outlook.com

University advisor:
Dr. Johan Hagelbäck
Linnaeus University, Dept. Computer Science

Faculty of Computing
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

Abstract

Context. Video games are a big entertainment industry. Many video games let players play against or together. Some video games also make it possible for players to play against or together with computer controlled players, called bots. Artificial Intelligence (AI) is used to create bots.
Objectives. This thesis aims to implement cooperative behaviors between two bots and determine if the behaviors lead to an increase in win ratio. This means that the bots should be able to cooperate in certain situations, such as when they are attacked or when they are attacking.
Methods. The bots win ratio will be tested with a series of quantitative experiments where in each experiment two teaming bots with cooperative behavior will play against two teaming bots without any cooperative behavior. The data will be analyzed with a t-test to determine if the data are statistical significant.
Results and Conclusions. The results show that cooperative behavior can increase performance of two teaming Real Time Strategy bots against a non-cooperative team with two bots. However, the performance could either be increased or decreased depending on the situation. In three cases there were an increase in performance and in one the performance was decreased. In three cases there was no difference in performance. This suggests that more research is needed for these cases.

Keywords: AI, cooperative behavior, cooperation, real time strategy

Preface

Acknowledgements

I would like to thank everyone who has helped me to complete this Master thesis, especially my supervisor: Johan Hagelbäck. His positive attitude during the project was unbelievably helpful for me, it helped me keep focus during stressful times.

Further I would like to thank all of my family. Especially my girlfriend, Matilda Liljegren, for her enormous support during the whole project. Special thanks to my father, Christian Karlsson, for encouraging me and his ability to restore faith in myself when it wavered during the years.

Finally I would like to thank my friends for their valuable input and comments. Especially Erik Persson for all his help.

Abbreviations

AI Artificial Intelligence

RTS Real Time Strategy

FoW Fog of War

BWAPI Brood War Application Programming Interface

API Application Programming Interface

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Artificial Intelligence (AI) is a field of study that is used to create intelligent systems that are capable of showing intelligent behavior. An intelligent agent is an autonomous entity that is capable of observing its environment and act upon it to complete its goals. An intelligent agent is capable of making own decisions. Agents can be used to create an intelligent system that has the ability to play video games. In video games there exist goals that a player needs to accomplish for advancing or winning the game. Agents can be used to simulate players for video games, usually referred to as bots.

Some video games let players play with each other, either against or cooperating with each other. By playing together the goals become shared between the teammates and the goals can be accomplished together. Sharing and completing mutual goals together is the definition of cooperation.

According to Robertson and Watson there exists no research in cooperation between bots [19]. They also state that there is a difference between bots developed by commercial game companies and bots developed for academic research. For commercial bots it is more important to give the player a fun and enjoyable gaming experience while having high replay value whereas academic bots focus on winning. This creates two interesting research areas: does cooperating bots increase the chances of winning or is cooperating bots more fun to play against? The aim of this research is to see if cooperative behavior between bots increases the chances of winning against non-cooperating bots.

This report will first present a description of Real Time Strategy games in general and the game StarCraft. Then a background on cooperation and an introduction of AI teammates is presented. Which is followed by the presentation of related work. Afterwards the aim, research questions and objective are presented. The report will also give a description of the implemented system as well as the cooperative behaviors. Afterwards the methodology of the experiments is described. Then the results and analysis is presented which is followed by a discussion of the results. Lastly the conclusions and future work are presented.

1.1 Real Time Strategy

Real Time Strategy (RTS) is one of many genres in video games. RTS games are based on real time decisions instead of turn-based. It is derived from the turn-based strategy games such as chess. In RTS games players are independent of each other and are capable of playing the game simultaneously without waiting for other players to finish their turn. This makes time a valuable resource in these types of games compared to turn-based games, since if a player is not spending his time effectively his opponent gains an advantage. Another difference with RTS games and chess is that chess has perfect information. This means that each player sees exactly what the opponent does, for example which pieces each player have left. In RTS games the player only has information about the area that is visible by any of his pieces. The rest of the game world is hidden, this is also known as Fog of War (FoW), which makes these games operate in imperfect information conditions.

The key components to RTS games are resource gathering, advancing in technology, base building, recruiting and controlling units. As the name implies, real time and strategy, it is of utmost importance to strategically and quickly manage these key components. The gathered resources are used to advance in the technology tree, construct buildings and recruit units.

A typical RTS game is set in a military setting where units (similar to pieces in chess) are used to destroy units and buildings of the opponent. It can be fantasy, sci-fi, modern combat, historical warfare or something else entirely. The goal of a game is to defeat your opponent by destroying all buildings and units of the opponent. In most RTS games there is a choice to play against other human players or against bots or a combination of both, for example a team of human players against a team consisting only of bots.

1.2 StarCraft

StarCraft is a computer RTS video game made by Blizzard Entertainment and was released in 1998 [3]. An expansion of the original game, called Brood War, was released later that year. StarCraft has been one of the most popular RTS games mainly because it is considered to be very well balanced and has therefore been a very popular e-sport game [18]. Which has made Blizzard Entertainment maintaining the game for many years. Its sequel, StarCraft 2, is currently considered to be the most popular RTS game. Henceforth the game will be referred to as only StarCraft. Information about StarCraft that is presented below has been collected through various sources [4, 14, 21].

StarCraft is a non-deterministic game which means there are random elements that affect the outcome of actions, for example units have a chance to miss attacks. This means that the outcome of a match is unpredictable even if all players are

doing exactly the same thing every time.

1.2.1 Races

There exist three different distinct playable races in StarCraft: Protoss, Zerg and Terran, all with different units, buildings and playstyle. Below are three short descriptions of the different races:

- Protoss is a highly evolved alien race that has psionic abilities. These abilities are enhanced with robotics using the advanced technology of Protoss. Their units are the strongest in the game but also the most expensive, which leads to players having fewer units on the battlefield. All units and building used by Protoss is protected by a shield that will regenerate if damaged.
- Zerg is an alien-like race. The goal of the Zerg is to assimilate other races into their race to become even stronger. Their units are the weakest but also the cheapest. They focus on numbers and try to overwhelm their enemies. Zerg buildings regenerate health back and their units will generate health when standing on creep, a Zerg organic substance.
- Terran is an advanced human race and is the third and last race. Their units are weaker than Protoss but stronger than Zerg and focuses on traditional warfare with tanks, air units and marines in their army. They need to repair buildings and mechanical units and heal organic units when damaged.

1.2.2 Supplies

A common requirement between the races is the need for supplies. Supplies determine the maximum number of units that can be constructed and constructing units costs supplies. Protoss use pylons as their supply building which also functions as a power generator. Zerg uses overlords, a flying alien which also is a transport unit. Terran has supply depots and their only function is to provide supplies. Supplies are considered to be one of the three resources in the game.

1.2.3 Resources

The other resources are minerals and vespene gas and needs to be gathered and dropped off at a command center. Minerals are the most common resource and multiple veins exist at each base location and expansion site but the total amount that can be gathered from each vein is limited. Only one mineral vein can be used at a time, workers must wait for their turn if a vein is used. Vespene gas is gathered from gas geysers, which are located at start and expansion locations and there is usually only one gas geyser per location or site. All races are required to build an extractor on top of a geyser. The gas in the geysers is unlimited, but a

geyser will collapse after certain amount of gas has been collected and then the amount of gas per worker is reduced significantly. Considering that both minerals and gas are limited makes expansion necessary.

Units, buildings and upgrades have a mineral and gas cost. Early units, buildings and some upgrades only requires minerals and gas only becomes required when building, upgrading or recruiting more advanced types.

1.2.4 Units

Different unit types have different attack speed and attack damage, and both ranged and melee unit types exist. Some unit types have an ability that can be activated by the player. The game also has some units that cannot attack other units, for example the unit is a support unit and can heal other units.

There exist three different types of damage and unit sizes. The three unit sizes are small, medium and large. The unit size determines which type of armor units have. The three attack types are normal, concussive and explosive. Depending on the armor type, different amount of damage is dealt to a unit. Normal damage deals 100% damage to all armor types. Concussive deals 100% damage and explosive deals 50% damage to small armor. To medium armor concussive deals 50% damage and explosive deals 75% damage. Against large armor concussive only deals 25% damage and explosive deals 100% damage.

Because of the different characteristics of unit types, damage types and armor types units counter other units in a rock-paper-scissor fashion. This makes it crucial for players to combine different unit types to be able to counter the enemy units' strength and complement own units' weaknesses.

A worker is a special unit type that all races have in common. A worker is used to gather both types of resources but also for constructing buildings. Each race has their own workers that work a bit differently from the other races. Protoss workers, called probes, warp buildings instead of constructing them making probes not needing to be present at the construction site as the building will continue to warp in without help. Protoss buildings require power to function which is generated by their supply buildings. Zerg workers, called drones, morph into a building and are therefore lost when the building is finished. Zerg buildings are organic and generate creep that grows outwards from their buildings. New buildings can only be constructed on locations covered by creep. Terran workers, called SCVs construct buildings from start to completion. Terran is the only race that has no requirements when placing buildings. No race can place buildings upon other buildings. Therefore buildings must be spread out in the base. Buildings are considered to be large units. Some buildings require that another building is already constructed, leading to advancement in available units and technologies.

1.2.5 Upgrades

The technology tree, or tech tree, is a hierarchical tree of upgrades. The parent of a technology must be upgraded before the technology is available. This makes it possible to only focus on upgrades that are needed and not needing to spend resources on upgrades that do not suit a player's playstyle.

An upgrade enhances existing units. Some upgrades only affect one type of unit while others affect multiple types. Each upgrade requires time and resources and will only get more expensive and time consuming for each research level. Tech buildings are needed for these upgrades and they will also unlock more advanced units and buildings.

1.2.6 Gameplay

The game can be divided into two distinguishing tasks: macromanagement and micromanagement. Macro, short for macromanagement, is the global activities in the game: resource management: building units, constructing buildings, advancing the tech tree, gathering resources and when to expand. Micro, short for micromanagement, is the local activities in the game: how units are being controlled. This includes all commands executed by the player given to the units: abilities, placement of armies, attacking, retreating, defending, and focusing fire. What differentiates these is the level on which the game is played. Macro is high-level management where micro is low-level management.

1.2.7 Reasons for StarCraft

The Brood War Application Programming Interface (BWAPI) is a free and open source C++ framework that is used to interact with StarCraft. The interface makes it possible to read the game state, supplies, resources, issue commands to units and buildings and more. These are fundamental features that are required to build a bot for StarCraft. The API is also able to hide information covered by FoW to the bot making it possible to create a non-cheating bots that needs to deal with uncertainty that comes with imperfect information [7].

Commercial bots sometimes use cheats to gain an edge over the player, for example by having perfect information or having increased building speed for buildings and units. The cheating is done to be able to provide skilled players with a challenge as it is very difficult to make a bot competitive against medium and upwards skilled players.

StarCraft is a popular platform for developing bots and a lot of the research that exists into RTS AI is being developed with the help of BWAPI [8]. BWAPI is considered to be one of the best platforms as it allows for the development of AIs for a commercial RTS game.

Balance is a high priority in e-sport games and being an e-sport for many years StarCraft have received many patches and balance fixes. A balanced game should make experiments with it more robust.

The project will use an already functional bot that is capable of managing the core gameplay of StarCraft. This makes it possible to focus on the implementing the cooperation between bots instead of core gameplay.

1.3 Cooperation

A definition of what a teammate AI in real-time is that it should take the behavior, state, needs and plans of the players it cooperates with into account. The teammate shall be a part of coordinated behaviors or decision-making and share relevant goals (or sub-goals). Coordinated behaviors and decision-making includes player-related uncertainty that requires real-time calculation, reasoning, interference or other methods. The teammate should prioritize the player participation whenever it is possible [16]. It is also stated that many bots only supports their teammate by sharing a mutual goal.

The definition from McGee and Abraham is about an AI that plays with a human player. If a bot is teaming with another player, human or bot, different techniques are required. Gathering information about a player is referred to as player modelling and can be used to try and predict the next move. Teammate modelling is a form of player modelling but focuses on a teammate. With this information a teammate's state, behavior and plans can be answered and from this the bot could possibly recognize the teammate's strategy. This could be taken into account when the bot makes its own decisions. Teammate modelling is similar to opponent modelling which has existed for some time [9]. Opponent modelling is also mentioned as one of the key problems in Real-Time Strategy (RTS) games [6] and that it is an essential part of a player's match preparation [20].

In current RTS games it seems that bots are acting independently without any collaboration with its teammate [19]. Abraham and McGee presents four models of real-time teammates and their relationship [1]. These models are master-slave, semi-autonomous slave, clone and buddy. Magnusson and Balsasubramaniyan added one more model to this list, a semi-autonomous model [15].

- **Master-Slave model** is a relation where one is the master and the other follows. The master is the controller of its teammate and will issue orders to its slave, which will carry them out. The slave must obey its master at all times and the slave is incapable of actions unless the master commands it.

- **Semi-autonomous slave model** is very similar to the master-slave model, but the slave is capable of operating without its master's commands. The

slave must however obey the master's commands.

- **Semi-autonomous model** also has a similar relationship as master-slave and semi-autonomous slave. This model is however different in that it is able to disobey its master if deemed necessary. For example if the master orders an attack when the teammate is under attack, it ignores the command and focuses on defenses instead.

- **Clone model** is a different kind of cooperation mode as it requires a team of multiple clones (agents) which shares a goal. All clones share the same abilities and roles which makes it possible for a single bot to complete the shared goal. The main reason for this model is to be able to use more clones to accomplish a goal faster.

- **Buddy model**, the buddy should be capable of dynamically adapt to the needs and behaviors of its teammate. The buddy is acting more as an equal player than a slave or clone.

These models present the relationship between a human player and an AI teammate. It is also possible to apply these relationships on two teaming bots as the relationship between the teammates stays the same.

By using the buddy model, Abraham and McGee created a bot in a simple game called Capture the gunner. Being a buddy bot caused the players to feel as they had a responsibility over the bot and because of this some players felt it was partly their fault if the bot died. It is stated that this game requires cooperation between a player and a bot because none of the testers that ignored the bot succeeded in completing the first level. This suggests that the cooperation improved the team's performance [1].

1.4 Related Work

As no research about cooperative behavior between RTS bots exists the presented articles in this section focuses on cooperative behavior in general and its effect on different techniques, methods and approaches.

The suggestion that cooperation can improve performance is also stated by Cutumisu and Szafron. They added cooperative actions to multiple behavior queues that proved to be efficient and robust enough for commercial games as the technique could run for days without any noticeable stalling or getting stuck with high activity. This means that the cooperative actions added to the technique made it perform better [10].

For example Cutumisu and Szafron added cooperative actions to multiple behavior queues that proved to be efficient and robust enough for commercial games as the technique could run for days without any noticeable stalling or

getting stuck with high activity. This means that the cooperative actions added to the technique made it perform better [10].

Zhou and Shen presented in their article a multi-agent reinforcement learning approach that uses traditional Q-learning, teammate modelling and reward allotment [23]. The approach is to increase the learning rate of their multi-agent system, to which the least amount of actions lead to a successful goal in a cooperative environment. The approach was tested against two similar approaches. The first approach is an individual Q-learning, which means that each agent does not take the other agents into consideration when learning. The second approach lets the agents take each other's strategies into account with teammate modelling when learning. Their approach did show an increase of the learning rate compared to the other two approaches. Also when only considering the individual Q-learning technique against Q-learning with teammate modelling the later come out ahead in their experiment.

When applying decision techniques to an AI, who's main goal is to support a human player in a cooperative game, the AI is modelling the player's goals as states and with the help of a Markov decision process determines which goal the player is currently performing [17]. It was shown that the AI had near-human level performance when helping its teammate. Nguyen et al. also showed that the cooperative AI outperformed an AI that only performed random moves independent of the human player. However this could be a result of the random movement instead of not being able to cooperate.

The thesis by Magnusson and Balsasubramaniyan presents a bot that can cooperate with a human player and communicate its intent [15]. The player is able to give orders to the bot and the bot will respond with a comment depending on what it decided to do. New players to the game found it too overwhelming to both play the game and control a bot, however they found the communication aspect fun. The results from their study showed that it is more fun to play with a bot that is controllable and able to communicate.

Jansen created an adaptive cooperative bot that is able to support and complement its teammate's strategy [13]. When applying opponent modelling techniques to model the teammate the bot is able to do pattern recognition and is able to analyze the strengths and weaknesses of its teammate. The bot uses a decision tree to be able to determine if the teammate is in an aggressive or defensive state and with the use of a neural network the bot is able to mimic its teammate actions. The bot however had one problem. It cannot determine which of the possible actions that is the most optimal at a given situation.

Chapter 2

Problem Description

A common statement from the presented research is that cooperation had a positive impact on their results. As no research in cooperative behavior between bots exists makes it an interesting research opportunity to be able to determine what effect cooperation has on two teaming bots.

2.1 Aim

The aim of this thesis is to implement cooperative abilities between bots to an already functional bot. The thesis should determine if the implementation leads to an increase in win ratio. This means that the bots should be able to cooperate in certain situations, such as when they are attacked or when they are attacking.

2.2 Research Questions

The performance, meaning the win ratio, is decided by the win loss scores from multiple matches. RQ1: Is there an increase in performance when two teaming bots cooperate compared to when two teaming bots do not cooperate?

1.1. Do cooperative assist actions lead to increased performance?
1.2. Do cooperative defend actions lead to increased performance?
1.3. Do cooperative attack actions lead to increased performance?
1.4. Do combining assist and defend actions lead to increased performance?
1.5. Do combining assist and attack actions lead to increased performance?
1.6. Do combining defend and attack actions lead to increased performance?
1.7. Do combining assist, defend and attack actions lead to increased performance?

With the categorizing of cooperative situations the sub questions were created to help answer the main research question.

2.3 Objectives

The main objective is to add cooperative functionality upon an already functional bot without affecting the performance of the functional bot. By using an already functional bot a lot of time was saved that was used to implement cooperative abilities. The implemented abilities are an extension of the functional bot and therefore the performance has not been affected. A definition of what fully cooperative bot is capable of and a description of what a fully cooperative bot means were created. A fully cooperative bot means in this thesis that it has the decided abilities that are needed to provide answers the sub RQs. The actions and abilities need to be implemented to the original bot.

An environment needs to be set up for the experiments to be able to measure the performance. There is also a need to solve some problems that comes with BWAPI and StarCraft: There is no way of creating teams outside of a game, or retrieve and save score from multiple. One map needs to be selected and used for all experiments. The map will be selected from a pool of tournament maps that is known to be well balanced and should not introduce unbalanced starting locations or any disadvantage on the rest of the map.

An individual bot's performance is irrelevant for this thesis, for example there is no need to improve scouting, strategies and more as all teams would benefit from it.

Chapter 3

System Description

The chosen bot for this thesis is called OpprimoBot [12] and is developed by the thesis supervisor, Hagelbäck. A reason for choosing OpprimoBot was that it is developed by the supervisor. This enables great and fast support if needed due to his close familiarity of the thesis and bot. OpprimoBot will henceforth be reference to as the original bot. The Terran race was chosen for the bot to play as the original bot works better with Terran than with the other two races.

The original bot is a multi-agent system based bot. A multi-agent system is where multiple intelligent agents are interacting with each other. Different agents can have different goals and with multiple agents interacting in the same system more complex problems can be solved [11, 22].

A multi-agent system can have a hierarchy structure which means that agents are at different levels depending on its functions. A high-level agent would make decisions on a higher level that requires the ability to utilize or command low-level agents. This structure is similar to a military hierarchy in which a general makes decisions and pushes orders down the chain of command. It is often referred to as a command hierarchy.

3.1 Agents

In the original bot there exist five high-level agents, but three of them could be considered to be manager agents. A manager agent is an assisting agent which main task is to help and provide information to other agents, thus assisting the other two agents in making their decisions. The three manager agents are map, exploration and agent manager.

Map Manager's main task is to create and update an influence map based on enemy units. An influence map provides useful information about the world, for example who is controlling what region, where the frontline is and unit strength of players, both enemy and friendly. The unit strength determines the influence. An example of how the influence work is shown in Figure 3.1 by the red and blue circles. Blue circles indicate your own influence (positive) and red circles indicate the opponent's influence (negative).

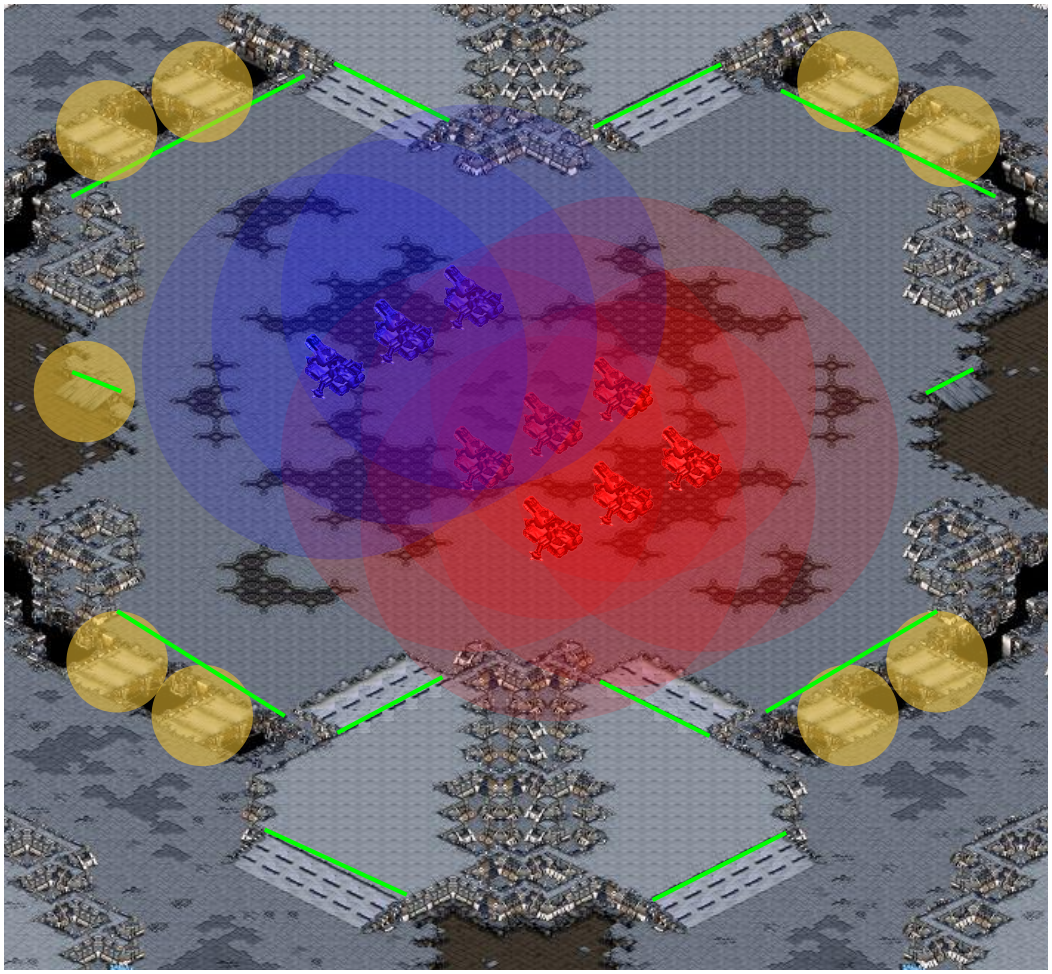

Figure 3.1: Yellow circles indicate chokepoints and green lines shows where two regions connect. The area between these green lines is called a region. In the middle blue and red circles are shown to simulate influences. This figure is a simplification of how regions and influence work.

The influence map is used to find attack and defend positions. Finding a defensive position is determined by finding which chokepoint is closest to the base. Chokepoints are small passages that usually connect two regions together and are shown as yellow circles in Figure 3.1. Finding an attack position is done by using the region with the lowest influence and finding the building closest to that region, even if it is outside the selected region.

Exploration Manager remembers spotted enemy buildings otherwise the bot would forget them when they are covered by the FoW again. It also provides information on the scouting by keeping track of which region is currently being explored and which region shall be explored next. A region is explored when the center of it is visible. The next region to be explored is determined by finding the region which has not been explored for the longest time.

Agent manager is a container of low-level agents and has a list of all living agents. These agents are added when recruited/built and removed when destroyed/killed. There exist two categories of low-level agents: structure agent and unit agent. Every building, unit and worker is assigned to an agent and every agent has a unique agent type. The low-level agents are an extension upon a BWAPI unit.

Some low-level agents make decisions based on their agent type, for example medic agents will look for damaged units to heal. Each structure agent is responsible of recruiting the units the building type is capable of.

Agents are capable of returning fire and fire on enemy units and buildings without getting any orders to do so. A targeting agent exist that help units pick a target. When a target exists the units can engage it. An agent is also capable of moving around the map. To do this it uses a navigation agent that calculates the route and gives agents their next position to go. This makes units on their own capable of micromanaging the game.

Unit agents, that not are buildings or workers, are grouped together in squads. Squads have a goal which is a position they try to reach. As units are capable of moving and firing on their own, a squad can do this as well. When a squad has a goal all units in the squad will move towards the goal.

The three manager agents will provide their information to the other high-level agents: constructor and commander agent. These two are making decisions based on the information provided.

The constructor agent's task is to construct buildings. The agent has a queue of which buildings to construct next. An abstraction of the map exists and is used by the constructor to know where new buildings can be placed. The constructor agent is also in charge of expansion. When the combined minerals of all expansions have reached a certain amount the agent adds a new resource depot to its queue and finds an expansion site to build at. The expansion will not happen until the resource depot is to be built.

The agent also deals with the different races' unique way of constructing buildings, for example the constructor agent will assign a Terran worker agent to the building, making sure that it finishes. If the worker agent is destroyed the constructor agent will assign a new one.

The resource manager is in control of the resources and will grant other agents access to spend resources. If an agent is allowed to spend resources the resource manager will lock it from other agents. This is done to make sure that the worker is able to construct the building when it arrives at the build site.

A strategy is represented as a state machine with predefined instructions that the bot follows. A strategy contains the unit configurations of squads and when buildings shall be placed into the building queue and when technology shall be advanced. Transitions between states occur when certain conditions are met during a match. Conditions vary but can for example be that a higher total supply limit is met or that a certain building has been built. New states add

different instructions, for example new or more units and buildings or upgrades.

The commander's main task is to macromanage a game. It sets a goal for each squad immediately when conditions for the type are met. A goal could both be a defensive or an attack position depending on the state of the commander. The commander is in either defensive or offensive state. The bot starts in the defensive state and will change state if it decides to engage the enemy, which happens when certain predetermined squads are filled with units. In the defensive state the goals will be at defensive positions, for example chokepoints. If a building is under enemy attack the commander will immediately order all its squads to defend it. A goal in the offensive state is an enemy building to attack. The map manager is used to obtain positions for both defensive and offensive goals.

The commander will add buildings to the constructor queue when new strategy stages are reached. When a new unit is built it is assigned to an agent, and the agent in turn is assigned to a squad. Structure agents will only recruit units if there are units missing in any squad, i.e. the squads are not full.

An overview of all the components and their relations can be seen in Figure 3.2. The figure is a simplification of the original bot and is not the true representation.

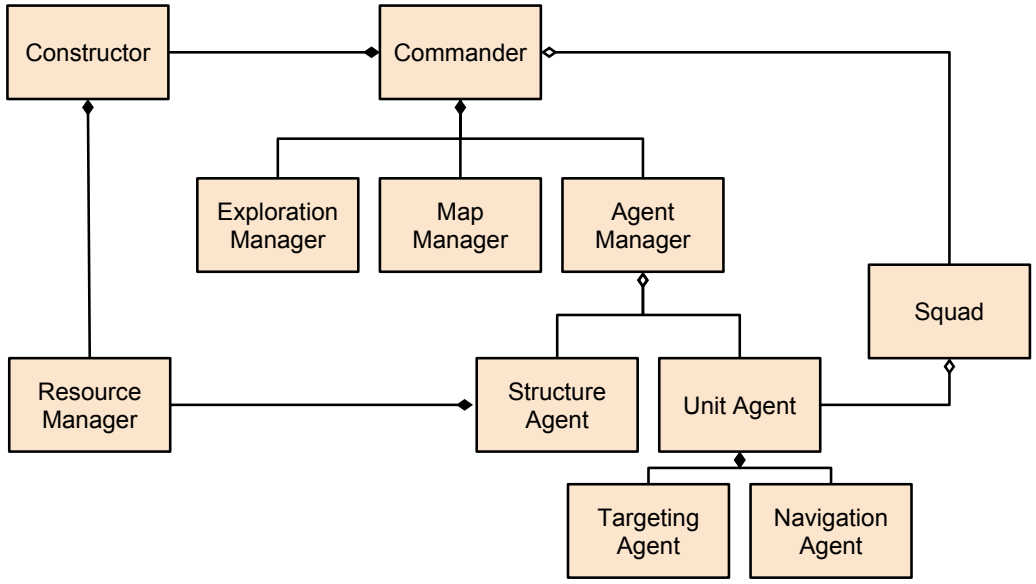

Figure 3.2: A simplified overview of the original bot with components and their relations.

The original bot utilizes the singleton pattern for every high level manager, which means the commander, constructor, agent manager, exploration manager and map manager. Singleton is a design pattern that is used to ensure that only one instance of its object exists in the program.

3.2 Changes to original bot

Restructuring the original bot was necessary to be able to implement the cooperative behavior. As mentioned the commander gave orders directly when conditions for an action were met and the need for postponing logic was required. To have more freedom when in time orders are executed a simplified event system, called triggers, was implemented

3.2.1 Triggers

Trigger is an object that needs three variables: trigger type, position and priority. Created triggers are added to a list that is managed by a trigger manager. The trigger manager is a singleton and holds all triggers created during an update.

All variables that are needed exist in the same scope and is therefore calculated when the trigger is created. The trigger type is used to determine what event has happened and is used to execute different logic. A position is needed for a squad's goal and the priority is used for being able to sort the triggers in order of importance.

The attack and defend triggers that are created by the commander has a lower priority as one of them will always happen. Some triggers use the number of units visible, for example if there are two teammates that are under attack, the teammate that has more enemy units at its base is prioritized.

With the possibility of multiple triggers with different priorities during an update it was necessary to find the trigger with the highest priority. Finding the highest priority is a simple selection by iterating over all triggers that the trigger manager currently holds and compare the triggers priority. When a priority is higher than the previous it switches the triggers and repeat this process until it reaches the end of the trigger list. When a trigger is found the trigger manager clears the list and returns the selected trigger which enters a switch statement and the specific logic for that trigger is executed.

Triggers make it possible to postpone logic that would otherwise have caused the bot to apply multiple goals during an update.

3.3 Cooperation

The cooperation module is an extension of the original bot. The module is only accessed through the commander. It is unlike the commander not implemented as a singleton. An extension makes the implementation more isolated and new implementation would only benefit the cooperative bots.

The module is responsible for the cooperative behavior. This includes teammates, analysis and cooperative functionality.

3.3.1 Teammate

Teammates exist as player models in the bot. Player models hold information about a teammate's units, buildings and the current status of the teammate. At every update the lists are cleared then re-filled by iterating over all the teammate's units and buildings. By filtering unnecessary units, for example workers, and only add buildings to the building list means fewer iterations required when the need for access the teammate's buildings. This was done because it caused no crashes as was the case when the BWAPI event system was used.

A player model also has the status of the teammate: offensive or defensive. These states are the same as the original state that the commander uses. As in the original bot the player model state defensive as default.

3.3.2 Cooperative functionality

BWAPI has functionality for allowing bots to use the in-game chat system. This enables communication between teammates. It was decided that communication should be used as it is a simpler solution than advanced teammate modelling to the state of a teammate. A simple announcement is used to let the teammate know that a state change has occurred.

A fully cooperative bot shall be able to:

1. Communicate about state changes.
2. Remember the state of the teammate, e.g. offensive or defensive state.
3. Keep track of teammate's units and buildings.
4. Assist a specific teammate's unit and help the one who is in most danger.
5. Assist damaged teammate's units by repairing/healing them.
6. Defend teammate's buildings from enemy units.
7. Defend teammate's units when it is in a defensive status.
8. Start attacking together by utilizing teammate's army when checking shall engage condition and announce the attack.
9. Remember locations of enemy buildings when they are found by the teammate.
10. Keep together by attacking the same location as its teammate.

The bot's cooperative functionality can be divided into three categories: attack, defend and assist. In each category there are different kinds of abilities the bot is capable of performing. Abilities are defined by the functionality it uses for cooperation. The functionality needed can for example be analyzing the enemy or teammate's units. Each category holds different abilities to cooperate with its teammate. Abilities 4 and 5 belong to the assist category, abilities 6 and 7 belong to the defend category and abilities 8, 9 and 10 belong to the attack category.

These abilities are on a micromanagement level, which make the bots not able to cooperate on a higher level such on the macromanagement level.

3.3.3 Assist

The list of units that exists in the player model of the teammate is used for the ability to heal and repair damaged units. Heal and repair works in the same way, the only difference is that heal works on biological units and repair on vehicles. The right agent type has access to this list when it checks for units that are damaged. The logic for healing and repairing units is the original implementation but it now has access to the teammate's units as well.

As the bot will play the Terran race it was decided that an ability would be to assist important Terran units. The siege tank is considered to be the strongest unit in the Terran army with a large armor type and is dealing explosive damage. The unit is able switch modes and be deployed, which more than doubles the amount of damage the unit deals as well as increases its firing range. The tank is however stationary when it is deployed and its firing range is longer than its sight [5]. A good tactic is therefore to defend it from units that are effective against large armor types and increase the visibility around the siege tank.

Figure 3.3 shows how the threat is determined by checking the damage types that are most effective against siege tanks. The most threatened siege tank will be focused and the assisting bot will move its units towards the enemy unit closest to the siege tank.

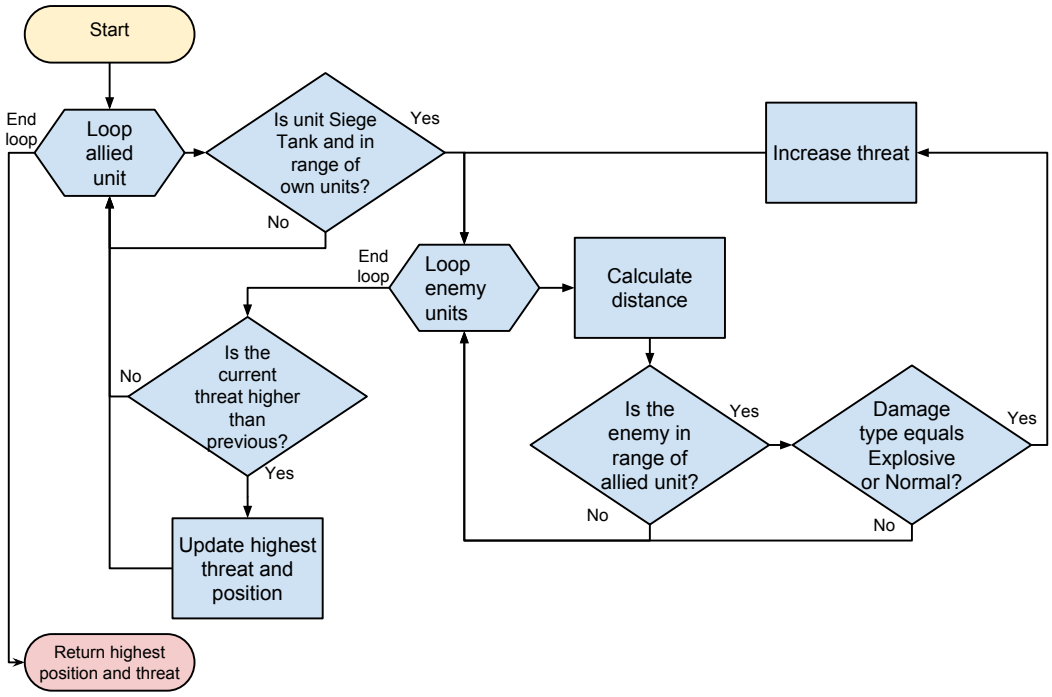

Figure 3.3: Flow chart of the implementation of how to find the most threatened siege tank. The siege tank has to be close to avoid units traveling very long distances to support it.

3.3.4 Defend

When enemy units are near a building and no friendly units are present then the enemy units will attack the building. With units present there is a chance that the enemy wins the skirmish and proceeds to attack the building afterwards. Fast reactions are very important in this case and the faster the bot can help its teammate could help increase the chances that the teammate survives the attack.

The bot is able to defend both its teammate's units and buildings. Defending the units should give the bot the advantage of being closer to the teammate's base before the buildings are under attack. The bot is only allowed to defend the teammate's units when the teammate is in a defensive state because the bot shall not defend the teammate's units when both of them are attacking.

3.3.5 Attack

The original functionality in the exploration manager for remembering units had to be changed because it allowed for unwanted cooperation such as bots attacking the same location. This was caused by the original bot as it always added new enemy buildings to be remembered. Figure 3.4 shows that the exploration manager is not allowed to add spotted units without checking which unit spotted the building, and only when the attack category is enabled the bot is allowed to add units discovered by the teammate. This function is executed whenever an enemy unit is spotted.

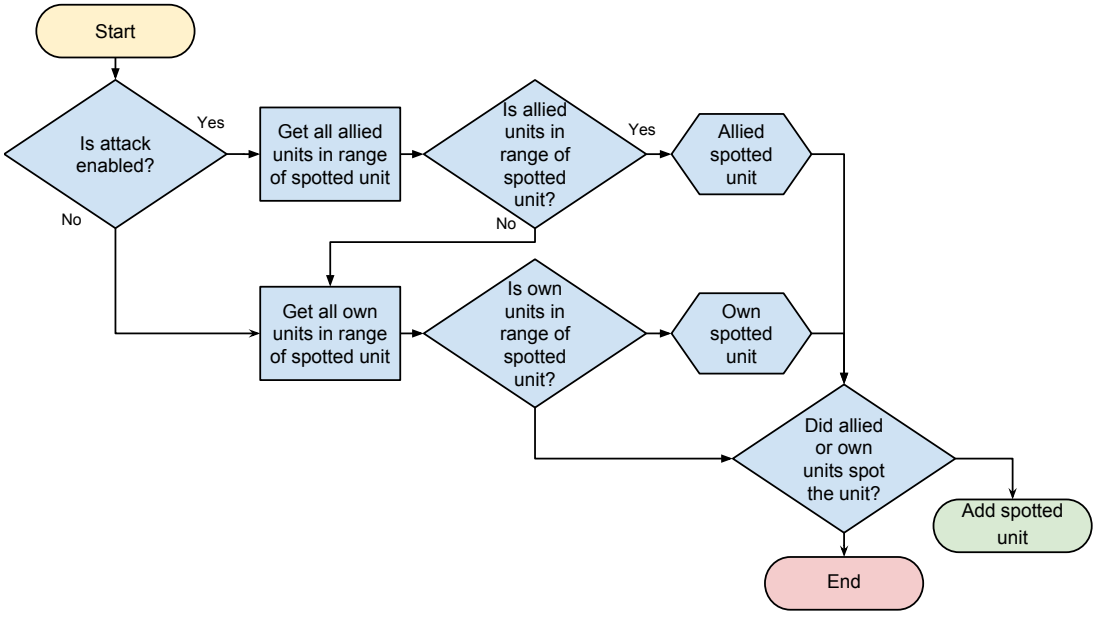

Figure 3.4: Flow chart for remembering spotted units that executes when a spottedUnit event is created from BWAPI.

When the bot's condition for attacking is met the bot will message its teammate about launching an attack. The teammate will check the number of units in its main squad and determines if it also is capable of launching an attack. Due to the existence of two armies both bots condition for launching an attack is reduced. The unit strength only has to be a percentage of the full squad size. If the teammate agrees to attack the bots launch the attack.

By remembering the locations of the same buildings the bots could decide to attack the same enemy. To guarantee that the units of both bots stay together after the initial attack the bot will attack an enemy unit that is near the teammate's units.

The ability to attack the same position was created due to for what military tacticians call force concentration. This is a practice which states that there is force in numbers, meaning that a bigger army is better than a smaller army. Attacking the same position makes the combined army much larger than the army of a single bot, which hopefully outnumber the enemy being attacked.

3.3.6 Analyze

Analyze is a utility class that is used to supply functions that are capable of analyzing different aspects of the teammate or the bot itself. Analyzing itself occurs when the teammate is asking if bot shall launch a joint attack. The function iterates over the existing squads and selects the squad with the highest number of total agents (the number of agents the squad holds) and checks if the current active agents reach a certain percentage of its full capacity. If the number of active agents exceed a threshold of half the original full capacity it returns true.

Each fifth frame an analysis update is requested. The flow of the update function is broken into smaller flow charts. A yellow node means the start of an analysis. A green node is another analysis that is separate from the current flow chart and a red node means that the update function is done and the end of the analyze update has been reached.

Figure 3.5 shows the flow for determining if the bot shall defend allied buildings. The flow chart for defending teammate's units is similar enough not to warrant its own chart. The only difference is that an added decision that checks the teammate's status.

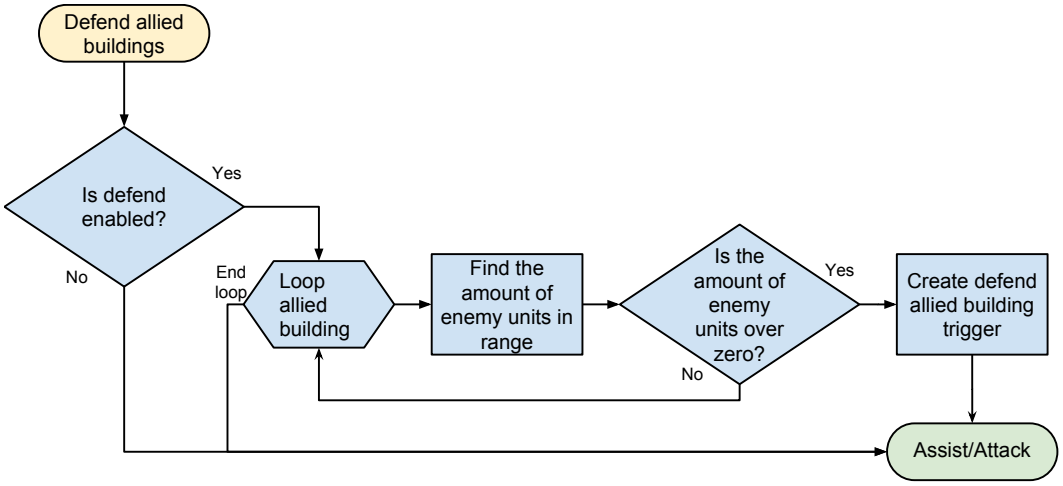

Figure 3.5: Flow chart for defending teammate's building.

Figure 3.6 shows what happens when the teammate is in offensive state and if either the assist category or attack category is enabled. This is where the most threatened siege tank function is used and it is shown as a grey colored box. The last decision leads to either defending the teammate's units or ending the analysis.

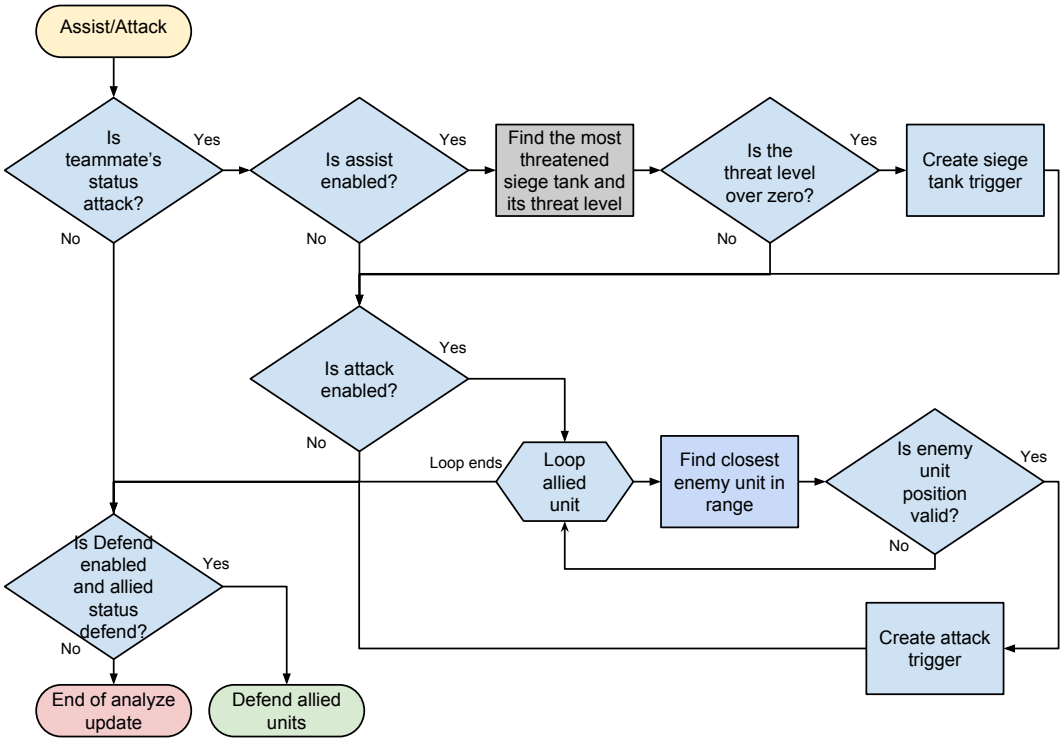

Figure 3.6: Flow chart for assisting siege tanks or keeping together with teammate.

3.3.7 Overview

A simplified overview of how the cooperation components have been integrated into the existing bot can be seen in Figure 3.7. As seen in the figure the cooperation components are isolated from the rest of the bot and can only be accessed through the commander.

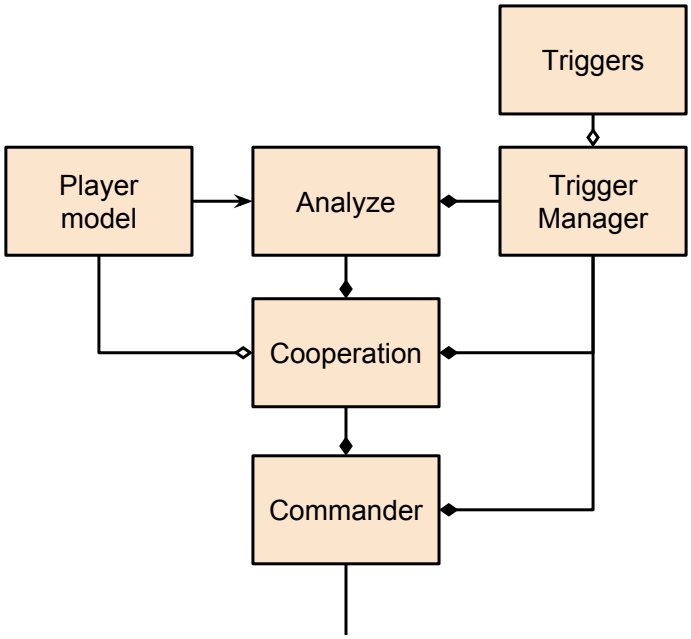

Figure 3.7: Simplified overview of how the cooperation module is integrated in the original bot. The components and their relations are also shown.

3.4 Extended features

To be able to change the cooperative functionality of the bot without recompiling, a simple configuration file was created. This configuration file contains settings for cooperative functionality of a bot. By modifying this file it is possible to enable and disable categories. The configuration functionality is implemented as a singleton, and exists in the memory for the duration of a match for the need of knowing which category is enabled or disabled.

The bot is capable of recording statistics for a match. When a match ends an event from BWAPI is created informing the bot that the match has ended. The event provides the outcome of the match which is either a loss or a win. This information is used for the experiments.

A safety measure is implemented that make all bots end a match after 101 minutes to ensure that a match ends in a reasonable time. The reasons for this are there exist a chance of bots stalling. Stalls are when a bot gets stuck in logic and

cannot move forward with its functionality such as recruiting units, constructing buildings due to not collecting resources or that the army is stuck at a chokepoint. When a match is canceled the outcome is recorded as a draw, which is the third type of outcome recorded for a match.

Chapter 4

Methodology

The null hypothesis of this thesis is that cooperative behavior between bots does not affect the performance. If the null hypothesis is rejected the alternative hypothesis is accepted, which is that cooperative behavior leads to an increase in performance.

To answer the research questions and reject the null hypothesis a series of quantitative experiments were conducted. Two teams of two bots each played in a mode called Top vs Bottom. Top Vs Bottom is a game mode in StarCraft that makes it possible for players to play in teams against other teams. For each match its outcome was recorded.

A total of 200 matches were conducted for each experiment and as time was limited more matches for each experiment were not possible. Team A and B have the same bot implementation and depending on the experiment team A had different categories enabled or disabled. Seven experiments were conducted to be able to test all combinations of the three existing categories. Table 4.1 shows the order of the experiments from left to right.

Assist	x			x	x		x
Defend		x		x		x	x
Attack			x		x	x	x

Table 4.1: Experiment order and combinations of the different categories.

The results, the statistical mean of team A's and team B's win percentages were analyzed with a statistical test called pairwise two-sample t-test between proportions. The calculations are used to test if the differences between percentages are significantly different from each other. The input data used was the number of matches played and the statistical mean win percentage of both teams.

The t-test outputs a P-value that is checked against a defined level of significance of 0.1. If the P-value is below the level of significance it is safe to state that there is a difference between the percentages and reject the null hypothesis. If it is above, the null hypothesis cannot be rejected.

4.1 Setup

The experiments ran on a dedicated computer able to emulate three virtual machines. Virtualization is a technique to emulate hardware using software. Virtualization makes it possible for multiple machines to run on a single hardware. StarCraft and BWAPI are capable of running on Windows 8.1 and Windows 8.1 uses less resources than it previous versions, so it was used as the operating system on all machines. The version of BWAPI in this thesis is 4.1.0b.

BWAPI has the ability to automate the menu selection in StarCraft by selecting a desired map and game mode. The chosen map is called Circuit Breaker and was selected as it proved to be the best map in the tournament pool as it caused the least problems such as stalls.

It was not possible to fully automate Top vs Bottom mode as it always defaulted to teams of one vs three instead of the desired two vs two. By selecting a map that does not exist makes BWAPI stop at the map selection screen, were it is only necessary to press OK to continue. To press OK, an autohotkey [2] script was used to press the letter 'O' (shortkey for the OK button), which made it possible to automate the creation of a match. BWAPI also makes bots auto join matches so this workaround was only needed for the creation of a match.

Due to how the auto joining process works, the teams cannot be predetermined. The solution for this was to use the names of the bot as a mean to determine if they shall be able to cooperate or not. The bot named Main or bots with a teammate named Main are allowed to cooperate. The cooperative team will therefore always be team A.

To not record duplicate results of matches only the cooperative bots were allowed to record scores. At the end of a match they will decide which of them shall record the match. If it is only one bot left at the end of a match that bot will record the results, and if both are present the bot named Main recorded. A match ends when either the time limit of 101 minutes have been reached or both bots in a team are defeated.

To retrieve the results from the dedicated computer a bat file was used. This file copy the results from each client to a single location on a workstation, where then a python script was executed to merge the results into a single file. Merging the files made it possible to keep track of the results during the execution of the experiments. The merged file was later used for analysis.

4.2 Risks

Matches in the experiments were observed to determine if bugs surfaced since the bots ran for a longer time in the experiments than during the implementation phase. If a bug was noticed it was fixed and the experiment was restarted.

It exist variables that cannot be controlled which may have had some impact

on the bots performance in either a negative or a positive way. Since StarCraft is a non-deterministic game it will have some effect on the results, but there are other variables that could have a greater impact. For example there is a possibility that not all bugs were caught during observation making the cooperative ability not working as intended. Another problem could be that one of the bots do not reset properly after a match, which could cause the bot to play at decreased performance during one or more matches.

Starting positions in StarCraft are selected at random which could put a bot at a disadvantage due to how the bot place buildings. In StarCraft it is favorable to place certain buildings near the chokepoint and some buildings at the back of the base. The bots will however always place buildings in the same pattern, independent on the starting positions.

Chapter 5

Results and Analysis

An increase in memory usage was noticed during the experiments which suggests that memory leaks occurred. The leaks could be either from the bot or BWAPI. It is impossible to interpret if the leaks are affecting the results since a lot of variables that are out of control exist, such as random starting locations, non-deterministic game, unnoticed bugs and so on.

It was noticed that approximately after 70 matches the results had a tendency to become unstable. Unstable results mean that anomalies occurred, such as the same team wins multiple matches in a row. It was not coherent for all experiments but the earliest tendencies occurred at around 70 matches.

The reason for the unstable results is unknown and it is impossible to say what or which variables caused the results to become unstable. It is however possible to conclude that the impact of these variables was too severe after 70 matches and the matches after 70 were discarded in the results.

In Figure 5.1 the win percentages of both teams are shown. Team A has various cooperative behavior enabled during each experiment. The experiment's name states which category or categories that are enabled. The draw percentage of each experiment is also shown.

The Assist experiment in Figure 5.1 shows that team A and B have a similar win percentage of around 47%. The difference between team A's and team B's win percentage is the smallest compared to the other experiments. Team B achieved the highest win percentage in this experiment. The draw percentage of the experiment is below 5% which is in the lower range.

Team B in the Defend experiment is similar to its percentage in the Assist experiment, although slightly below which makes it the second highest win percentage for team B. Team A's win percentage ends at 40%. The draw percentage of the experiment is about 11% which is in the middle range.

The Attack experiment has the highest draw percentage at around 26% and team A's win percentage is about 42% which is slightly above the result in the Defend experiment. The high draw percentage puts team B's win percentage at above 30%, resulting in a difference in win percentage between team A and team B of over 10%.

AssistDefend was the only experiment where team A's win percentage ends

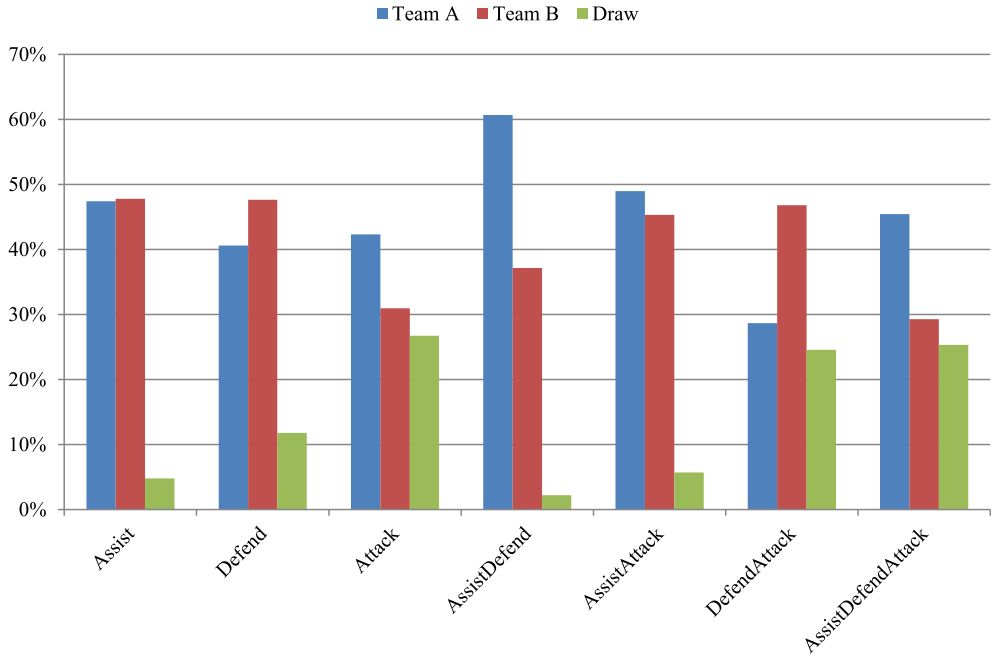

Figure 5.1: Statistical mean win percentage of both teams and the statistical mean draw percentage of each experiment. Presented in the same order they were conducted.

above 50%, to be specific 60%. The large win percentage for team A makes team B's win percentage drop down to approximately 37%, which created the largest difference compared to the other experiments. The experiment also has the lowest draw percentage at around 2%.

The AssistAttack experiment has a larger draw percentage than the Assist-Defend experiment with almost 4% but can still be considered to be in the lower range. The percentage of team A and B are both between 45% and 50% where team A's percentage is slightly below 50%, which is the second highest win percentage for team A. Team B's percentage is about 45%.

The DefendAttack experiment has the lowest win percentage for team A at around 28%. The win percentage for team B is about 46% which is the third highest percentage for team B. This creates a difference at around 18% of favor for team B. The experiment has a high draw percentage of slightly below 25%.

The last experiment, AssistDefendAttack, is a combination of all categories. Figure 5.1 shows a win percentage for team A at around 45% and team B's win percentage is slightly below 30%. The draw percentage for this experiment is in the higher range with 25%.

During observation it was noticed that units got stuck at chokepoints. This

occurred when the teammates wanted to enter and exit a region through a choke-point at the same time when for example the bots are scouting, defending each other or attacking together. This could contribute to the high draw percentages seen in some of the experiments.

Experiment	Team A	Team B	Difference	P-Value
Assist	47.41%	47.80%	-0.39%	0.482
Defend	40.61%	47.63%	-7.02%	0.202
Attack	42.31%	30.95%	11.36%	0.083
AssistDefend	60.68%	37.13%	23.55%	0.003
AssistAttack	48.98%	45.32%	3.66%	0.333
AttackDefend	28.65%	46.79%	-18.14%	0.014
AssistDefendAttack	45.43%	29.27%	16.17%	0.025
Average	45.94%	39.63%	6.31%	

Table 5.1: Statistical mean of the win percentage for both team A and B is shown and the difference between them. The average of team A and B and draw percentage is calculated and shown in the last row. Cells marked green are statistically significant (below 0.1) and cells marked red are not (above 0.1).

5.0.1 Analysis of RQs

As each category has multiple abilities implemented, and there is only a choice of which category is enabled and not which specific ability that is active or not. This could in theory interfere and affect the results in both a negative or positive way, making it impossible to analyze which specific ability that had the largest effect on the results.

RQ 1.1: Do cooperative assist actions lead to increased performance? Table 5.1 shows that the difference for the Assist experiment is almost zero percent. The experiment is not statistically significant therefore the null hypothesis cannot be rejected. The performance is not increased with Assist abilities.

RQ 1.2: Do cooperative defend actions lead to increased performance? The null hypothesis for the Defend experiment cannot be rejected due to the difference is not statistically significant at -7.02%. The performance is not increased with Defend abilities.

RQ 1.3: Do cooperative attack actions lead to increased performance? The Attack experiment's difference of 11.36% is statistically significant which makes it possible to reject the null hypothesis and accept the alternative hypothesis. The performance is therefore increased by cooperative Attack abilities.

RQ 1.4: Do combining assist and defend actions lead to increased performance? The difference at 23.55% for the AssistDefend experiment makes

it statistically significant and the null hypothesis can be rejected, thus the alternative hypothesis is accepted. The performance increased by combining Assist and Defend abilities.

RQ 1.5: Do combining assist and attack actions lead to increased performance? The last non-significant experiment was the AssistAttack experiment. The null hypothesis can not in this case be rejected. The performance is not increased by combining Assist and Attack abilities.

RQ 1.6: Do combining defend and attack actions lead to increased performance? The only experiment to be statistically significant and have a negative difference is the DefendAttack experiment. The alternative hypothesis is not accepted as the performance was not increased. The performance decreased by combining Defend and Attack abilities.

RQ 1.7: Do combining assist, defend and attack actions lead to increased performance? Combining all categories for the AssistDefendAttack experiment resulted in a positive difference of 16.17%, which is statistically significant and thus able to reject the null hypothesis. The alternative hypothesis is therefore accepted. The performance increased by combining Assist, Defend and Attack abilities.

Chapter 6

Discussion

The answer to RQ 1 is not straightforward. For three of the sub-RQs there was an increase of performance, and for only one sub-RQ the performance was decreased. In three cases there was no difference in performance at all. This suggests that cooperative behavior can in some cases increase performance, while in other cases there are no difference in performance or performance is decreased.

The small increase in performance in the **Assist experiment** could be because its abilities work best when the bot's and teammate's units are near each other. For example the teammate usually repaired or healed its units before the bot's agents would reach them. The experiment did not reject the null hypothesis and thus more research is needed for these abilities to determine how they affect the performance.

The Defend category makes a bot abandon their attack and run to defend its teammate if the teammate is under attack. It makes it possible for the enemy bot to attack the units and destroy them one by one as they move towards the teammate's base. By abandoning their attack the enemy could be given a chance to rebuild and be able to retaliate before the bots have successfully defended the base under attack. This could be a reason for the lower win percentage of team A for the **Defend experiment**. However, as the experiment was unable to reject the null hypothesis it is uncertain what effect the abilities had on the performance and more research is therefore needed.

The **Attack experiment**'s increased difference and higher draw percentage suggests that the bots were able to destroy their enemies but sometimes unable to finish them, probably due to units getting stuck at chokepoints more frequently compared to the other experiments. Force concentration could be the reason for the increased performance, but also the main reason for units getting stuck.

The results in Table 5.1 show that both the Assist and Defend experiment are not statically significant and both have a negative difference, but the **AssistDefend experiment** has the highest positive difference. If both bots attack the same enemy and the teammate is attacked, the bot could stay longer at the enemy base to assist the siege tanks before helping its teammate. This could let the teammate leave first which would reduce the amount of units passing through the chokepoint and thus making the teammate reach its base faster. By staying

longer at the enemy base the bot prevents enemy units from following the teammate's units and destroy them one by one. Also the bot could destroy enough units and buildings to stall the enemy before leaving to help its teammate. Afterwards it is not certain that they would attack the same enemy again thus be able to prevent the other enemy to flank the cooperative bots' bases while one of the bot finishes the stalled enemy.

A similar scenario occurs in the **AssistAttack experiment** as in the AssistDefend experiment. Although the absence of the Defend abilities cause them to always protect each other while the other go to defend its base. As one bot will stay at the enemy base it is a higher chance that the enemy is destroyed, but by not helping the teammate could result in that the teammate is also destroyed. This makes the match become one vs one. As the bots have the same implementation no bot should have an advantage over the other and thus resulting in an even win percentage for both teams. However, as the experiment was unable to reject the null hypothesis, more research is needed to determine the performance of the combined categories.

The **DefendAttack experiment** has the largest negative difference. It could be because both categories cause an increase in units passing through the same chokepoints. In some cases it was observed that the bots were unable to reach the attacked base in time, probably because of it taking longer time for units to move through chokepoints. It was observed that if the teammate was destroyed its units could be passing through a chokepoint which cause the teammate's units to become stationary. This makes the bot's units getting stuck if they were at the wrong side. This causes the bot to be unable to use its units at all for the rest of the match, which could be the main reason for the largest negative difference.

The **AssistDefendAttack experiment** is similar to the AssistDefend and AssistAttack experiments. The situations that occurred during those experiments were observed in this experiment as well. The draw percentage is much higher in this experiment compared to the other two. A reason for the higher draw percentage could be that the bots were unable to finish off the enemies as in the Attack experiment. The higher draw percentage reduces the overall difference as it affects both teams' win percentages.

There exists no other research of the type of cooperative behavior used in in this thesis, which makes it impossible to compare the results against other data. However, as the analyzed research showed, cooperative behavior had a positive effect on their result [1, 10, 13, 17, 23], which is also the case, depending on the situation, in this thesis.

Chapter 7

Conclusions and Future work

This thesis has shown that cooperative behavior can increase performance for two teaming RTS bots against a non-cooperative team with two bots. The results have been analyzed with a t-test for statistical significance which proved that some experiments were able to reject the null hypothesis and others able to accept the alternative hypothesis: cooperative behavior increases performance. However, the performance could either be increased or decreased depending on the situation.

The Assist, Defend and AssistAttack experiments were unable to reject the null hypothesis and are considered as negative results. It is important to note that because the results are heavily dependent on the implementations, it could be possible to get increased performance with Assist, Defend or AssistAttack abilities if it were implemented in different ways. It is also possible that the performance would decrease if they had different implementations. Therefore more research is needed to define which cooperative behaviors that increase performance and which ones that decrease performance.

The Attack, AssistDefend and AssistDefendAttack experiments showed that cooperative behavior can increase performance and the DefendAttack experiment showed that cooperative behavior also can decrease performance. The Attack, AssistDefend, AssistDefendAttack and DefendAttack experiments are considered to be positive results. The DefendAttack experiment is considered to be a positive result as it was able to reject the null hypothesis, although it could not accept the alternative hypothesis. The results therefore show that cooperative behavior affects performance depending on the situation.

In the future it would be interesting to implement new logic for the bots that make it possible for the bots to resolve units that got stuck at chokepoints. Another future improvement would be to analyze a match that ends in a draw. The analysis would for example compare the amount of units and buildings for both teams and if one team has significantly lower units and buildings the other team could be declared as the winners.

Using categories was a bad design decision, and which created a limitation for the experiments. They made it impossible to determine what effect an ability had on the performance. In the future it would be interesting to focus on single abilities to determine their effect on the performance in more detail. Single

abilities could also be tested in more focused experiments, such as small scaled combat. This could help the cooperative ability to be tested individually and more extensively. This could also be positive in that fewer variables that are out of control exist.

It would be interesting to implement more cooperative awareness, for example the bot could know more about the teammate's ongoing attack. The bot could also analyze the attack and determine if the teammate needs help or not. The bot could choose to attack the other enemy to prevent that enemy to flank the cooperative bots' bases.

In all categories behavior for micromanagement, have been implemented, but it could also be implemented for macromanagement. This would create cooperative behaviors at a higher level of gameplay. Cooperation for macromanagement could for example be to utilize the strengths and weaknesses of each other's strategies. This level of cooperation could be made more adaptive, similar to how Jansen made a bot that could determine a human player's strategy and complement it [13].

References

[1] Aswin Thomas Abraham and Kevin McGee. Ai for dynamic team-mate adaptation in games. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium*, pages 419–426. IEEE, 2010.

[2] AutoHotkey. URL `http://www.autohotkey.com/`. Last accessed: 2015-09-05.

[3] Blizzard Entertainment. Starcraft, . URL `http://us.blizzard.com/en-us/games/sc/`. Last accessed: 2015-09-06.

[4] Blizzard Entertainment. Starcraft compendium, . URL `http://classic.battle.net/scc/GS/`. Last accessed: 2015-09-06.

[5] Blizzard Entertainment. Starcraft compendium on siege tanks, . URL `http://classic.battle.net/scc/terran/ut.shtml/`. Last accessed: 2015-09-06.

[6] Michael Buro and Timothy Furtak. Rts games and real-time ai research. In *Proceedings of the Behavior Representation in Modeling and Simulation Conference (BRIMS)*, volume 6370, 2004.

[7] BWAPI. URL `http://bwapi.github.io`. Last accessed: 2015-09-05.

[8] BWAPI. Academics, Juni 2015. URL `http://github.com/bwapi/bwapi/wiki/Academics`. Last accessed: 2015-09-05.

[9] David Carmel and Shaul Markovitch. *Learning models of opponent's strategy in game playing*. Technion-Israel Institute of Technology. Laboratory for Parallel Computing Research, 1993.

[10] Maria Cutumisu and Duane Szafron. An architecture for game behavior ai: Behavior multi-queues. In *AIIDE*, 2009.

[11] Edmund H Durfee. Distributed problem solving and planning. In *Multi-agent systems and applications*, pages 118–149. Springer, 2001.

[12] Johan Hagelbäck. Potential-field based navigation in starcraft. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference*, pages 388–393. IEEE, 2012.

[13] TJA Jansen. Player adaptive cooperative artificial intelligence for rts games. Bachelor's thesis, Universiteit Maastricht, 2007.

[14] Liquipedia. Starcraft: The game, October 2014. URL `http://wiki.teamliquid.net/starcraft/StarCraft`. Last accessed: 2015-09-05.

[15] Matteus M Magnusson and Suresh K Balsasubramaniyan. A communicating and controllable teammate bot for rts games. Master's thesis, School of Computing, Blekinge Institute of Technology, 2012.

[16] Kevin McGee and Aswin Thomas Abraham. Real-time team-mate ai in games: A definition, survey, & critique. In *proceedings of the Fifth International Conference on the Foundations of Digital Games*, pages 124–131. ACM, 2010.

[17] Truong-Huy Dinh Nguyen, David Hsu, Wee-Sun Lee, Tze-Yun Leong, Leslie Pack Kaelbling, Tomas Lozano-Perez, and Andrew Haydn Grant. Capir: Collaborative action planning with intention recognition. *arXiv preprint arXiv:1206.5928*, 2012.

[18] Patrick Howell O'Neill. 'starcraft: Brood war' proves the best esports are timeless, January 2014. URL `http://www.dailydot.com/esports/starcraft-brood-war-starleague-esports-survive/`. Last accessed: 2015-09-06.

[19] Glen Robertson and Ian Watson. A review of real-time strategy game ai. *AI Magazine*, 35(4):75–204, 2014.

[20] H Jaap van den Herik, HHLM Donkers, and Pieter HM Spronck. Opponent modelling and commercial games. In *Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games (CIG'05)*, pages 15–25, 2005.

[21] Wikia. Starcraft. URL `http://starcraft.wikia.com/wiki/StarCraft`. Last accessed: 2015-09-06.

[22] Minjie Zhang, Quan Bai, Fenghui Ren, and John Fulcher. Collaborative agents for complex problems solving. In *Computational Intelligence*, pages 361–399. Springer, 2009.

[23] Pucheng Zhou and Huiyan Shen. Multi-agent cooperation by reinforcement learning with teammate modeling and reward allotment. In *2011 Eighth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, 2011.