Master Thesis
Computer Science
Thesis no: MCS-2012-97
09 2012

A Communicating and Controllable Teammate Bot for RTS Games

Matteus M. Magnusson
Suresh K. Balsasubramaniyan

School of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona
Sweden

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

Contact Information:
Authors:
Matteus M. Magnusson
E-mail: matteus.magnusson@gmail.com

Suresh K. Balsasubramaniyan
E-mail: suresh.draco@gmail.com

University advisor:
Dr. Johan Hagelbäck
School of Computing

School of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona
Sweden

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

Abstract

Communication in team games between human player is important, but has been disregarded in teammate bots for Real-Time Strategy (RTS) games. Control over the teammate bots have existed for a while in other genres than RTS and if implemented correctly it adds another dimension to the game.

In this study we investigate whether players think it is more fun to play with an RTS bot that communicates its intentions and reasons and is controllable by the human player. But also what features are liked, disliked, missed to provide guidelines to future researchers and companies.

For this we create a StarCraft RTS bot with communication and control abilities. The experiment consists of four scenarios, turning on/off communication and control, to conclude what players think is fun.

All testers agreed on communication being important and more fun to play with. Beginners did, however, not like the control feature as they already had enough on their mind whereas experienced players preferred having some control over the bot.

We conclude that communication is an important role in team games, including RTS games. More work needs to be done how to integrate control so that beginners do not feel overwhelmed and at the same time experienced players do not feel restrained by too simple control commands.

Keywords: AI, RTS, StarCraft, communication.

Acknowledgements

We would like to thank our supervisor Johan Hagelbäck for his assistance who always kept a positive attitude even when we were delayed. For his well-documented bot, BTHAI, which helped us learn its system quickly.

Thanks to all members in the BWAPI freenode chat who always were there for help if one needed.

Thanks TeamLiquid.net, Battle.net and their members who came with suggestions on what they wanted in an teammate bot.

Thanks Blizzard for creating great games and letting researchers use BWAPI to create bots in StarCraft.

Thanks Day[9] for all your episodes on StarCraft strategy analysis which made me, Magnusson, a lot better analyzer.

Thanks to everyone around us that have been supportive when we were facing obstacles.

Contents

List of Figures

Listings

Chapter 1

Introduction

Have you ever wanted the bot[1] you played with in an RTS to communicate or control it to test that strategy that you want to play with your friend when s/he is not available? Then you can continue reading as this thesis is about that, a communicating bot that tells its intentions and reasons to you, the player, and allows you to control the bot using simple commands like, attack, expand, drop.

Although there exist a few new (released less than 5 years ago) Real-Time Strategy (RTS) games, none of these lets you control your bot in a simple way and at the same time communicates its intentions and reasons to you, at least not to our knowledge. This includes research, where we have yet to find a single subject on communication between human player and bot.

Our focus lies in evaluating if it is worthwhile to do further research in this unexplored area, both for the purpose of researchers and for companies to help them decide if they shall invest money in a collaborative teammate bot for RTS games.

To familiarize you with RTS games, StarCraft, and collaborative game bots a brief description on all subjects is given below. Once you have basic knowledge of the subjects we will give a thorough background what others have done in closely related subjects. Then a detailed system description, why it was implement this way, and how it was evaluated (methods). Next the results of our experiments and our conclusions are presented; and finally we present new subjects for future work.

1.1 RTS

An RTS game is a computer game that heavily relies on strategy and fast execution. An analogy would be to play chess where both you and your opponent can move pieces at the same time, you will only see your own pieces and a tile beside them—meaning you do not know the location of the opponent's pieces or when s/he makes a move—on top of that you have to think of a strategy fast before your opponent gets a lead.

[1] A bot is a computer player that you either can play with or against

In reality, RTS games resembles more a battle field than a game of chess—you control the army (both units and structures) and what to spend the resources on. At the start of the game you have a single base and your goal is to build up the technology to create powerful units by collecting resources. Often you cannot go straight for the best technology as it takes time and resources to build up all necessary technology and if you only focus on technology you will not have any units for defense if the enemy comes.

In most, if not all, RTS games the player can choose whether s/he wants to play a campaign—usually accompanied with a story—or a simple skirmish game. A skirmish game is in its simplest form a quick battle where both you and your enemy or enemies start with the same amount of resources and structures. Many variants of skirmish games can be played, this thesis will focus on team skirmishes, more specific teams of 2 players versus 2 players. One human collaborating with our proposed teammate bot in one team, and two default bots[2] in the other team. How this will work is explained throughout the thesis.

There is no real standard for RTS games, they all work a bit different from each other. Instead of covering all different types of RTS games a short description how StarCraft: Brood War works will be given to make the reader familiar with the game before talking about more specific problems and solutions. We choose StarCraft: Brood War because it was the most well-known and widespread RTS game of all time until maybe the introduction of StarCraft 2. More reasons why we chose StarCraft: Brood War can be found in section 2.3

1.1.1 StarCraft

The bot is implemented in an RTS game: StarCraft, more specifically StarCraft: Brood War—an expansion to StarCraft that introduced new units. Henceforth StarCraft: Brood War will be referenced as StarCraft for simplicity. The StarCraft description below was gathered both from the StarCraft manual[7], the inofficial StarCraft wiki[24], and our observations in gameplay. E.g. to test whether Protoss shields regenerated in battle, we created a single player game where a probe was sent to attack another probe and we observed that shields regenerates in battle since the probe's shield points regenerated (increased). Figure 1.1 shows a print screen of a Terran base from one of the campaign maps.

Gameplay

StarCraft has three races—Terran, Protoss, and Zerg—which the player can choose from. Each of these races have their own unique types of units and varying play styles.

[2] A default bot is the bot that already exists in the game, usually when the player want to play against it during skirmish matches.

Figure 1.1: Screenshot of StarCraft

Commonalities between races All races have worker units that collect minerals and gas—game resources used to build structures, units and upgrades. Minerals and gas are usually located in clusters of several mineral fields and occasionally a Vespene geyser (gas). The player always starts at a base with both mineral fields and a Vespene geyser. To mine minerals and gas, workers go to the fields and then return to the closest main structure of the player. Because of the travel distance you usually want to build new main structures near resource clusters.

Mineral fields have a fixed number of resources they can yield; when all have been depleted the mineral field disappears. A Vespene geyser on the other hand does not disappear after depletion, instead each worker now brings back 2 units of gas each turn, instead of 8. To be able to mine gas in the first place a structure needs to be placed on a Vespene geyser. Only one worker can simultaneously collect minerals from a mineral field, or gas from a gas depot, other workers are automatically queued to start mining when the first worker brings back the resources.

StarCraft limits the number of units one can have to 200 supply[3]. Each unit occupies some supply, ranging from a half supply (Zerglings) to six supplies (Carriers), armed nuclear silos take up eight supplies. In general, the better the unit the more supply it occupies. The player starts with 9 to 10 max supply (depending on the race). To increase the max supply a special structure (unit

[3]Called Psi for Protoss, and Control for Zerg which makes more sense for these races. Supply is, however, the most common used jargon for all races.

for Zerg) can be built; main bases also generate supply, 10 for Terran Command Center, 9 for Protoss Nexus, and 1 for Zerg Hatchery.

Constructing Structures When Terran's worker unit, SCV, constructs a structure it becomes occupied and cannot do anything else; the SCV can however be targeted, and can stop building and let another SCV take over—e.g. when an enemy starts attacking the SCV.

Zerg's worker unit, Drone, will morph itself into a structure. The drone will disappear once the morphing stage begins, but if the player cancels the building progress the drone will reappear. Once the morph completes the drone is forever lost. Zerg structures need to be placed on creep which Hatcheries and colonies help to spread; exceptions are Hatcheries and Extractors (for Vespene Geyser).

Protoss's worker unit, Probe, need to construct structures in a psionic power grid, a circle grid generated by Pylon structures; exceptions are Nexuses, Pylons, and Refineries (for Vespene geyser). If a structure becomes unpowered it will stop functioning until a nearby Pylon is reconstructed to create a new psionic power grid. Once a structure has been planted, it will construct itself leaving the Probe free for other tasks.

Constructing Units Terran and Protoss function in basically the same way: they both have separate unit producing structures for different kinds of units, e.g. Barracks/Gateway for basic ground units, Starport/Stargate for flying units, and so forth. All units, however, cannot be built from these structures directly, some units require another stand-alone structure, like Academy for Firebats and Medics. The main structure (Command Center and Nexus) can only build workers. Only one unit can be built simultaneously from these structures, if one wants a huge production of units, large amounts of unit producing structures are needed.

Zerg on the other hand only has one unit producing structure, the Hatchery, that pops out larvae in a certain time interval—one Hatchery can maximum have three larvae. These larvae can be morphed into the unit of choice. To build other units than Drones and Overlords the player must build the structure needed for the unit, e.g. building a Spawning Pool to be able to morph Zerglings from a larvae, in a similar way Academy is needed for building Firebats from a Barracks. This makes Zerg good for creating lots of better units with a minimum extra cost, because it only needs one extra building and some Hatcheries. E.g. if Zerg has 3 Hatcheries it can save its larva and when the new building is complete it can directly start producing 9 of the new units. Whereas Terran needs to build 9 Barracks, Factories, or Starports (depending on the unit) to have the same rate of production.

While it seems most obvious to build "the best" units at the beginning, this is not possible—although there is no such thing as best unit as all units are good depending on the situation, powerful unit might be a better wording. A powerful

unit require certain structures, that in turn require other structures, i.e. unit and structure requirements can be seen as an acyclic graph where the best structures and units are at the ends. For example, to build a Carrier one requires a Fleet Beacon (for the Carrier tech) and a Stargate (to build the unit), the Stargate then requires a Cybernetics core, the Cybernetics core requires a Gateway and a Gateway requires a Nexus. As you might conclude, you need a lot of resources and time to be able to build a Carrier; if you go straight for Carriers your enemy can just walk into the base with any number of units and kill the entire base.

Unit Health The unit health for each race works a bit different. Terran units do not generate health. Mechanical units (both ground and air) and structures can be repaired by an SCV at the cost of minerals and gas, depending on what resources were required to build the unit or structure; an SCV can also repair another SCV since they are vehicles. Biological units, such as marines and medics can only restore lost hit points (HP) by medics which have a healing ability that restores 2 HP per 1 energy; this ability can be cast every second. In addition medics have the ability to heal Protoss and Zerg biological units—which makes Terran a good team race.

Protoss units are a bit special, they have two health bars, one for health the other for shields. These are usually divided by half, meaning units have equal amount of health as shields, an exception is Archons which have 10 HP and 350 shield points (SP). Shields regenerate much faster than zerg's regeneration ability, including in battle, but always takes full damage from all damage types. Terran Science Vessels can drain all the shield instantly through its EMP ability. Shields can, however, be fully recharged by a protoss support structure, shield battery, which converts its energy to shields, just as medics can heal biological units.

All Zerg units and structures automatically regenerates health at a very slow rate, even during battle, but because it regenerates slowly units usually only heal 1 or 2 HP during a shorter battle.

1.2 Teammates

A teammate is a player, either human or bot, that helps the team towards a common goal. Teammate bots often prioritize the human player because it is human's experience, thus the bot acts as a helper to the player and team.

1.2.1 Human teammate

You can divide human teams into essentially two groups: where players more or less cooperate with each other and teams whose players "do their own thing" not caring about what the others do. Although the reality is more a gray scale, often

players do not only cooperate or only do their own thing, but instead it varies throughout the gameplay.

Humans are good at forming complex strategies, or at least complex to program an AI to understand what to do. They can come and aid their teammate intuitively without the teammate asking for help.

When playing together with other humans, communication is important. If verbal communication is possible, players can tell their intentions to the teammates—e.g. attacking, defending, moving—other important game changes—e.g. enemy missing, getting shot, spotting an enemy—or just requesting help. When the player cannot communicate verbally, text messages through the games are often possible, but this can disrupt gameplay since actions are often disabled while typing (depending on the game). Short text messages are often sent instead to convey the entire message; e.g. "attack" instead of "I'm going to attack the enemy base from this side, join me."

StarCraft In StarCraft, players that know each other usually decide what strategy to use before starting the game, whereas if the players do not know each other they can discuss a common strategy to use, announce their strategies, or both. If verbal communication is possible, players like to talk about tactics during the game, they like to convey their intentions to the teammate player.

1.2.2 AI teammate

A general teammate bot tries to replace a human player. In some games they are used to replace an ideal human player. This bot usually tries to figure out what the player is doing and either figure out a strategy that works with this or trying to deduct what the player want bot to do. When a bot works, it feels like a human player, especially if the game provides some sort of communication to the player. If you, however, have played a game where the bot breaks, you can feel that it is mechanical.

In other games teammate bots only have a minor role; in Modern Warfare 2[23], a first-person shooter (FPS), you have teammate bots helping you in certain areas, these bots do not do much, but are there for the player. They do, however, "cooperate" with the player—we say "cooperate" because the bots usually only follow the player while making sure behaving accordingly such as avoiding places between the player and his/her target.

StarCraft The standard AI in StarCraft probably does not have any specific AI when it plays a cooperative game with a human player; as it does not interact with the player and acts on its own, neither helping the player nor taking help from the player.

Chapter 2

Background

While RTS games has been around since the 80s[2, 18], only a handful of scientific articles can be found on teammate bots for RTS games, and current big RTS titles have yet to implement a good teammate bot. In general little research has been done for teammate bots throughout all genres; RTS researchers have focused on enemy bots to either create a fun opponent[21] or to create the best bot to compete in RTS bot tournaments, such as AIIDE's StarCraft AI Competition[15] and CIG's StarCraft AI Competition[14].

Next we will begin covering related research topics, continuing with teammate bots in games across all genres and what current RTS games lack.

2.1 Research

First we will describe the definition of real-time teammate bots, then covering teammate bots across all genres asking what guidelines applies to RTS games; continuing with communication between humans and bots; and ending with RTS enemy bots and asking, what implementation strategies exist, how a good bot shall play.

2.1.1 Teammate bots

As mentioned, little research has been done in the area of teammate bots, especially for RTS games. To our knowledge there exist one paper on teammate bots[27] which brings up communication; in their survey, McGee and Abraham, presents their definition of real-time teammate, which their survey is limited to. A summary of their definition reads; a real-time teammate bot

1. works together with team players while taking into account the state, needs, behavior, goals, plans and intentions of these players;
2. uses coordinated behaviors or decision-making...
3. ...that aligns with the team goals;

4. where these coordinated behaviors or decision-making includes player-related uncertainty requiring inference, reasoning, classification, calculation, or another method; and

5. whenever possible, prioritizing the player experience.

We will use the same definition and follow these five points for our proposed bot.

In their survey[27] McGee and Abraham noticed that, although human player participation and engagement are one key functionality of a game[32], often the player's preferences are neglected and the bot(s) behave what it thinks is the best for either just itself or both the player and itself; while the second option might sound as if it prioritizes the player, it does not, it steers the player into how to play rather than the player steering the bot. When the bot prioritizes the player, some challenges arise; how to create priority rules that do their job correctly[27], i.e. the bot has to know, or use a qualified guess, what the player wants. This is no easy task, probably impossible in the near future; humans have a hard time understanding each others intentions, why think AIs (that humans have created) understands us better[28] without even asking?.

Communication McGee and Abraham points out the lack of research on communication between human players and bots, "This survey suggests that there are also some aspects of real-time team-mate AI where there seems to be little or no work: ..., and communication."[27] Meaning "little or no work" spans through multiple topics and multiple genres; we have yet to find any paper that talks about communication between human players and bots, for any genre. Games in other genres has, however, implemented some sort of communication between human players and bots, this is covered in section 2.2.1

Teammate bots across all genres Abraham and McGee created a teammate bot for a simple game: Capture the gunner[1]. The goal of this game is to capture the gunner by touching him from both sides while not being shot. The game required cooperation with their bot because selfish players never passed the first level. Players had great responsibility over the teammate, because of this they found that even if the bot died by the gunner players never felt it was unfair; in fact, some players felt that it was partly their fault if the bot died.

Player classification To beat an enemy in an RTS game, as a team or a single player, you need a strategy that exploits the opponent's weaknesses while eliminating your own (team's) weaknesses. Both tactics and strategy needs to be good to beat an enemy—although when human beginners play against each other one can win with either just good strategy or good tactics.

To find the teammate's and opponents' weaknesses one can make use of simple classification rules, or make it more advanced and use a model. These weaknesses

are used to either exploit the opponent's weaknesses or complementing the teammate's weaknesses; these techniques can be used for both purposes, although information gathering and bot action will, however, be different.

Teammate modeling Jansen has created a player model using opponent-modeling for his RTS bot[25]. The bot actions are calculated from a decision tree computed with supervised learning and neural networks. The goal of the teammate bot is to 1) match the number of units and structures the player has, this includes the unit/structure type, e.g. aggressive, defensive; 2) be able to deduct when the player is under attack from the model; 3) what action the bot shall take next using the decision tree from the learning strategies; 4) for the bot to find when the player has either a hole in their defense or attack; and 5) when the player is switching from defensive to offensive mode, or vice versa. He found that his bot could mimic the player; but two problems were identified, the bot could not identify which of the actions were the best one and in addition some players does not always know or do the best action thus the learned actions can use bad strategies.

In their paper[31] Pucheng and Huiyan uses Q-learning, teammate modeling, and reward allotment for their teammate bot to faster learn which actions leads to a successful goal. Their experiment tested this new learning technique against traditional Q-learning where the bot does not take the teammate into account, and teammate Q-learning where the teammate is taken into account but the reward is not split between bots.

For actually creating a player modeling system, Houlette presents a methodology for how to implement a player model in code[22]. He talks about what a player model is, what it contains, and what it is good at and used for. In addition he gives an simple code example and a description when to update the model and two possible update implementations.

Opponent modeling Kabanza et. al has implemented an RTS bot, HICOR (Hostile Intent, Capability and Opportunity Recognizer). HICOR can, as specified by the name, infer the opponent's intention (i.e. its plan) and use these to analyze the enemy capabilities and opportunities for the bot. Put easy it can infer what build order the enemy is using, what tactic it is using and where it will attack and use this information to its advantage. The underlying system uses Hidden Markov Model to infer the enemy plan.

For recognizing the behavior of the opponent, i.e. aggressive/defensive and what type of aggressive/defensive behavior Schadd et. al uses a hierarchical approach model with two classifiers for their RTS bot. A top level classifier uses fuzzy models to classify the opponent as aggressive vs. defensive, and a bottom level classifier for the type of aggressive/defensive behavior, e.g. the opponent uses mainly tanks or ships when aggressive, or techs when being defensive.

Synnaeve and Bessière RTS bot uses Bayesian networks to model enemy opening strategies[38]. The bot learns by watching replays manually label with the strategy of the players.

2.2 Teammate bot in games

Teammate bots have been around for quite a while in sports game, such as FIFA[17], but have just started to make a breakthrough in other genres. In most games[23, 19, 39] the teammate bots cannot be replaced by another player as they either are a part of the story, and thus might not be around all the time, die, or have something else happen to them. In games that are meant be played cooperatively with friends (or strangers), these can be replaced with bots[12, 13].

2.2.1 Communication

Communication has been implemented across several games and genres, most noteworthy are genres where you play as one character, such as FPS games, third-person shooter (TPS) games. In these games some bots communicate you, warning when they spot enemies, get shot, or comes with tips when the player is stuck.

Mass effect[5], a TPS game, does this in its game by letting the bots tell the player when for example enemies are sighted, and an area is cleared from enemies. Mass Effect 3, goes beyond regular communication and lets players on Xbox 360 to control the bots through voice commands, like a squad leader. This creates a better flow in the game since players do not have to open the action screen (which pauses the game) as often.

2.2.2 Controllable bots

Today there exists quite a few games that implements the possibility for the player to actively control teammate bots (if the player wants to). We cannot possibly find and go through each game that lets you control its teammate bots, but we will mention a few to show that the feature can be found in games.

Mass Effect[5] does this by having the player the possibility to decide where the bots shall move for cover and hold that position, retreat for cover, even order the usage of certain abilities on target enemies. Rainbow Six Vegas 2[39] and Brother in Arms: Road to Hill 30[19] lets you control its teammate bots much like Mass Effect.

2.2.3 RTS games

Today, only one RTS game, that we know of, allows to communicate with and control your teammate bot, this game is Red Alert 3[16]. Before describing Red Alert 3, however, a description is given of how teammate bots in RTS games commonly works. These teammate bots acts more or less (depending on the game) on its own, i.e. it does not really collaborate with the player; some bots might try to complement the player's behavior but does not ask if this is the preferred choice for the player. Because commercial games are closed source we do not know to what extent the bot complements the player's behavior, or if they are taking the player into account at all.

The bot in the first StarCraft[6] installment acts entirely on its own, and it does not feel as it behaves differently when playing together with it. In WarCraft 3: The Frozen Throne[8] the bot reacts to the player coming to aid if s/he is under attack, and communicates its attack position—by pinging on the minimap—to the player when moving out to attack a target. Much like WarCraft 3, the bot in StarCraft 2: Wings of Liberty[1][9] aids the player when s/he is under attack, although it does not ping the minimap when it attacks. In Age of Empires 3[36] the bot acts almost entirely on its own; it can, however, request resources from the player and give the player hints.

Red Alert 3[16] on the other hand has the most advanced teammate bot. The game's campaign mode is played cooperatively with either another human player or bot. The bot can be given simple commands: move to specified position or strike a target; although these have some restrictions as the bot needs to to have the free units to execute the commands. In special missions, the bot will have super weapons that the player will have full control over. Like WarCraft 3 and StarCraft 2, the bot comes to aid the player when it is under attack.

2.3 Why StarCraft?

Why choose StarCraft and not another RTS game? Other games or engines the bot can be implemented in is SpringRTS[35], which is an open RTS game engine and is by itself not a game and requires a game mod[2]—several game mods are currently availble. ORTS[29] is aimed for developers and researching, and finally, Wargus[40], a WarCraft II clone that allows for modifications and implementation of an AI. So why not choose one of these instead of StarCraft?

[1] First game in the StarCraft 2 trilogy.

[2] A game mod in this case is the set of rules, units, graphics, to create a new RTS game.

Carefully balanced Blizzard Entertainment released StarCraft: Brood War in 1998 and continued to patch it until beginning of 2009[3]. The other games have neither had the time nor the amount of players to carefully balance the game. One factor might be because StarCraft has become a huge E-sport in South Korea[3].

Easy to find experienced players Because StarCraft have been around for so long and is a commercial successful game it is easy to find experienced players to test the game. By using experienced players as testers, the players do not have to learn the game mechanics and can focus on evaluating the bot instead of the game.

Big community StarCraft has a big community, this makes it easy to find and ask people what functionality they would like to see in a teammate bot to gain more ideas, but also evaluate our ideas.

Extending an already existing bot We have the opportunity to extend an already existing bot, BTHAI[20], for BWAPI[10]. By extending a bot we can focus on making the bot a good teammate and not worry about all the other details; such as good path finding, building placement. Sometimes we, however, improve some already existing systems, e.g. build order, to meet our needs, but we do not have to build the entire system from scratch. In addition, BTHAI is developed by our supervisor, Hagelbäck, and we can therefore get fast help of the system if needed. While we have not searched for other bots to extended, we figured it would be hard to top the support we would get from BTHAI.

2.4 Bot strategy

While our focus lies on communication and conveying intentions, a bot still needs a decent strategy and tactics to win and be useful for the player. There does, however, not exist any specific research on what is a good cooperation strategy that prioritizes the player for RTS games. Instead we will rely on general one player strategies from "Day[9]"[30] ([9] is part of the name and not a citation), our own experience playing cooperative games, and by evaluating the bot throughout the development.

[3]No official date can be found, the only inofficial page we found mentioning the date was Wikipedia at: `http://en.wikipedia.org/wiki/StarCraft:BroodWar`, accessed 2012-09-13

Chapter 3

Problem description and problem statement

As has been stated, we have yet to find any work on communication with teammate bots for RTS games, to this point. The problem, or rather our hypothesis, is that we think a teammate bot that conveys its intentions and reasons, and can be somewhat controlled by the player to execute various tactics will give more diversity to the game, opening up for new game experiences.

3.1 Aims and objectives?

Create a teammate bot that communicates its intentions and reasons to the human player and where the control of the bot can be transfered to the player by letting the player send strategic and tactical commands.

- Extend Hagelbäck's BTHAI[20] to create a teammate bot that
  - Communicating: Sends messages to the player, conveying its intentions and reasons and warns about enemy attacks.
  - Control: Can receive tactical (attack, drop) and strategic (transition to next build order) commands from the player.
  - Prioritizes the player.
- Evaluate the bot by having human players playing our bot with and without communication, and controls. Evaluation will occur in three stages; first before the actual implementation asking players what features they would like to see in a teammate bot, then during the later stages of development to receive feedback on what features need more tweaking, and last to answer the research questions below.

3.2 Research questions

1. Which is most fun to play with, communicating bot, controllable bot, both communicating and controllable bot, or none?

1.1. Which features are most liked?

1.2. Which features are most disliked?

1.3. What features are missing?

Hypothesis

1. We think it will be most fun to play with both a communicating and controllable bot, then communicating, controllable bot, and last a bot with none of these features.

The most liked features will probably be the ability to control the bot and that it communicates its intentions and reasons to the player, mostly because the player wants the ability to control the player as he wants it to play and that the player wants to know what the bot is doing. Second to that will be that it feels as though it is programmed to be a good teammate. Disliked features will probably include minor bugs that will not get solved in time, other than that we are not certain how much feature we will have the time to implement; thus we cannot speculate which features the players are going to miss either.

3.3 Expected outcomes

Game companies can use this teammate bot as guidelines to create new fun game experience without having to put as much money into researching whether this would pay off; with an incorrect hypothesis (i.e. not fun) the companies will know what works and what does not. In general the bot and this paper will provide guidelines how to create, or not create, a teammate bot for RTS games, and what features are most liked, disliked, or missing (i.e. what players might want).

For researchers, this work will either prove that further research in this field will be worthwhile, if the hypothesis is correct; if not, we have at least provided some results what does not work and that this area will not be worthwhile to research. In addition for both cases it will provide guidelines for future researchers what to do, and not to do.

Chapter 4

System description

We begin by describing the main features, behaviors, and goals of our bot; these features have been selected by either what players wanted, what we think would be a good feature, or both and then evaluated. We then give a brief overview of the system and how the different components/classes work and communicate with each other to get a quick grasp how everything is connected.

After the overview we will give a detailed description of all the important features of the classes, giving motivations of our design choices. The design choices were selected by either research, common knowledge in the StarCraft community, what we think would be good, or a mix of any and then evaluated, first on ourselves and then, close to the end, on our test player to find weird and non-acceptable behaviors.

Configurable variables Fixed variables in BATS (BTH (Blekinge Tekniska Högskola) AI Teammate StarCraft), our proposed bot for this thesis, are configurable from a config.ini file. This design choice was made to easily tweak these variables in-game, instead of having to recompile, start the game, wait until the game state is present (necessary buildings have been built) and then see the results. The configuration file could be reloaded anytime in-game using */reload config* command. Throughout this chapter these will be presented with their current value, variables that are configurable will have a superscript of *attached to it, e.g. 150^* seconds. If a range has a superscript, e.g. $[0.0, 1.0]^*$, both 0.0 and 1.0 is configurable.

Terminology explained There are a couple of words that needs to be explained before digging into the details.

Region & choke point A typical map from StarCraft contains many regions, the regions are (often) big open spaces divided by choke points—narrow paths, e.g. paths moving up or down a cliff. A region could be fully isolated, this means it can only be reached by air—BATS does not work on these kind of maps, all regions must be connected in one way or the other for BATS to work properly. Regions and choke points are illustrated in figure 4.1—this is also the map used

for all experiments. Regions and choke points are automatically generated by BWTA[11] in BWAPI[10].

Squad Units grouped together with a common goal or behavior, such as Attack squad or Scout squad.

Teammate/enemy squad A virtual squad used to group close allied or enemy units together. BATS uses this information to deduct certain information from these players.

Expanding The time from when one wants to build another base (to mine minerals and gas from) until the base has completed.

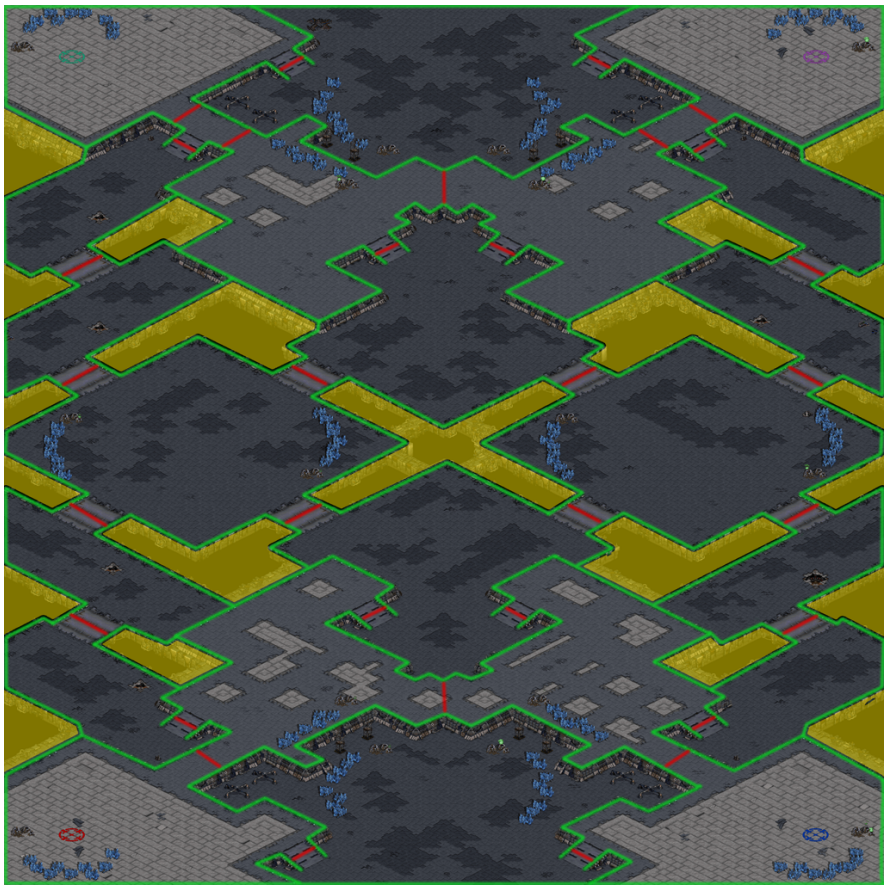

Figure 4.1: Regions and choke points. (—) Region borders; (—) choke points; and (▭) unwalkable terrain.

4.1 Behavior and system overview

In their article[1] Abraham and McGee mentions four teammate models.

Master-slave model The player commands the teammate bot fully, and the bot has little or no artificial behavior.

Semi-autonomous slave model The player commands the teammate bot when s/he desires. When no command is active the bots behave autonomously. Although bots can behave autonomously they will still always act as a slave.

Clone model This requires that all teammate have equal abilities and roles; the main reason is to complete the goal faster, i.e. the more teammates the faster it goes. E.g. almost like fetching water 100 buckets of water from a distant well, you can manage it yourself but it goes faster with more people.

"Buddy" model Both player and bot has comparable weaknesses and the bot does not act just as the player's slave.

We would like to add another one, semi-autonomous model, which is how BATS works, i.e. acts autonomously and it can take commands, but does not always listen to them, e.g. when BATS is under attack and the teammate orders it to attack, BATS will rebel and ignore the command and instead stay and defend its base; this means that BATS is not a slave (by their definition).

While the autonomous behavior of BATS is described throughout the rest of this chapter, an overview will first be given.

The goal of BATS is to behave more like a human, i.e. use strategies that humans use, communicate more like humans, and so forth, as it has been shown that at least enemy bots are more fun when they behave like humans[34].

Figure 4.2 shows a high level overview of the most important classes of BATS. The figure is a bit cluttered, but 7 of the 15 associations either go to the Squad or IntentionWriter.

The Commander is in charge of expanding, attacking, and scouting and uses SelfClassifier or PlayerArmyManager to decide on what action it shall take—PlayerArmyManager handles the teammate's virtual squads and is used to see what the teammate does with those squads. Expansions can be created through build orders, or added autonomously to the build order when BATS's resources are getting high. It can also expand when it has launched an attack or vice versa, attack because it expands. Instead of doing random attacks, it will try to create attacks when it expands, so the attack covers the expansion which is a common (among professionals) and good strategy in StarCraft[30]. It can also launch an attack if an upgrade will finish soon, or if the teammate is moving out to attack. Scouting is done all the time, except the first couple of minutes in the game and

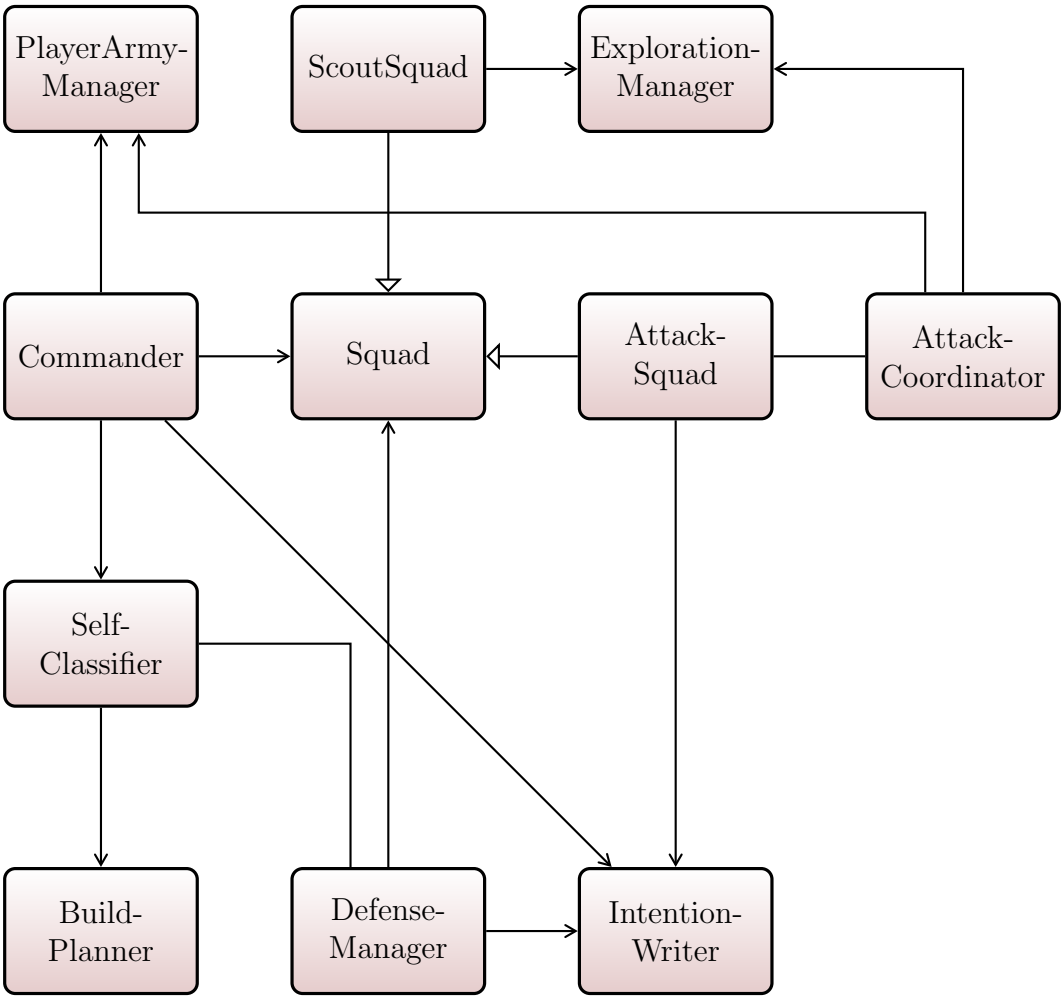

Figure 4.2: System overview of BATS

when BATS is under attack. Section 4.4 Commander describes the triggers to expand, attack, and scout in more detail.

DefenseManager decide where units (good at defending) shall be placed, i.e. close to choke points that abuts to an undefended region. The rest of the units will patrol between the choke points. BATS will however help the teammate player when s/he is under attack, but will never have any stationary units there. Section 4.5 Defense manager describes BATS's defense in more detail.

Attack squads asks AttackCoordinator for a place to attack which prioritizes the location depending on the structure type and distance from other attacks. It prioritizes important structures, such as newly created Expansions. The squad will retreat when the enemy squad is too strong, or if it destroyed all structures near the attack location. Section 4.3 Attack coordinator further describes where the Attack squads finds their location and 4.7.2 Attack squad describes Attack squads in more detail.

Scout squad moves to an location while trying to avoid enemies, the scout asks ExplorationManager for the location to scout, which is the location that was visited the longest time ago. When it either gets to the location or is interrupted by enemies it will ask for a new location to scout.

The Build planner creates structures from a build order, the build order is not fixed at supply counts but is instead listed in order and the first (not built) structure in the build order will be build directly when BATS has enough resources. BATS will, however, prioritize unit production over all structures.

The Commander, DefenseManager, and AttackSquad uses IntentionWriter to write messages; messages are sent by intention and an optional reason, the same intention will however not be sent again if it is sent within the timeout limit.

4.2 Exploration manager

The Exploration manager in BATS uses most of the functionality from BTHAI, but some functionality has been changed, added, and removed to work better with BATS. The main function of the Exploration manager tracks when a region and expansion was last checked. It does this by by checking if either a region's center or an expansion position is visible, if it is, it will automatically update the time of the last visit. This information is used by both Scout squad to get the next scout location—i.e. the one with oldest visit time—and Attack coordinator to get expansions that have not been visited for 150* seconds—used for attacks when no structures have been found. How Scout squad and Attack coordinator works are described in section 4.7.6 and 4.3 respectively.

Exploration manager saves all structures it spots, because BATS does not have full map vision and thus needs to save the information of the enemy. This information is used by Attack coordinator to decide a place to attack and Attack squad to find a common spot to attack when it is close to the target location.

4.2.1 Resource counter

Although the Exploration Manager handles where to explore and enemy structures placements, it does not handle how many resources there are in each base. This is done by the Resource counter, although it currently only counts the number of minerals. In the start of the game it goes through all static mineral locations and set their mineral count to their initial value (1500 usually). Whenever a mineral field is visible it will update the value of that mineral field. It will, however, not try to calculate how many minerals there might be currently if the mineral field is not visible (as it could by checking the rate of the decline).

To keep track of which resource belong where the Resource counter has resource groups, which in turn contains the mineral fields. Both the Resource and ResourceGroup classes have the ability to return current and initial number of re-

sources, in addition it can calculate how much of the resources are left in fractions in the interval of [0.0, 1.0].

4.3 Attack coordinator

Frontal attacks attack the enemy head on and usually use the majority of the player's army, distracting attacks are small attacks either used to distract the enemy from one position where another attack will come (hence the name), or they can be used to try to deal some damage to the enemy, with heavy emphasis on try. These distracting attacks can be drops, harassment, counter-attacks[1]—BATS has, in time of this writing, only implemented the drop functionality. The distracting attacks are not suicide attacks, although exceptions exist; they are mostly distractions to pin the enemy at his base.

Where to attack? The attack coordinator uses two to three weights multiplied together to get the near optimal attack location—the attack location is not optimal as the weights have not been extensively balanced, but the algorithm works good enough for these experiments. The attacking locations are derived from seen enemy structures and expansions that have not been scouted for 150^* seconds.

The first weight, *distance*, prioritizes structures (and locations) far away from other existing attacks, not including teammate attacks yet; as it is a good idea to spread out multiple attack to split up the enemy army—i.e. we are not implying that multiple armies are better than a big, only that when using multiple armies it is good for the armies to spread out. The second weight, *type*, determines how important a structure or location is to attack, and the third weight, *defended*, is how well defended an area is—the defended weight is not implemented yet, but distracting attacks will use it to get a location that can be attacked without taking the risk to die (such as drops dying to anti-air).

In addition to these weights, if the squad is a frontal attack and follows an teammate squad, it will get an attack position near the teammate squad's target instead of using the weights. The attack position is still the same positions used for the regular algorithm.

Enemy armies are not included in the calculation, because one rarely gain anything on trading armies, one should find gaps in the defense instead[30]. It is still good to engage a weaker enemy army and BATS will do that only if the squad is close by, it will not search for those small armies. In addition, because armies are mobile it would be hard to track them, i.e. time consuming to implement such a feature.

The flowchart in figure 4.3 shows an Attack squad requesting an attack from attack coordinator. It can either get an weighted attack location or join the

[1] A drop when we are under attack can be called counter-attack, but our meaning of counter-attack is a frontal attack, but smaller

teammate's frontal attack, if s/he has one. Attack coordinator uses a rule based system to decide the how and where the squad shall attack.

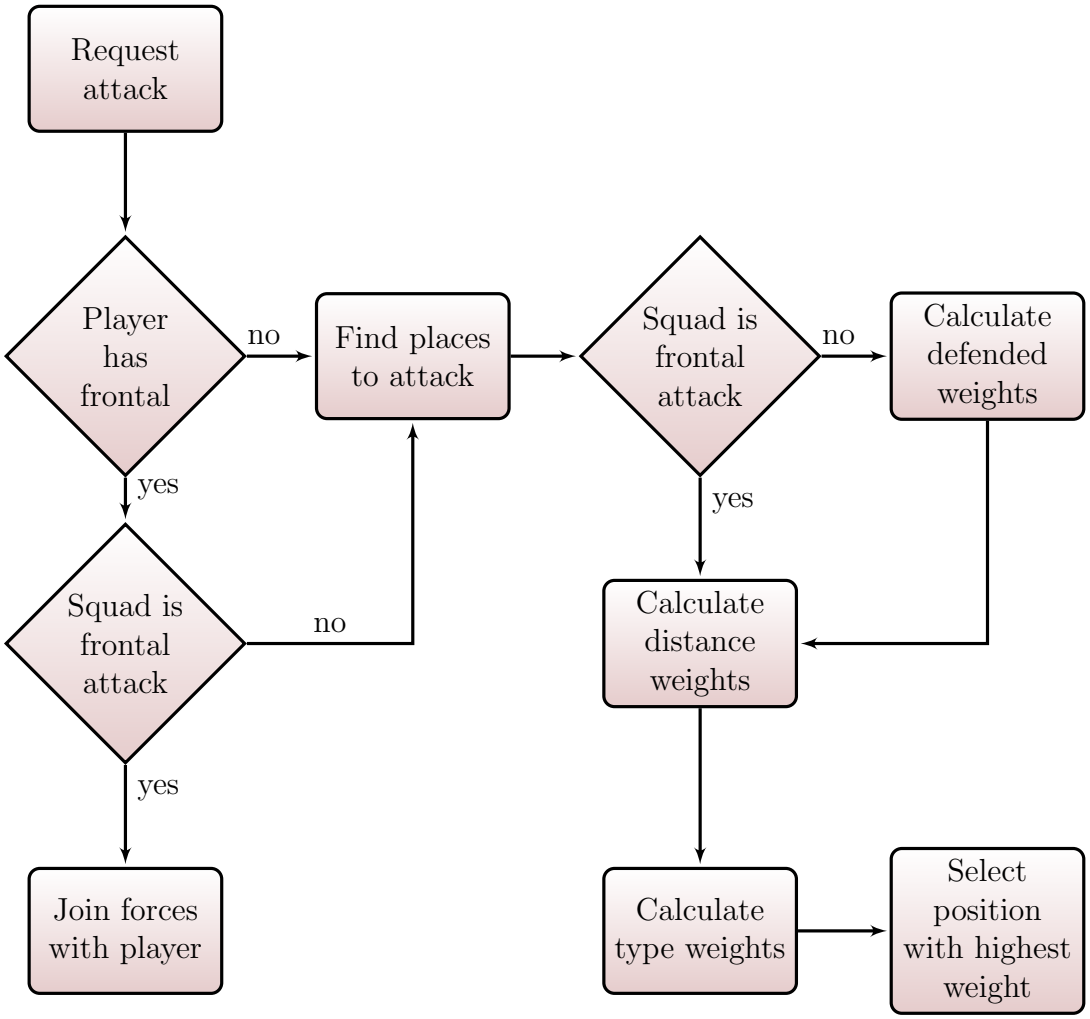

Figure 4.3: Flowchart of Attack Coordinator's requestAttack() function

Coordinated attacks To achieve a sense of coordinated attacks, Attack squads have Wait goals added by the attack coordinator—fully described in section 4.7.8. The Wait goal for Attack squads is a Wait goal that succeeds whenever an attack is within position (close to its goal and waiting), or is already attacking something. The goal fails when it has timed out (30^* seconds). The Wait goal is added to all existing squads and the new squad gets the already existing Wait goals, when all attacks are in position they will start their attack simultaneously.

4.3.1 Calculation of distance weight

Distance weight is a simple calculation: if no other Attack squads are present it defaults to 1.0. Otherwise the average from all other attacking squads are calculated as described in equation 4.1.

$$weight = \frac{\sum_{i=1}^{s} distance(i)^2}{s} \qquad \{s = \text{number of Attack squads} \tag{4.1}$$

Here *weight* is the average distance from all other attacking squads. The weight is then normalized to [0.0, 1.0], shown in equation 4.2—dividing the weight with the maximum map distance. As with most distance calculations, they use the squared version for faster calculation since no root calculations are needed.

$$normalized\ weight = \frac{weight}{MAP\_WIDTH^2 + MAP\_HEIGHT^2} \tag{4.2}$$

4.3.2 Calculation of type weight

This weight prioritizes different structures and locations. Most of the values here are fixed, expansions are however calculated between fixed values. The structures and locations are presented in priority order below with the current values assign to them. These values are rough estimates what might be good, thus one shall not think these are *the* values. Because the weights are multiplied this will give a logarithmic behavior, i.e. the difference between 0.1 and 0.2 is greater than 0.2 and 0.3 (0.2 is the double of 0.1, whereas 0.3 is less than double of 0.2).

Not scouted expansions 1.0* These are expansion that have not been visited for the last 150* seconds. The idea is to check expansions when moving out to attack, although the current version does not work as expected; because the Scout squad cannot cover all the areas quickly the Attack squad will act more as a scout than Attack squad because of visiting expansions in various location around the map. To fix this only the closest expansions to the enemy shall be checked and possibly only one with this squad, as this is roughly what Magnusson has observed when watching professional StarCraft players. This feature was therefore disabled during the experimentations.

Expansions [0.0, 1.0]* Expansion weight for existing enemy expansions. The weight is higher for fresh expansions; fresh expansions being defined by the expansions current mineral amount (in fractions)—the Resource counter tracks the amount of all mineral fields.

The minimum value can, in addition to its normal state, be used as a kind of ceiling function; e.g. if it is set to ceil*and the minimum is set to 0.5 while only 20% of the minerals are left it will ceil the weight to 0.5, see equation 4.3. If it

is set to normal, the fraction will be normalized in the [0.5, 1.0] range, meaning the weight will be 0.6, see equation 4.4.

$$weight = \begin{cases} minValue & \text{if } fracMinerals(x) < minValue \\ fracMinerals(x) & \text{if } fracMinerals(x) \geq minValue \end{cases} \tag{4.3}$$

$$weight = fracMinerals(x) \times (maxValue - minValue) + minValue \tag{4.4}$$

Addons 0.4* Addons are structures built in connection to another structure and only exist for Terran. These are both upgrade structures and can make either the attached unit producing structure create advanced units (such as tanks), or all specific unit producing structures able to produce the unit—e.g. the addon Covert Ops for a Science Facility makes Ghosts available for production in all Barracks. The only exception to this is the addons to the Command Center which is either Comsat for detection or Nuclear Silo for nukes.

Supplies 0.35* Supplies are structures that provide space (or food) for more units to be created, see section 1.1.1 for how supplies work. This supply priority only makes sense for Terran as they work differently for the other classes: Protoss also uses their supplies for powering buildings, i.e. a higher priority would be good here; for Zerg the supplies are increased by the Overlord unit, this would require an additional algorithm for searching for Overlords, the priority would be higher, maybe even higher for only air attacks.

Upgrade structures 0.3* Upgrade structures does not, despite its name, include all structures that can upgrade, only structures that can upgrade general attack and defense bonuses are treated as upgrade structures. Meaning Terran Academy (which upgrades Stim packs for Marines) is not treated as an upgrade structure. The reason is because of simplicity; attack and defense upgrades come in three steps meaning there are 6–12 upgrades in total (depending on what structure) for that structure and is continued to be upgraded throughout the game. This means its a lot bigger chance that the structure still is useful (for the enemy) even when attacking it later in the game, as opposed to the Academy which upgrades are long done—although in this case medics and firebats could not be created by the enemy, but the enemy can simply prioritize marines in the meanwhile and not much harm would be done.

As an improvement BATS could keep track of which upgrades it has seen and then exclude these buildings. Attack and defense upgrades can be directly seen on the enemy while unit abilities, but stim, and other ability upgrades needs to be activated by the enemy for BATS to know about it.

Unit producing structure 0.2* All structures that can produce units. These are not ordered in any specific order, not to say that it does not matter because it does, but it will always vary depending on the situation. If the enemy has 10 Barracks and 1 Starport, it is probably best to destroy the Starport, whereas if the enemy has 1 Barrack and 4 Starports it might be best to destroy the barracks.

Other structures 0.1* All other structures that have not been covered, this includes research structures and defensive structures, such as Academy, Bunkers, and Turrets.

Why this order?

Because StarCraft is mostly about managing expansions[30], BATS tries to deny and kill fresh expansions as its first priority (the top two priorities). Targeting addons can both stop the production of an important upgrade (siege mode for tanks) and the ability to create tanks, ghosts, etc. Delaying late game units is usually good as they are generally better than early game units. Supply depots are almost always good to destroy since it halts all unit production, unless the enemy has stacked up lots of supplies—which almost always is a bad strategy—or lost many units in a recent battle. Stopping an upgrade is probably better than killing a unit producing structure, because if the upgrade finishes all existing units will get the upgrade, while if we kill a unit producing structure only 2–4 units will get stopped, but it depends on the type of unit producing structure, what upgrade structure etc; but this priority will do for now.

4.4 Commander

The Commander creates all orders for BATS, except defenses as the Defense manager does that. This means that it orders attacks, drops, expansions, scouts, and when to transition to the next phase.

4.4.1 Orders/Commands

Below are all orders that both BATS can decide to use (described further down) and the teammate can order through his commands. The commands almost behave exactly the same independent on who ordered the command except that if a command fails it will only print out an output if the teammate ordered the command—some other exceptions exist.

The orders work differently depending on the current state of BATS. Because of all tests and different behaviors in the commands, these are easily described in pseudo-code. Common for all functions are the two parameters:

`teammateOrdered` and `reason`; teammateOrdered is set to true if the teammate ordered the command, reason is the reason to print out if the command is successful, the reason parameter is only used when BATS ordered the command—e.g. attack because BATS expands. The messages are sent with the help of `IntentionWriter`; how IntentionWriter works is described in section 4.9.

Attack Has in general three behaviors: Creates a new frontal attack, reinforces the existing frontal attack, or does nothing. Listing 4.1 displays the pseudo-code for the attack command.

```
// Never do a frontal attack when under attack
if (isUnderAttack()) {
  if (teammateOrdered) {
    IntentionWriter.write(BotAttackNot, BotIsUnderAttack);
  }
  return;
}

// Teammates can create attacks even if few units
if (teammateOrdered) {
  canAttack = !freeUnits.empty();
} else {
  canAttack = canFrontalAttack(); // Checks enough units
}

if (canAttack) {
  oldSquad = mSquadManager.getFrontalAttack();

  // Add free units to the old Attack squad if it exists
  if (oldSquad != NULL) {
    oldSquad.addUnts(freeUnits);
    IntentionWriter.write(BotAttackMerged, reason);
  }
  // Create new attack and ping position
  else {
    attackSquad = new AttackSquad(freeUnits);
    attackPos = attackSquad.getAttackPosition();
    IntentionWriter.write(BotAttack, reason, attackPos);
  }
} else {
  IntentionWriter.write(BotAttackNot, BotNotEnoughUnits);
}
```

Listing 4.1: Pseudo-code for the attack command

Follow Follow almost behaves as the attack command, but instead of creating a regular Attack squad, it will instead follow the largest teammate player's squad. In addition to the attack's three behaviors—create new frontal attack (that follows

teammate), reinforce the frontal attack, or does nothing—it has one additional behavior: If BATS already has a frontal attack, that is not following a teammate, it will abandon its goal and instead follow the teammate.

Drop Drop will try to create a drop from any of the unit compositions that are available.

```
availableCompositions = getUnitCompositionByType(freeUnits, Drop);

// Create drop
if (!availableCompositions.empty()) {
  randomId = rand() % availableCompositions.size();
  chosenComposition = availableCompositions[randomId];

  drop = new DropSquad(freeUnits, chosenComposition);
  IntentionWriter.write(BotDrop, reason, drop.getDropPosition());
}
// No drops available
else {
  if (teammateOrdered) {
    IntentionWriter.write(BotDropNot, BotNotEnoughUnits);
  }
}
```

Listing 4.2: Pseudo-code for the drop command

Scout As with drop, scout uses available unit composition; it does however not choose a random composition but uses the composition with highest priority—i.e. the first composition, as UnitCompositionFactory's getUnitCompositionByType() returns composition sorted by priority, where the highest priority is first.

Expand Expand will first check if there are any expansions available, i.e. not already taken; if there are expansions available it will append a command center (or equivalent for other races) to the Build planner, which in turn will build an expansions when it has resources for it.

Transition Transition transitions to the next phase in the game—there are three phases: early, mid, and late game. How the Commander handles the transition command can be seen in listing 4.3.

4.4.2 Order creation rules

The Commander can create orders either from its own actions and states, or what the teammate player is doing. Examples of this is it might attack when it

is expanding, and it might expand if it has high amount of minerals, for reaction to teammate actions it attack if the teammate player is expanding.

```
// Only transition if not in late game already
if (mBuildPlanner.canTransition()) {
  if (mBuildPlanner.getCurrentPhase() == "early") {
    intention = BotTransitionMid;
  else if (mBuildPlanner.getCurrentPhase() == "mid") {
    intention = BotTransitionLate;
  }

  mBuildPlanner.switchToNextPhase();
} else {
  if (teammateOrdered) {
    IntentionWriter.write(BotTransitionNot, BotTransitionNoMore);
  }
}
```

Listing 4.3: Pseudo-code for transition command

Reacting on own actions and states

These are the reactions (commands) to BATS's own state.

Expand In order to expand from own reactions BATS needs to meet all conditions below.

- **Not be under attack.** Expanding when we are under attack will most likely kill the expansion directly
- **Not already be expanding.** We do not want to spam expansions
- **Less than 3* active expansions.** Good number of active bases as it keeps a large and steady income to support many unit producing structures. Active expansions are expansions where at least 15%* minerals left.
- **No new expansion shall have been build in 150* seconds.** Again we do not want to spam expansions and this seemed like a reasonable time, but has not been extensively tested.

When all four conditions are met it will check if any of the following conditions are met, if they are an expansion will be added to the beginning of the build order.

- **BATS is attacking.** It is good to expand when attacking (or vice versa) as this distracts the enemy from the expansion long enough for the expansion to complete[30]

- **Expansions are saturated.** This tests if the number of workers per mineral patch is at or above 2.5*. Simply put, even if you add more workers to mine, they will have to wait longer for the mineral patch to be free, thus it will not increase your income. This rule will allow BATS to expand early in the game when it has less than 3* expansions.

- **An expansion is running low on minerals.** Making sure we stay on the same number of active expansions. This is the same as an expansion that is not active (i.e. having less than 15%* minerals left) but still having some minerals left to be mined.

- **High on minerals.** If we have too much minerals that means we cannot build enough structures or units, thus we can as well add another base and possibly increasing the amount of gas mined. Activates when the mineral count is above 550*.

Attack To attack from own reactions BATS shall meet all conditions below.

- **Shall not be under attack.** It is better to stay and defend and then possibly attack, although a small counter-attack might be effective when the enemy army is small and BATS can spare some units for the attack, counter-attacks have not been implemented.

- **Not have a current attack.** Having one big army is easier to control and generally much stronger than splitting it into 2 smaller attacks for slower armies[30], and BATS will use slower armies in the experiment. Although ordering a second attack when we have a frontal attack will reinforce units, but we did not have enough time to implement and test when to reinforce the army.

When all two conditions are met it will check if any of the following conditions are met, if they are an attack will be created.

- **Expanding.** As explained earlier under the expand, it is good to attack while expanding.

- **Upgrade soon done.** When an upgrade will finish soon this means that the bot will get stronger and it might be possible that the enemy has either not caught up in upgrades, i.e. the longer we wait the less valuable the upgrade is[30]. Upgrade soon done checks if any of the free units that could be used for an attack will be affected with an upgrade that is currently upgrading and if the upgrade finishes within 60* seconds.

Scout The scout order conditions are much simpler than expand and attack. BATS will always send out a scout if it is not scouting, not under attack, and has 15* workers. The Commander uses Unit compositions for the scout to get the best available scout unit. From lowest priority this is:

1. SCV
2. Marine
3. Vulture
4. Wraith

Wraiths are flying units with cloak ability (if researched) this make them an ideal scout as they fly over terrain, are quite fast, and can cloak if they get too close to the enemy. Vultures are on the other hand the fastest ground unit for Terran. See section 4.7.6 how squads work, listing A.6 for the unit composition config file, and 4.7.7 for how unit compositions work.

Transition The transition order conditions are just as simple as the scout. It will transition if it is not already in late game, no buildings in the build order, and is high on resources (higher or equal to 550* minerals and 500* gas).

Reacting on teammate actions

All orders here are grouped by teammate actions instead of the commands (as opposed to reacting on own actions), this feels like a more intuitive approach and was implemented this way.

Teammate expands If the teammate player expands, BATS first want to create a distracting attack (for now only drops are available) to distract the enemy from the expansion, if no drops are available it will create a frontal attack. But before the Commander creates any attack it checks so that we are not under attack or already attacking.

Teammate attacks Depending on what type of attack the teammate player has BATS behaves differently, but common for all types of attack, it will not do anything if it is under attack. If the teammate attacks with a frontal attack BATS will first try to join it if BATS does not already have an frontal attack or not enough units (12* units) for a frontal attack, otherwise it will try to drop if it does not have a drop and has enough units for a drop.

When the teammate player has a distracting attack out BATS will try to create a distracting (drop) attack as well, but will only succeed if it has enough units.

4.5 Defense manager

The Defense manager handles the defense of its own base and the player's base. It calculates defense locations from BATS's and the teammate's outer choke points (marked (●) and (●) in figure 4.4. It does not defend a region that only abuts to BATS or teammate occupied regions (—) .

The Patrol squad patrols between the (●) locations. Hold squads guards these locations but are stationed in the roaming circle (○) where the center is calculated by finding a position in the range [5, 8]* tiles closest to one of BATS's or the teammate's buildings and the radius being 4* tiles. Units from the Hold squad will stay in the roaming perimeter until enemy units enter the defense perimeter (10* tiles in radius), they will then start to attack the enemy.

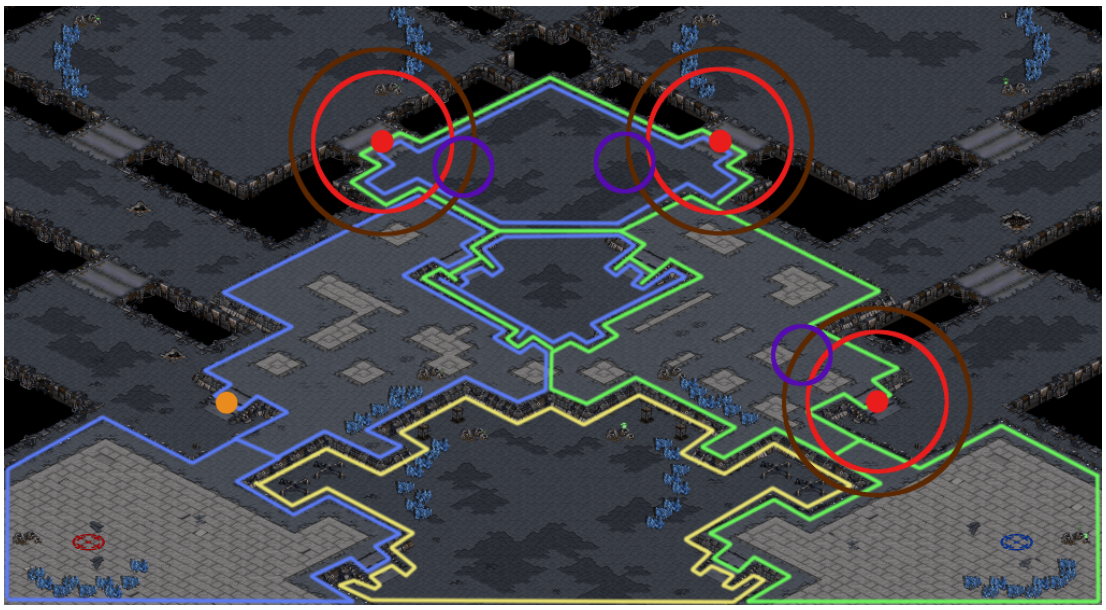

Figure 4.4: Example of defense locations. (—) BATS's regions; (—) teammate's regions; (=) belongs to both; (—) non-defended as they abut only BATS or teammate regions; (●) defense locations; (○) defend perimeter; (○) enemy offensive perimeter; (○) roam location for Hold squads; and (●) teammate's defense location.

Hold squads only contains unit good at defending choke points, whereas Patrol squads contains all free units and patrol between the defense locations. If enemy units enter the enemy offensive perimeter (12* tiles in radius) in a defense location all squad units will move to the defend location to defend it, all other hold squads will disband to join the defense with the Patrol squad. The idea was first to only disband the other Hold squads if the enemy is too strong, but the calculation proved to be more time consuming than guessed, thus we focused work on other more important areas. The strategy was tested and works good against

one StarCraft default bot, as it only attacks from one location at a time. Two default bots can however attack from different locations, but usually attacked from one location. Two bots were, however, often too strong for BATS to defend by itself. Although the strategy worked against the default bot it would probably not work against other bots or humans that can attack two different locations simultaneously.

Both Hold squads and Patrol squads roam or patrol between BATS's defense location, meaning the bot will not disturb the teammate with its units.

Defending other locations Although no Hold squads are located on and the Patrol squad never patrols between the teammates defend locations. It will still send the Patrol squad to defend a teammate defend locations if BATS is not under attack itself. In addition to defending all defend locations, the Defense manager will send out the Patrol squad to defend locations either inside BATS's or the teammate's base that are under attack, for example if the enemy drops inside the base. This will, however, never bring the Hold squads as they should defend the entrance to the base.

4.6 Player classification

The player classification is split into two separate modules. One handles the grouping of teammate and enemy virtual squads. They are virtual as these are BATS estimations that they belong to the same squad. The other module groups common classifying tests for both teammate and BATS.

4.6.1 Player squads

The `PlayerArmyManager` creates and holds all the virtual squads, both teammate and enemy squads. Teammate and enemy squads derive from the PlayerSquad class which keeps a record of its own center position and supply count for the last 5^* seconds; this enables more complex calculations, such as the direction of the squad and if it's increasing/decreasing in size. These are the most important PlayerSquad function:

`TilePosition getCenter()` Current center position of the squad.

`int getDeltaSupplyCount()` How many supplies have increased or decreased during the last 5^* seconds.

`TilePosition getDirection()` Non-normalized direction between the center of the squad during the last 5^* seconds.

`int getSupplyCount()` Current supply count.

`TilePosition getTargetPosition()` Calculates where the majority of the units has as a target position. Useful when wanting to know where the player is heading.

`int getUnitCount()` Number of units in the squad.

Grouping algorithm Simply put, player army manager starts creating a virtual squad for a teammate player and then searches for close units within the include distance (6^* tiles) to include in the squad. The next update it will start with a unit already in a squad and again try to include new units, but it will also remove units from the squad that are further away than the exclude distance (8^* tiles). Enemy squads are created the same way as teammate squads.

Testing the distance between so many units were too computational heavy, even when using the squared distance version with no square roots. Instead units are inserted in a grid with a width of half the exclude distance. Using half the exclude distance decreased the number of distance computations further. Figure 4.5 shown an example of the grid with outlined include and exclude distances. To further increase the computation speed, instead of computing which grid the unit shall be placed in a lookup table is used instead.

```
// Iterate through all squads, one unit each turn
do {
  foundUnit = false;
  while (moreUnitsToCheck() && !foundUnit) {
    if (unitSquad.isValid() && !squadChecked(unitSquad) {
      currentUnit = unit;
      foundUnit = true;
    }
  }

  if (foundUnit) {
    setUnitAsChecked(currentUnit);
    setSquadAsChecked(unitSquad);

    addCloseUnitsToSquad(currentUnit, unitSquad.getId());
  }
} while (foundUnit);

// Create new squads for the rest of the units that either
// went outside the squad's bounds or never had a squad
while (moreUnitsToCheck()) {
  setUnitAsChecked(unit);
  newSquad = new PlayerSquad();
  setSquadAsChecked(newSquad);
  addCloseUnitsToSquad(unit, newSquad.getId());
}
```

Listing 4.4: Pseudo-code of `rearrangeSquads()`

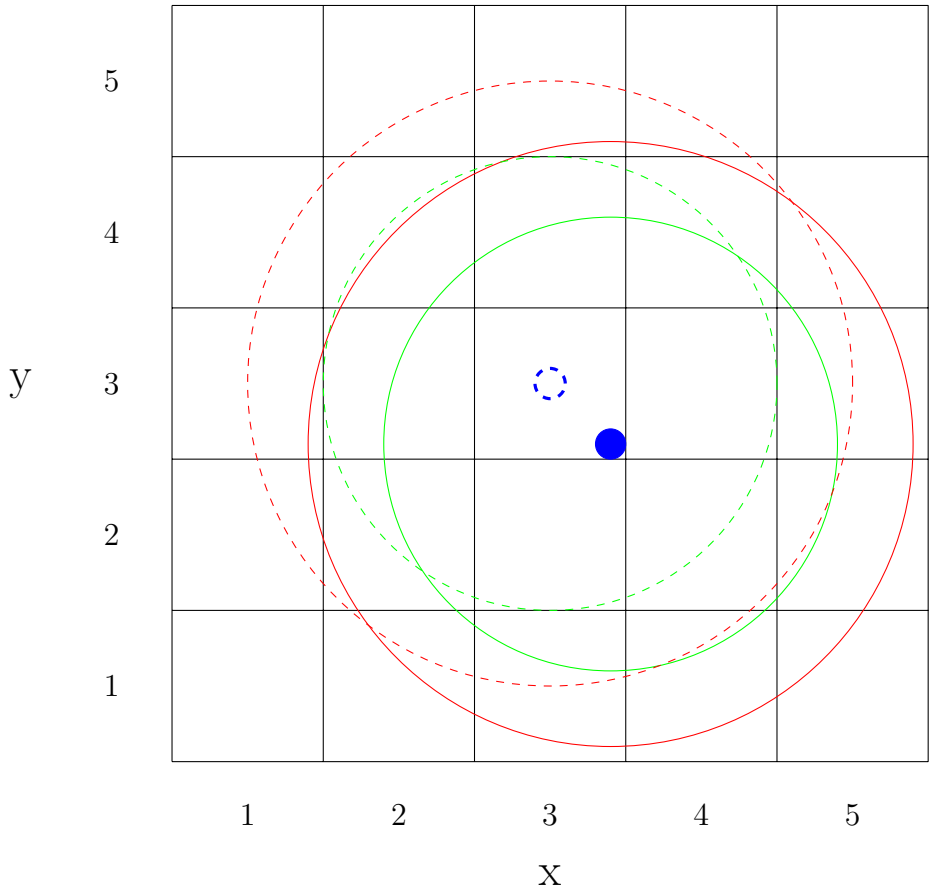

Figure 4.5: Example of using grid to minimize distance calculations. (●) The unit. (○) include distance; (○) exclude distance. The dashed circles represents another unit, its include and exclude distances from the middle of a grid square.

Only units within two grid units are used when calculating the distance, this generates a 5x5 grid just as figure 4.5. As can be seen in the figure units further away than two grid units can never be within the exclude distance. Units in the same grid or to the left, top, right, or bottom are almost always within the exclude distance, an exception can be seen in the figure where small portion of the upper left corner at (2,3) is outside the solid exclude circle; for simplicity all units within these grid positions are seen as within the exclude distance (even though exceptions might exist).

To make use of these calculation benefits, units in a squad are checked first (if they are further away than the exclude distance). A simplified pseudo-code example is shown in listing 4.4 where high-level calculations are made and in listing 4.5 where calculations of close units to add to the squad—note that units are not excluded form the squad here, they are not excluded until they are added in another squad which happens at the end of `rearrangeSquads()`.

```
// Add all units that are in the same square or the
// bordering left, top, right, bottom square.
foreach (square in same, or border square) {
  foreach (not checked unit in this square) {
    if (same squads as paramUnit) {
      queuedUnits.push_back(unit);
      setUnitAsChecked(unit);
    } else if (withinIncludeDistance(unit, paramUnit) {
      queuedUnits.push_back(unit);
      setUnitAsChecked(unit);
    }
  }
}

// Recursively check queued units, this is done afterwards so
// that all fast calculations can be done first
while (!queuedUnits.empty()) {
  addCloseUnitsToSquad(queuedUnits.front(), squadId);
  queuedUnits.pop_front();
}

// Directly recursive, instead of queueing units it will directly
// call addCloseUnits... because those units could add close units
// that were within bordering squares that otherwise would be
// calculated here using distance calculations.
// Uses the full 5x5 square, except squares already checked
foreach (other square) {
  foreach (not checked unit in this square) {
    if (same squad as paramUnit) {
      if (isWithinExcludeDistance(unit, paramUnit) {
        setUnitAsChecked(unit);
        addCloseUnitsToSquad(unit, squadId);
      }
    } else if (isWithinIncludeDistance(unit, paramUnit) {
      setUnitAsChecked(unit);
      addCloseUnitsToSquad(unit, squadId);
    }
  }
}
```

Listing 4.5: Pseudo-code of `addCloseUnitsToSquad()`

4.6.2 Grouped classifying tests

A common class uses the facade pattern to group both easy and complicated tests into one class to minimize the coupling between classes, reuse code, and have to have the code in one place if a tests was to change (instead of changing it in several places)—e.g. a simple test like `canFrontalAttack(units)` which returns

true if the unit size is larger or equal to the minimum number of units BATS needs for a frontal attack:

```
return units.size() >= FRONTAL_ATTACK_UNITS_MIN;
```

None of the classifying classes modifies any data, thus all functions are const and all three classes (although an enemy classification was not implemented) were supposed to have a common interface to make it easier to use all three classes without having to learn three different interfaces. Due to lack of time and not being the most important aspect this feature was skipped in this version.

Self classification This classifier groups together common tests for BATS and might not be seen as a classifier. Commander and Defense manager uses the Self classifier to easier calculate their goals and decrease their coupling. Below all important functions are listed and described, although most of the names should be self-explanatory.

`bool areExpansionsSaturated()` Tests if there exists enough workers to saturate all mineral patches.

`bool canDrop()` If BATS have enough units to create a drop.

`bool canFrontalAttack()` If BATS have enough free units to create an attack. This is only used for when BATS want to create an attack, if the teammate orders an attack it is enough with one unit.

`int getActiveExpansionCount()` Number of active expansions, i.e. expansions with more than 15%*.

`double getLastExpansionStartTime()` How many game seconds ago the last expansion was started.

`bool hasDrop()` Checks if BATS has a drop.

`bool isAnExpansionLowOnMinerals()` Checks if an expansion has less than 15%*.

`bool isAttacking()` Checks if we have an Attack squads that is not retreating.

`bool isExpanding()` Checks if BATS is either expanding or going to expand (expansions in build order).

`bool isHighOnGas()` BATS has more than 500* gas.

`bool isHighOnMinerals()` BATS has more than 550* minerals.

`bool isScouting()` If BATS has a Scout squad out.

`bool isUnderAttack()` Checks if BATS is under attack, uses Defense Manager's isUnderAttack().

`bool isUpgradeSoonDone(affectedUnits)` Checks if an upgrade, that will affect the specified units, completes within 60* seconds.

Teammate classification The teammate classification works just as the Self classifier described above, but at the moment only has one function, isExpanding(), as the squad functionality was implemented in PlayerArmyManager. isExpanding() checks if the teammate currently constructs a new base structure (e.g. Command Center).

4.7 Squads

A squad, in BATS, are units grouped together acting in coordination. All squads have the abstract Squad class as their base class.

Creating, deleting, and retrieving squads The squad manager updates and removes (only from the squad manager) existing squads; any class can create a squad and it adds itself automatically to the squad manager (via the squad's constructor). The squads are added as `shared_ptr`, meaning they are automatically destroyed when no more references exist, i.e. no memory leaks. When a squad holds no units—achieved when all units died, units moved to another squad, or the squad disbands—the squad manager automatically removes the squad; it can still be used if another class saved it before it was removed. Removed Squads have to be recreated to be inserted into the squad manager again.

Squads can be retrieved using two (actually four) different functions, either by the squad's id (automatically set) or by squad type (template function), shown in listing 4.6. The template function will also return all sub-classes to the specified class, meaning `getSquads<AttackSquad>();` would return both Attack squads and Drop squads because Drop squad derives form Attack squad. The last two functions are there for convenience, the first returns the current frontal attack (if one exist), and the other returns all distracting attacks.

```
template <typename T>
vector<shared_ptr<T>> SquadManager::getSquads();
```

Listing 4.6: Template function to retrieve squads of the specified type

4.7.1 Squad base class

Only the core functions of the squad base class are covered below, as there are too many helper functions to cover them all. If the reader wants to find out more information about the internal structure of the squads, s/he can look in Appendix B. Squad movement will be covered first, then the squad's basic behaviors, which can be overridden by derived classes; and finally how the `update()` and `updateDerived()` functions work.

Squad movement This is not the specific unit movement, but where the squad's units move to. The potential field manager takes care of how units move and was already implemented in BTHAI[20]—although some minor changes have been made to the potential field manager to accept some general squad behavior, more specifically moving towards the retreat location when retreating and when close to enemies. The squad has five prioritized locations it can move to. Starting from the lowest priority is

1. The goal location where the squad want to go to do their main task (e.g. scout, attack, defend location).

2. A retreat location to be able to retreat from e.g. an attack; there is no automatic retreat behavior in the base class, derived squads has to set the retreat location and the base class calls `onRetreatCompleted()` when the retreat succeeds or fails.

3. Via path, the squad will move through these locations when moving to either the goal or a retreat location. Useful when attacking or retreating with a drop to make it move along the map's edges so that it does not run straight through the enemy.

4. Temporary goal location, used when derived classes need an additional goal location, such as Attack squads waiting in a location to attack or Hold squads moving from their roaming area to the defended area when an enemy enters the defend perimeter. The temporary goal location is, however, disabled when a retreat location is active.

5. Regroup location, the regroup is automatically handled by the squad when units are too spread out—a unit is further away than 12^* tiles from the squad's center, the regroup stops again when all units are within 10^* tiles. The squad will regroup to the unit who is closest to the goal location—first the squad's center was used but this caused units in the front to move backward and did not work very well, in addition the squad movement felt weird as if BATS could not decide what it wanted. Bugs can still occur when moving around a C shaped cliff as units to the left of the C are further away then those at the bottom location, if the goal is to the right of C. The regroup functionality can be disabled by derived classes, useful when sending reinforcements to the squad.

Behaviors The squad has four elements that can change the behavior of the squad, described below.

Regrouping which has been fully described above in squad movement.

Retreating . While the base class squad never uses this directly, derived classes can set a retreat location and the squad will then check when it is close to the retreat location. Once close it will call `onRetreatCompleted()` which has the default behavior of disbanding the squad, thus merging it with the Patrol squad. By overriding this function another behavior can be accomplished. Retreat locations are most commonly retrieved from Defense manager via the `findRetreatPosition()`, if a squad uses another location this will be mentioned.

Unit composition a unit composition limits the squad to only contain units of the specific type, useful when creating special type of squads. Section 4.7.7 describes unit composition in more detail.

Avoid enemies Some squads do not want to get close to enemies, in this case avoid enemies can be turned on. This causes attacking units to move to a specific location without attacking while trying to avoid enemies, this works most of the time but not always; the potential field manager in BTHAI did not support this behavior entirely as it needs to check that the unit retreats in the right direction, this was slightly improved by adding the goal location to the potential field calculation when avoid enemies is set, the calculation of goal location was already implemented but not used in BTHAI.

Wait goals has been mentioned earlier in section 4.3 Attack coordinator and is fully described in section 4.7.8 Wait goals. While adding these does not give the base squad any specific behavior, they can be used in the derived classes for trigger behaviors.

4.7.2 Attack squad

The Attack squad can both function on its own, but is also the base class for all attacks—in the current version only drops derives Attack squad. Used alone, it can either do a regular attack or follow a teammate squad—a regular attack or battle in this section is a synonym for an attack or battle where it not follows a teammate squad.

For regular attacks the squad requests an attack from Attack coordinator to get an attack location to move to and regroup when needed. When the squad is close to the attack location (25^* tiles), i.e. its wait location, it will wait here until all its Wait goals are finished, then it will continue moving to its attack goal. This will sync the attack with other attacking squads as they wait until the squad either is in the wait location, under attack, or already attacking.

If it encounters any enemies on the road it will try to kill them, unless they are too strong where the squad will retreat instead; the squad can decide to retreat from any regular battle at any time. For an attack to succeed with its mission all structures within the radius of 15^* tiles from the attack location needs to be

destroyed. After succeeding with the mission the squad will retreat back to the base, disband the squad, and merge with the Patrol squad.

Following a teammate squad To follow a teammate squad it can either be set directly when creating the Attack squad or later by a call to `followTeammateSquad(teammateSquad)`, this converts the regular attack to follow the specified squad—Attack squads can only convert to following a teammate and not to a regular attack.

```
switch (teammateSquad.getState()) {
  // Regroup if not close
  case IdleOutsideBase:
    handleTeammateRegrouping();
    setAvoidEnemyUnits(true);
    break;

  // Go to teammate target location, don't attack
  case Retreating:
    if (isRegroupingWithTeammate()) {
      clearTeammateRegrouping();
    }
    setGoalPosition(teammateSquad.getTargetPosition());
    setAvoidEnemyUnits(true);
    break;

  // Go to target location, attack if see anything
  case MovingToAttack:
    if (isRegroupingWithTeammate()) {
      clearTeammateRegrouping();
    }
    setGoalPosition(mteammateSquad.getTargetPosition());
    setAvoidEnemyUnits(false);
    break;

  // Find something close to attack
  case Attacking:
    handleTeammateRegrouping();
    if (!isRegroupingWithTeammate()) {
      setAvoidEnemies(false);
      AttackCoordinator.requestAttack(this);
    }
    break;

  // Retreat, then merge with Patrol Squad (disband)
  case IdleInBase:
    setRetreatPosition(DefenseManager.findRetreatPosition());
    teammateSquad = NULL;
    break;
}
```

Listing 4.7: Squad actions depending on the teammate squad's state

When the squad follows an teammate squad it behaves differently, depending on the teammate squad's current state, described in listing 4.7.

While most of the code speaks for itself, some functions need extra clarification. `handleTeammateRegrouping()` checks if the distance to the teammate squad becomes too large (15^* tiles), it both sets the regrouping location to the teammate's center location and the squad to avoid enemy units—the regrouping is considered done when the distance between the squads are less or equal to 10^* tiles away. `setGoalPosition(mteammateSquad.getTargetPosition())` sets the squad to move to the location where most of the teammate squad's units are moving to; thus it will meet up with the teammate's squad around that location. If `handleTeammateRegrouping()` was used instead it would try to chase after the squads current location—much like in football (soccer) you shall run where the ball is heading and not to its current location. This allows the teammate to control where the bot shall attack as it will attack where the player decides to attack.

`AttackCoordinator.requestAttack(this)` uses Attack coordinator to find a place to attack; as mentioned in section 4.3 it will find an attack location close to the teammate squad's target. Why not use the target as with MovingToAttack and have the AttackSquad create the attack location? Because 1) we want all attack request to be in the same location, if the attacks needs to be fixed or better coordinated we only need to change it in one place; 2) in the future the squad could get a location of a prioritized building to attack; and 3) using teammate target directly will cause our units to fight exactly where the player is, which might crowd the place not making all units able to fight, but could be better if squads are smaller—more experimentation on this subject is needed.

In addition the squad needs to cope with teammate squad splits or merges. If the current teammate squad is empty of units (as it can become when it merges with another teammate squad) it will search for close (30^* tiles) teammate squads and follow the largest squad. If no teammate squads were found, it will retreat and then disband.

4.7.3 Drop squad

A Drop squad is an attack containing a flying transportations along with some offensive units it carries. The goal of the drop is to attack the enemy base from an undefended angle (as flying units can fly over all terrain). In ideal situations the drop will unload its units and destroy either significant bases or workers, but the goal is rather to distract the enemy for various reasons—e.g. we expand and do not want the enemy to attack, or we will attack from another location simultaneously.

The Drop squad uses Attack squad as its base class. Listing 4.8 shows the behavior of the squad.

```
switch(mState) {
case LoadUnits:
  if (isTransportDoneLoading()) {
    setState(Transport);
  }
  // Note, no break!

case Transport:
  // travelsByGround() = cannot load all units
  if (travelsByGround() && !isRetreating()) {
    setState(Attack);
  } else if (isEnemyAttackUnitsWithinSight()) {
    if (isEnemyFasterThanTransport()) {
      setState(Attack);
  } else {
    if (!hasWaitGoals() && isTransportInGoalRegion()) {
      setState(Attack);
    }
  }
  break;

case Attack:
  // If we cannot load all units, don't load
  if (travelsByGround()) {
    break;
  }

  // Retreat when enemy units arrive, unless they are faster than us
  if (isEnemyAttackUnitsWithingSight()) {
    if (!isEnemyFasterThanTransport()) {
      setState(LoadUnits);
    }
  }
  // No enemies within sight, load if we aren't in the goal region
  else {
    if (!isTransportInGoalRegion()) {
      setState(LoadUnits);
    }
  }
  break;
}
```

Listing 4.8: Drop squad behavior

While the code explains it self, setState() does not, as it changes the behavior of the squad. `setState(LoadUnits)` loads all ground units and disables regrouping; `setState(Transport)` enables regrouping again; and `setState(Attack)` sets the transportation to unload all units and enables regrouping.

The Drop squad uses the same method as Attack squad to check if its goal has completed, although it rarely does as the goal of the drop is not to destroy

all buildings, merely to distract the enemy. In addition it has a timeout function which will make the drop retreat after 90* seconds, but only if it is outside the target region.

Now you know how it works, but which units will be used for the drop, as drops should not just use random units and hope for the best? It uses Unit compositions for this—section 4.7.7 describes unit compositions, all available unit compositions for drops squads can be found in listing A.5. In general the Commander will at random pick one of the drop compositions that it has units for and use it.

Drop squad movement The target location to drop is, as mentioned, required by the attack coordinator, but the transport does not move in a direct path to the location (once the units are loaded), instead it will move along the edge of the map illustrated in figure 4.6.

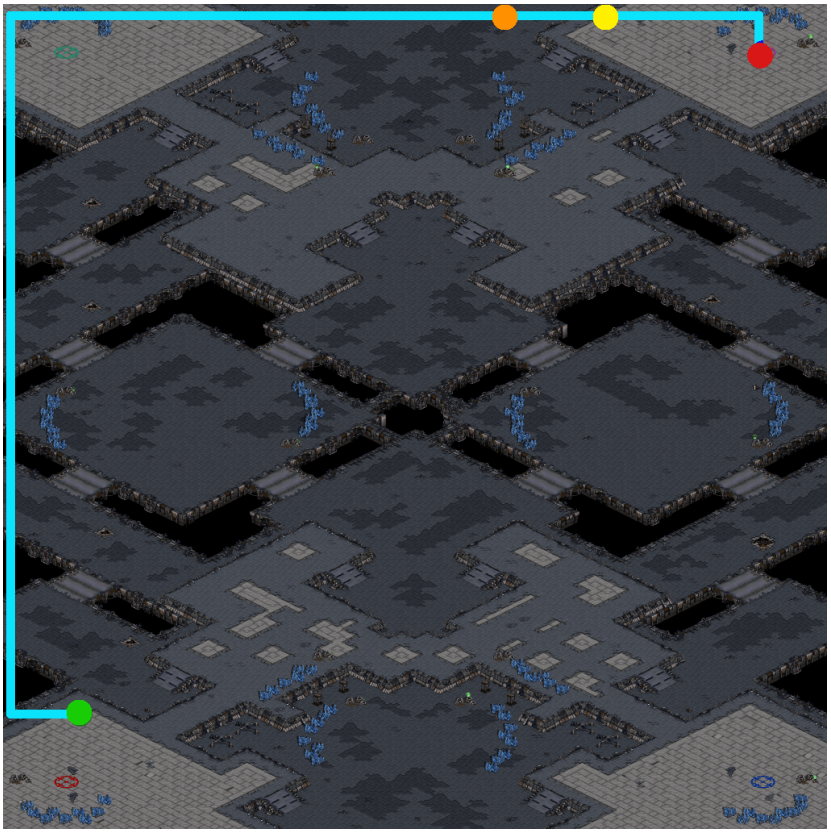

Figure 4.6: Example of Drop squad movement. (●) Start location where the transportation loads the units. (—) Path to the drop target. (●) Squad waits here, if it has Wait goals. (●) Where the squad starts to unload. (●) Target location to attack.

4.7.4 Patrol squad

Patrols units between a set of locations, in the current version of BATS it only patrols between our defend positions which the Defense manager takes care of. The units in the squad automatically attacks close enemies, but all units will not go to that location; instead another manager needs to tell the defensive squad what to do, e.g. the Defense manager will tell the squad where to defend. The squad will move to the defended area and attack all units within the defend perimeter (10^* tiles), but will remain in the defended area until no enemy units are within the enemy offensive perimeter (12^* tiles).

The Patrol squad will never succeed with its goal, as the goal is to patrol between the specified locations the entire game— although the locations usually change during the game.

4.7.5 Hold squad

The Hold squad has one goal, to hold a location from enemies; any location could be held, but it is designed to hold choke points. The squad contains a hold location and a roam location. The hold location works in more or less the same way as Patrol squad's defense location, i.e. if enemy units enter the defend perimeter (10^* tiles in radius) it will move and attack those units, as the squad always will defend here it does not bother checking if enemies are within the enemy offensive perimeter as the Patrol squad does. The roam perimeter (4^* tiles in radius) is where the units in the squad will move to and stay until enemy units move within the defensive perimeter.

Terran siege tanks have a special ability when their are in this squad and the bot has researched the siege mode ability. The siege tank will automatically siege up in the roaming area effectively defending the hold perimeter (and other close positions) from its location.

Hold squads use Unit compositions to know which units should be in the Hold squad—section 4.7.7 describes unit compositions in general and listing A.4 shows the composition file used for Hold squads. The current unit compositions used by BATS are listed below; the top is the composition with least priority, whereas the bottom is the one with highest priority. The name of the composition is in bold text.

Marine spotter 1 Marine
Bunker Marines 4 Marines
Marine Medics 4 Marines, 1 Medic
Tanks 1 1 Siege tank
Tanks 2 2 Siege tanks
Tanks 4 4 Siege tanks

The Marine spotter is used to watch for enemies as one marine alone is not a good defender. Bunker marines are used for bunker, this strategy does not work with the current version of BATS as one cannot bind a Hold squad to a specific location where there for example is a bunker. Marines and medics generally make a good composition, but these number are too low to hold of almost any attack, but they can still delay the attack until reinforcements can arrive, these are however not used in the experiment as none of the build orders contain medics. Tanks are very good positional holders and can hold of almost anything coming through choke points[30].

4.7.6 Scout squad

Scouts regions and expansions that have not been scouted for a long time, to be exact it will always move to the region or expansion that our team visited the longest time ago. The Exploration manager handles this and thus the Scout squad gets it target location from it. To avoid multiple scouts moving to the same location the scout location will set its last visit time when the squad gets the location, although it has not been visited at all.

When an enemy is seen close to the scout it will abandon its current scout location and find another one, the idea is to save the scout from being killed by the enemy. This, however, does not work correctly all times as the units will move away from the enemy and at the same time towards the goal, but if the new scout location happens to be on the other side of enemy the scout will just roam around the outer edge of the enemy. To be sure that the scout does not switch scout locations all the time when an enemy is present it will only switch once its close to any enemy, it will then need to get out of the range before it can switch goal again.

Further optimizations When another unit visits the target location the squad will not complete its goal until it has reached there itself. Currently the squad moves from one end of the map to the other (already passing recently scouted areas), an improvement would be to either let the squad get regions and expansions that abut to its current location (or more correctly improve Exploration Manager), or search for the most optimal path (in regard to visit not-recently visited locations) when moving to the target location—i.e. use its via path. The first option is probably preferable as it is easy to calculate and easy to check if the first limitation should be solved, otherwise one must check all via paths and the path might not be the optimal anymore if one location is removed.

4.7.7 Unit composition

In a nutshell, the unit composition is a feature for the squads to require specific types of units. E.g. Hold squads require specific unit groups (compositions) as

was explained above in section 4.7.5. All unit compositions used in bats can be seen in appendix A.2. The unit composition has two important elements: The priority and the units, higher value means higher priority, the units for the composition are listed under the units sub-section.

To use compositions one calls the unit composition factory which takes a composition type (hold-squad, drop, etc.) of and a list of units (usually free units). The unit composition factory will check which compositions can be created with the available units—the composition needs to be full—and then return these sorted by priority. It is then up to the caller of unit composition factory to decide which composition to use—for Hold squads it will always use the one with highest priority, drops on the other hand takes a random one.

4.7.8 Wait goals

Wait goals were created for simple one time trigger abilities. These can be added to any squad at the moment, but will only have an effect on the squad if the derived squad handles them, as the squad base class only checks if the Wait goal has succeeded, failed, or timed out and send an event to the derived class.

To be updated Wait goals shall be added to the Wait goal manager which updates the goals, completed goals (succeeded, failed, timed out) will automatically be removed from the Wait goal manager. When adding an Wait goal to the manager an optional set type can be set, this allows all Wait goals with the same set type to be easily extracted—Attack coordinator uses this when adding existing Wait goals to a new Attack squad.

Existing Wait goals For now there is only one type of Wait goal, WaitReadySquad. Attack squads uses these, as described in section 4.7.2, to synchronize all its attacks—i.e. to attack the enemy simultaneously from different locations. The Wait goal will complete when the Attack squad is in position, 25^* tiles from the attack location, when the squad is under attack or attacks, or if the Wait goal times out (30^* seconds).

4.8 Build planner

Purpose Build planner is the core for planning structure and army creations. Structure includes constructing structures, upgrades, add-ons and army refers to all combat and non-combat (e.g. Medic) units. The production process during the whole game is divided into three phases namely early, mid, and late. BATS follows specific build order provided in the configuration file (/BATS-data/buildorder/) and can be changed by the player, but was not during the experiments. The idea is that the player shall freely be able to change early, mid, and late game build orders to match his play during these phases, this can be done both before or

within the match (but not in the experiments as stated, as we thought this would be too much to think of).

Commands used The phase transition during the game is done by the command transition or transition BUILD_NAME The configuration in the transition file is overridden by the command transition BUILD_NAME.

How it works The priority of the production is the order in which the units are placed in the configuration file. The army is classified into *must have* and *percentage* units for each phase; must have units are always prioritized and in the order they were entered, when no more must have units need to be build it will instead try to maintain the number of percentage units.

A short example of the format is given below in listing 4.9.

```
<name>
Marine Medics
<description>
Builds marine and medics for a fast offense.
<build-order>
Terran_Supply_Depot
Terran_Barracks
Terran_Refinery
Terran_Academy
Terran_Factory
<units>
80\% Terran_Marine
20\% Terran_Medics
<must-have>
2 Terran_Medics
1 Terran_Vulture ; for scouting
```

Listing 4.9: Build order example file

<name> Name of the build order and its strategy.

<description> Description of the strategy.

<build-order> List of structures to be built during this phase, the structures are built in the order they are listed.

<units> Defines the army the current phase shall have and is specified in percentage. It will always prioritize the unit that is furthest away from its goal. For example using the values from the example build order, the current army consists of 74% marines and 18% medics. This will prioritize the medics, why? Marines are $74/80 = 0.925$ (7.5%) from their goal whereas medics are $18/20 = 0.9$ (10%) from their goal.

<must-have> Compulsory units that are to be built during a phase.

If a unit with the highest priority cannot be built because of a) low resources, wait for the resources to build and and do not build another unit; or b) no structures are free to build the unit, build the next unit in the priority list instead.

Destroyed structures and units are automatically readded to the build order, meaning they would be built again.

4.9 Communication

BATS uses the IntentionWriter to send messages to the teammate, a message consists of an intention together with an optional reason and ping location. Each intention and reason has multiple messages and are picked at random and put together to form a sentence with the intention first—if no reason is specified it will only use the intention as the whole message. The random function, however, is not entirely random, it will never pick the last used intention or reason message.

To not spam the teammate, messages with the same intention share a timeout, default set to 30^* in-game seconds; if the bot tries to send another message with the same intention before it has timed out IntentionWriter will skip the message. The timeout can be overridden by each intention, for example warning that BATS is under attack has a timeout of 60^* in-game seconds.

All messages and timeout settings are read from an ini-file to easily add, change, or remove existing messages, and to reload messages during gameplay. See appendix A.3 for the full ini-file with all intention and reason messages.

This simple communication system that picks two parts of the sentence at random and put them together works relatively well. A better approach would be to create specific messages for each intention and reason together, i.e. not randomize one from intention and one from reason, but create around 5 messages for each combination. This will create more tailored messages and will have better English, as this system has some limitations (described below). This was not implemented because it required much more sentences and thus time.

Limitations Because the intention message could be used with many reasons the intention needs to be generic and cannot be tailored to the situation. E.g. "Falling back", "I'm retreating", instead of "Fleeing from the enemy", while all work for the reason "because the enemy is too strong" the last intention will not work for "because you're retreating."

Many of the sentences may sound a bit strange and some could sound auto-generated. For example "I will attack, you're expanding" and "Leeeeroooooooy Jenkins, because I'm expanding"[2], but maybe not "I will attack, while you are expanding" or "Sending out an attack, because I'm going to expand."

[2]Leroy Jenkins is an Internet joke from World of WarCraft where Leroy was tired of waiting of his group and ran into several mobs (monsters) killing the entire group. In the beginning he yelled his name, thus this message.

Chapter 5

Methods

To evaluate our bot we conducted three evaluations and a final experimentation. The first evaluation was in the design phase of the bot; here we asked StarCraft players on forums[1] what they would like to see in a teammate bot.

Three weeks before the experiment the authors played with BATS to find major bugs to be fixed and minor improvements that it would benefit from. When the majority of the bugs was fixed, an additional tester evaluated the bot's control and behavior to find further improvements before the final experiment.

Player evaluation From the player evaluation we conducted that an improvement needed to be made to how commands were send. For one available commands were not visible to the player, but s/he had to remember them. This gave the idea to create GUI buttons for the player client. These can be seen in figure 5.1

5.1 Experiment method

The experiment is divided into three parts, first the tester preparation where the testers plays skirmish matches against the default bot to refresh his/her skills or familiarize him-/herself with StarCraft and the map the experiment will be conducted on. The testers started the preparation from a couple of days before the experiment to the day of the experiment.

The second part is the actually experiment itself. Here we present the tester with four scenarios: 1) no control over the bot and the bot would not communicate its intentions; 2) with control over bot, but no communication; 3) with communication, but no control; and 4) with both control and communication enabled. To avoid testers preferring a scenario depending on the order, i.e. the results depending on the order, all testers will play the scenarios in different orders. This will, however, have a side effect: tester that start with both communication and control might be overwhelmed with both having a teammate, check its messages, and be able to control it.

[1] Team Liquid: `http://www.teamliquid.net/forum/viewmessage.php?topic_id=308630` Battle.net: `http://eu.battle.net/sc2/en/forum/topic/3312961916`, accessed 2012-09-03.

Figure 5.1: Player client with GUI commands enabled

The scenarios are played at fastest game speed, both because normal speed is slow (the experiment would be too long) and it is considered a standard to play at fastest game speed in the StarCraft community. Each scenario is limited to 20 minutes—if matches were longer (an hour) the tester might be bored very quickly and would have to sacrifice a long time of their day. If the tester is winning after 20 minutes, the tester was asked if we should speed up the game or close it—the matches were sped up to roughly 10 times faster speed (depending on computer speed), thus the game usually ended within a minute or two. This was done to let the player win for the experiment to continue to be interesting.

Tester base The testers were all friends of us. The reasons we choose friends over strangers were: 1) we needed two computers as there is a bug that does not let you run the game on one computer unless you have installed a virtual machine; 2) the testers should be comfortable when playing, thus we tested the game at their homes where we also got the second computer; 3) the test was quite long, 1–3 hours of preparation and 2 hours in the experiment—we think it would be hard to get enough unknown players to be willing to play so long unless they were given some thanks money. 4) we did not live near campus (2.5 hours and 5 hours away) meaning it would be quite expensive to commute or burdensome to stay at friends' places for a week to test on students.

Although we were aware of possible negative effects such as their answers being biased on what we wanted to hear, we did not think this was enough to

invalidate them as a tester base. All the 4 scenarios included our bot, meaning they would not test against another bot. They did not know our thoughts or hypothesis before the experiment began and we tried to be as natural as possible not to mention our thoughts.

Gathering results Directly after the 4 scenarios the tester was given a questionnaire asking to grade how fun each scenario was in a scale from 1 (not fun) to 6 (very fun). The result from the questionnaire is evaluated using paired two-sample T-tests.

Instead of testing whether the player thinks a scenario is fun or boring, we could test if s/he wins or losses as one might think that winning is better than having fun. But the games we talk about is for having fun and that does not mean winning, as only winning is actually quite boring[21], and it is more fun to play when the player is challenges to play at his best[37].

After the questionnaire the testers were interviewed using a semi-structured interview method. The goal of this interview was to identify why they thought a scenario was more fun than the others, what they liked, disliked, and missed about the bot.

Bot evaluation questions These are the main questions we wanted to answer during the interview.

- Which scenario did you think was most fun?
- What did you like most with the bot?
- What did you dislike?
- What did you miss?
- What command did you like?
- What command did you dislike?
- What command did you miss?
- When (if) you lost, did you feel it was the bot's fault, your own, or both?
- When (if) you won, did you feel it was because of the bot, you, or both?
- Was there anything that surprised you?
- Other comments?

Experiment evaluation questions Some questions were asked about the experiment to get an overview how the testers perceived the experiment to be, and possible improvements for the next experiment.

- Was the experiment fun?
- Was the experiment too long?
- What did you think of the preparation? I.e. playing SC before the actual experiment.
- What improvements could be made to the experiment?

Chapter 6

Results

These are the results from the experiment on 14 people, 12 males and 2 females, ages 21–26, with skills varying from beginners who only played StarCraft a few times to those who have played more than 20 hours of skirmish matches in either StarCraft: Brood War or StarCraft 2.

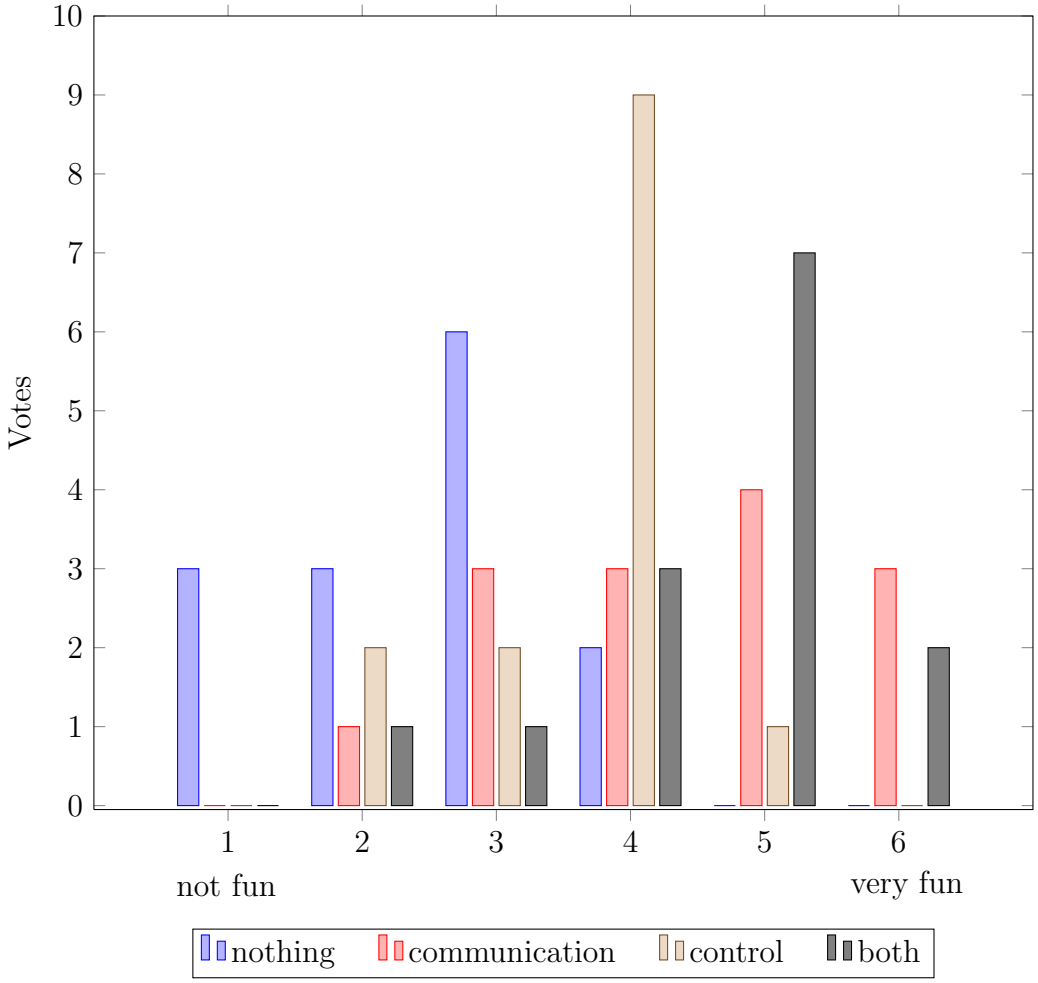

Figure 6.1: Fun degree of the four scenarios

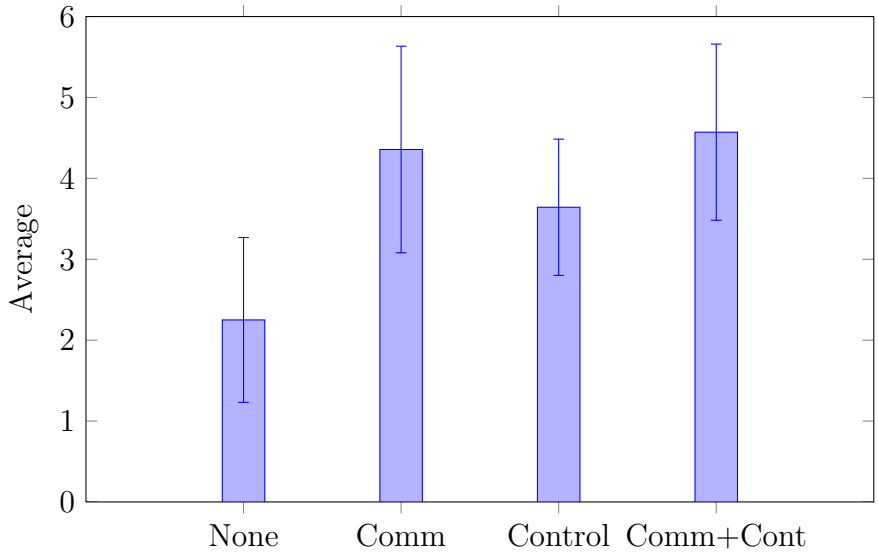

Figure 6.2: Average vote with standard deviation

6.0.1 RQ 1 Which type of bot is most fun to play with?

"Which is most fun to play with, communicating bot, controllable bot, both communicating and controllable bot, or none?" Figure 6.1 shows the questionnaire results from the four questions, one for each scenario; the results are summorized in figure 6.2 for an easier comparison.

To validate our hypothesis if the testers thought it was most fun to play (in this order) with a communicating and controllable, communicating, controllable, and a bot with none of these features we conducted paired two-sample T-tests, results found in table 6.1. These null hypotheses tests whether the differences between the scenarios are enough to deduct whether the order from figure 6.2 could be trusted.

	None	**Comm**	**Control**	**Comm+Cont**
None	N/A	0.0013	0.0029	0.0015
Comm	0.0013	N/A	0.1461	0.6301
Control	0.0029	0.1461	N/A	0.0094
Comm+Cont	0.0015	0.6301	0.0094	N/A

Table 6.1: Paired two-sample t-test

From table 6.1 we can see that there is not enough difference between neither communication + control and only communication nor only communication and only control. What we can deduct from this is the possible order of fun in table 6.2 where 1 is most fun and 4 is least fun.

An interesting observation was that some testers preferred communication

1	Communication + Control	Communication	
2	Communication + Control	Communication	Control
3	Communication	Control	
4	None		

Table 6.2: Possible fun order from the t-tests

over both control and communication. During the interviews we found a correlation between control and player skill—players with higher skill tended to prefer control whereas lower skill players became overwhelmed by all tasks, including the control of the bot.

Control & communication During the interviews 8 of 14 testers felt it was most fun to play with the bot when both communication and control was enabled. Two of the tester could not decide whether only communication or both communication and control was more fun, but one mentioned that s/he would liked control and communication more than only communication if s/he had been better at the game. Below you can find the testers' reasons why control and communication was most fun to play.

- I could build up my army, order him to attack and join his attack.
- Easier to plan because he told me his plans.
- I could order him to attack where I attacked using the follow command.
- You get a direct response from him after an order—you know that something will happen.
- Nice to be able to control him.
- It was fun that he explained his strategies.

Communication only 6 of 14 testers (including the two testers already mentioned that could not decide on whether only communication or both communication and control was most fun) thought it was most fun to play with the bot when it only had communication on. Below you can find the testers' reasons why communication was most fun to play.

- I already had a lot on my mind, controlling him was a bit too much for me.
- Easier for me to plan because he told me his plans.
- I am lazy therefore it was easier to let the bot do the hard work.

Control only 2 of 14 testers thought it was most fun to play only with control. Although these might not be credible as for one tester it was the only game s/he won in the test and s/he told us that it was only because of that it was most fun, otherwise s/he guessed it would be more fun with communication. The other tester only thought control was most fun as it was the first scenario s/he tested, otherwise s/he thought control and communication was most fun because "it was easier to play when I (the tester) can send attacks at the same time his and it gets easier to see what the bot does" (paraphrased), i.e. same reasons as other players who thought control and communication was most fun.

6.0.2 RQ 1.1 Which features are most liked?

Below the most liked features are shown in two lists, one containing general features and the other specific commands they liked. As mentioned these can serve as guidelines for future work what features shall be prioritized.

General liked features Most noteworthy is the communication which more than half of the testers commented on, more specifically "Fun to know what the bot is doing" and "Easier to plan because he told me his plans."

- Communication
  - He said why attacked, expanded, and so forth—I could learn from his strategies.
  - Fun to know what the bot is doing.
  - Easier to plan because he told me his plans.
  - Feels more like team play when he is communicating.
  - Direct response after ordering a command, then you knew if the bot would follow the command or not.
  - It had humor, e.g. Leeeerooooooy Jenkins message when attacking.
  - Almost like playing with a real player.
  - Asked player for help when under attack.
- Uses same strategies as humans, such as attacking when expanding.
- Very good macro, always has much more units than me.
- Own initiative to do things.
  - Followed me when I wanted to attack, even though I did not order him—feeling that I play with someone.
  - Very good at expanding, and often.

  - I could plan after how it acted and collaborate.

- Coordinated defense—came and defended me when I needed help.
- It did not run its own race, I felt involved in its decisions.
- Built tanks—I could back with my units and the tanks would clear the enemy units.
- Its attack cleared the path for me.

Liked commands

- Follow
  - Doubles the amount of army size at one place.
  - I did not have to attack by myself.
  - Easier to coordinate where he should attack.
- Attack
  - I do not have to attack at the same place with my army.
  - He could clear the path for me.
  - Easy to reinforce as one could spam the attack button.
  - He always had an army, if he attacked I could support him with my smaller army.
  - He decided where to attack and I could follow and attack somewhere close.
- Expand
  - I like to control the timing between when I attack and he shall expand.
  - I could order him to expand twice if the opportunity came, or just to expand when he did not notice he could expand.
- Scout
  - I do not have to scout and that is good as I am lazy.

6.0.3 RQ 1.2 Which features are most disliked?

Instead of asking what features the testers disliked, we asked what they disliked about BATS (notice the removal of feature). Below is a list of all things they disliked about BATS.

General disliked features Almost all testers mentioned they were annoyed by BATS halting its army in choke points making it impossible for them to pass.

- The army halted in choke points and made it impossible for me to come through.
- Got stuck when moving to a place, he ran back and forth until he finally ran more too long to the other side—note: regroup behavior.
- It did not follow me very well—note: when the squad it followed became very small (1-2 units) the player did not notice it had units that the bot followed with 30+ units.
- Bot asks for help when few units attack and he can easily handle it.
- Felt like it changed how good it was during each scenarios.
- He attacked cowardly, slow attack with siege tanks instead of just marching in and killing everything which he could do in most situations.
- Sends too many messages at the same time, maybe queue up messages instead?—note: sometimes the bot will decide to retreat, but decides to attack directly after and then expand because it is attacking, thus 3 messages are printed directly after each other which can confuse the player..
- Never had units for drops.
- When it dropped it did not function very well.
- A better scout that scouted bases and not random locations.

Disliked commands Two answered drop as it did not work properly, but most of the testers did not test all commands and thus could not answer if they disliked any command.

6.0.4 RQ 1.3 Which features are missing?

The testers were asked which features they missed during their scenarios, the first list presented includes general features whereas the second list are specific commands they missed. These can serve as guidelines for future developers what players want from a teammate bot.

General missed features

- More strategies, e.g. rush attacks, and would know about timings to make use of them.
- Use a variety of strategies depending on my build order.
- Give resources to the bot.
- Did not specify where he is under attack, would help me find the place.
- BATS could communicate what it thinks the player should do when s/he does not know what to do.
- More variety of units, more detection units.
- Scans in enemy bases.
- For the bot to play other races than Terran.

Missed commands, or improved existing commands

- A help button when the bot shall come and help me.
- Using way points for attacks to specify from where he should attack, either to not block my path or to create a more advanced attack.
- Drunken mode, random behavior.
- Command buttons grayed out when player could not use them.
- Retreat command.
- Abort command to abort the last command if the player misclicked.
- Hold position.

6.0.5 Other questions and comments

What surprised you in a positive way? A common misconception was that the testers would play against the bot and not with it. They thought BATS was a good bot, that built up his army fast and that the tester did not need to babysit it. They tended to like the funny messages, especially Leeeeeerroooooy Jenkins. Common for many testers were that the bot expanded just before they were going to send the expand command, which made them think the bot was good at expanding and had a good timing on expands. A tester liked the underlying reason behind each intention and not just the intention as s/he could learn from this. Another tester was surprised by that it worked so well to play with a bot as they can be quite stupid.

What surprised you in a negative way? More than half of the testers thought BATS crowded the places he attacked and made it impossible to bypass unless you had air units, 2 testers even changed their strategy to only air units because of this. He did not clear the entire base as humans would do, but left the base when few buildings were left. Twice the bot expanded to a location the player already expanded to, i.e. build a command center beside their command center (or equal building). Sometimes the bot followed a small army consisting of 1-2 units, this surprised the player as s/he thought it would either retreat or continue attacking on its own.

If you won or lost, did you think it was because of you, BATS, or both? This question was not asked until the 6th tester, thus only 8 of 14 testers were asked this question. Four of the tester thought they won because of both, where one of these thought it was because of him/her in one of the four scenarios. Their reasons were that BATS was better at building units, but they themselves had a better idea of strategy, where and how to attack. Two of the testers thought it always because of them as they felt better than BATS, these two were amongst the better players and also preferred control—a side note, there were two other players just as good as them, one of them thought it was because of both and strangely the other thought it was because of the bot. Another tester thought they won because of the bot as it had saved him/her at a very crucial moment.

Only one loss occurred for the last 8 tester. The tester who lost felt that it was entirely his/her fault as the bot had been attacked for 4 minutes and was defeated before the tester even noticed it, although s/he mentioned that if the bot had said it was under attack it would never have happened.

Commands used We noticed that many of the testers did not use all commands, no-one used the transition command as all felt the bot handled that fairly well. Only a few (2–4) used the drop, expand, and scout commands, with the same reason here as they felt the bot did a good job on this, except drop but they did not feel as a drop was needed.

Other noteworthy comments One tester mentioned the bot felt like Brother in Arms where you can control other units. Another tester mentioned that it was more fun to play with this bot than any other s/he had played with. A third tester thought BATS could be used either when one wanted to play a regular co-op game but did not have any friend to play with, or to fill an uneven team when playing with friends.

6.0.6 Questions and comments about the experiment

These are questions asked about the experiment itself and not directly associated with the bot.

Was the experiment fun or boring, and why? All testers thought it was fun to test the bot, both because StarCraft is a fun game and because it was a new experience for many to play with a bot that you could control and communicate.

Was the experiment too long? The experiment was estimated to take 2 hours. Some testers thought it sounded quite long, but none of the testers thought it was too long. Their reasons were that the scenarios were short (max 20 minutes) and they had fun testing the bot making time pass very quickly.

Other comments A tester commented how much difference there were between bots with and without communication, put in the testers own words: "I noticed how important the communication was and how bad something can be when a feature is missing."[1]

[1]Translated from Swedish "Jag märkte hur viktig kommunikationen var och hur dåligt någonting kan vara utan en feature."

Chapter 7

Conclusions and future work

First we go through our conclusions on the bot, what the players thought, liked and disliked about the bot and scenarios. Then we go through some lessons we have learned during the project we thought was worth mentioning, and last future work for researchers where we focus on three different areas: teammate bot behavior, communication, and control.

7.1 Bot conclusions

We conclude that a teammate bot for RTS games that communicates its intentions and reasons is indeed more fun to play with, although we did not find any statistical difference between Communication + Control and only Communication both of these include communication.

What surprised us was that some players preferred when the bot only communicated. This was later identified to depend on the player's skill where more experienced players liked the ability to control the bot whereas beginners tended to get overwhelmed by all the existing game action and thus did not want to focus on yet another task: controlling the bot. This is an interesting question on how to balance the control so that beginners aren't overwhelmed but at the same time meets the needs of a more skillful player. We talk more about this below in section 7.3.3.

The most liked feature was the ability for the bot to communicate its intentions and reasons to the player. The most reasons were "it was fun to know what the bot is doing" and "easier to plan because he told me his plans". This aligns with what Norman states, humans want to know what the AI is doing[28].

An important aspect BATS missed was point 5 from McGee's and Abraham's definition[27] of a real-time teammate bot, "whenever possible, prioritizing the player experience"—for the full list see section 2.1.1. When the player either ordered BATS to attack or follow him/her, BATS almost always ended up blocking the path for the player close to the attack location, not prioritizing the player experience as it should let the player through. Disrespecting this "rule" probably

made "The army halted in choke points and made it impossible for me to come through" the most disliked feature.

7.2 Project conclusions

Magnusson's notes Being a perfectionist is not easy, this was one of the causes why we did not meet our first deadline. I implemented features that were never used but what I thought were nice to have, such as the Wait Goals, advanced drop mechanics. If we remove Wait goals and drops entirely it would probably not affect the test results as Wait goals were never used and drops were barely used. In addition to this I changed existing systems much more than needed—instead of using the existing squad manager with squads I create a new squad manager, this was because the functionality differed quite much what we wanted to have in BATS, but maybe not needed. We would have saved some weeks on this task alone.

In addition I tweaked and fixed small thing that "it will only take two hours", but I found many of these small tweaks and some of them took more than two hours. After half the bot had been implemented and the deadline was close, we decided to decrease the number of features and aim for the next deadline, almost three months later. Now instead for fixing all small bugs and tweaks I created a ticket in our project manager, then I forced myself to not work on them until all the bot's main features were implemented. This worked much better and the bot was done, maybe not in no time, but much faster.

In essence what I learned was that task prioritization is a must if one wants to finish a project, and then only work on those assigned tasks, if new tasks were found one shall not do them directly but either put them in the right priority or put them in a bag of things to do if there is time left.

7.3 Future work

We will split the section in three parts, first discussing future work for the behavior of a teammate bot, then improvements on the communication, and finally the control of a teammate bot. While these are RTS game specific topics many of the Communication and Control topics can be slightly changed to match another game genre.

7.3.1 Teammate bot behavior

BATS behaves in ways that we thought would be good, both through own experience, research, and testing; but it still lacks a solid behavior to be used in a commercial game as its behavior does not change depending on the play style of the human player. More research needs to be done on what players want a

teammate bot to do and how much initiative the bot shall have. If the bot can be controlled the degree of initiative can depend on how much the player want to control the bot, i.e. the more control the player wants the less initiative the bot shall have, or does the players, independent of skill and play style, always want the same initiative?

A possibility for teammate bots can be either a full-time or temporarily replacement for a human player in online matches—temporarily when the player disconnects. But how shall the bot behave when it is used as a replacement for the entire game? It needs to be balanced so it either matches the human players' skills in its own team or by setting a predefined skill to either increase or decrease the entire team's total skill. When it acts as a temporarily replacement the players might want it to continue using the same strategy as the player and not do any hasty actions that could make the player loose, the bot could use some learning algorithm that always is on when the player plays matches and then send out this information to the other clients if the player gets disconnected, one of the clients could then take temporary control of the player continue play as nothing happened. For both scenarios it might be good to do player modeling for the teammate bot to know how it shall play, further resources on player modeling for RTS games can be found here[4, 25, 26, 33, 38].

7.3.2 Communication

The messages from BATS are very simple and aimed to feel like less auto-generated messages. In case of text messages the player might not want long "human-like" sentences, but instead short sentences.

There is also the possibility to either synthesize the message or record own messages to see if the player prefer messages that are read to him/her. Reading messages out loud might be good but cannot be processed as quickly by the player and it can be hard if the computer want to send messages very often, meaning some research needs to be done on how often and what type of messages shall be sent for the player not to be overwhelmed by messages—this type of research can also be done on text messages.

7.3.3 Control

We concluded that not all commands were used and players preferred control depending on their skill. One could research which type of commands are preferred and how often they are issued. This could then serve as another research where more control functionality is either displayed for a player depending on his/her skill level, for example in campaigns the game introduces new units for each mission and could in a similar way introduce more commands throughout the entire campaign.

Another experiment could be made how much level of detail the player wants over the commands. E.g. shall attack make the bot attack at some place (as it does now), shall the player have the ability to specify where it shall attack, shall the player have the ability to specify which path the attack shall use, or where the attack shall wait before it reaches its destination for a simultaneous attack? This level of degree probably depends on the skill of the player, but as you might have thought of now this can cause the player to simply play two players, but s/he only needs to control one bot. This leads to yet another research for a new genre.

Research a new type of RTS genre where the player simply tell the bot what to do, i.e. the player control the bot entirely but has setup some rules what it should build and how it behaves. This would be more like a strategy/tactic game where the player neither needs good macro nor good micro, but good strategy and tactics.

Mentioned in behavior, the possibility exists for acting as a player replacement in a team with more than 2 players. In regard to control, who shall control the bot, or can anyone in the team control the bot? Anytime?

Appendix A

BATS configuration files

A.1 Build orders used

These are the build orders that were used in the experiments.

```
<name>
One factory fast expand, siege mode

<description>
Starts building marines in the beginning and continues
with building two factories, tanks and siege mode
for the tanks. After siege mode has started it will
build an expansion.

<build-order>
Terran_Supply_Depot
Terran_Barracks
Terran_Refinery
Terran_Factory
Terran_Machine_Shop
Terran_Factory
Terran_Machine_Shop
TechType: Tank_Siege_Mode
Terran_Command_Center

<units>
60% Terran_Siege_Tank_Tank_Mode
40% Terran_Vulture

<must-have>
8 Terran_Marine
2 Terran_Siege_Tank_Tank_Mode
```

Listing A.1: Early game build order

```
<name>
Vulture and tanks

<descriptions>
```

```
Focuses on attacking with vultures and tanks.

<build-order>
Terran_Starport
Terran_Control_Tower
Terran_Academy
TechType: Tank_Siege_Mode
Terran_Factory
Terran_Machine_Shop
Terran_Armory
TechType: Spider_Mines
UpgradeType: Ion_Thrusters
UpgradeType: Terran_Vehicle_Weapons 1
Terran_Armory
Terran_Science_Facility
TechType: Cloaking_Field
UpgradeType: Apollo_Reactor
UpgradeType: Terran_Vehicle_Weapons 2
UpgradeType: Terran_Vehicle_Plating 1

<units>
60% Terran_Vulture
40% Terran_Siege_Tank_Tank_Mode

<must-have>
2 Terran_Wraith
1 Terran_Dropship
```

Listing A.2: Mid game build order

```
<name>
Tank drops.

<description>
Will focus on tanks, vultures, anad goliaaths for
direct attacks and use drops with tanks as distracting
attacks.

<build-order>
UpgradeType: Terran_Vehicle_Weapons 3
UpgradeType: Terran_Vehicle_Plating 2
Terran_Factory
Terran_Factory
Terran_Machine_Shop
Terran_Machine_Shop
Terran_Starport
UpgradeType: Terran_Vehicle_Plating 3
UpgradeType: Charon_Boosters
Terran_Factory
Terran_Machine_Shop
Terran_Starport
```

```
<units>
30% Terran_Vulture
40% Terran_Siege_Tank_Tank_Mode
30% Terran_Goliath

<must-have>
4 Terran_Wraith
3 Terran_Dropship
```

Listing A.3: Late game build order

A.2 Unit compositions

All the different unit compositions used in BATS.

```
; ---------- Marine Spotter ----------
[marine-spotter]
Priority = 1

[marine-spotter.units]
Terran_Marine = 1


; ---------- Bunker Marines ----------
[bunker-marines]
Priority = 2

[bunker-marines.units]
Terran_Marine = 4


; ---------- Marine Medics ----------
[marine-medics]
Priority = 3

[marine-medics.units]
Terran_Marine = 4
Terran_Medic = 1


; ---------- 1 TANK ----------
[tanks-1]
Priority = 4

[tanks-1.units]
Terran_Siege_Tank_Siege_Mode = 1

; ---------- 2 TANKS ----------
[tanks-2]
Priority = 5

[tanks-2.units]
Terran_Siege_Tank_Siege_Mode = 2

; ---------- 2 TANKS ----------
[tanks-4]
Priority = 6

[tanks-4.units]
Terran_Siege_Tank_Siege_Mode = 4
```

Listing A.4: Unit compositions for Hold squads

```
; ---------- MARINES ----------
[marines]
Priority = 4

[marines.units]
Terran_Dropship = 1
Terran_Marine = 8


; ---------- MARINE - MEDICS ----------
[marine-medics]
Priority = 6

[marine-medics.units]
Terran_Dropship = 1
Terran_Marine = 6
Terran_Medic = 2


; ---------- TANK - MARINE ----------
[tank-marine]
Priority = 2

[tank-marine.units]
Terran_Dropship = 1
Terran_Marine = 4
Terran_Siege_Tank_Siege_Mode = 1


;---------- TANK - SCV - MARINE (2 DROPSHIPS) ----------
[tank-scv-marine]
Priority = 5

[tank-scv-marine.units]
Terran_Dropship = 2
Terran_Marine = 6
Terran_SCV = 2
Terran_Siege_Tank_Siege_Mode = 2
```

Listing A.5: Unit compositions for Drop squads

```
[scv]
Priority = 1

[scv.units]
Terran_SCV = 1

[Marine]
Priority = 2
```

```
[Marine.units]
Terran_Marine = 1

[vulture]
Priority = 3

[vulture.units]
Terran_Vulture = 1

[wraith]
Priority = 5

[wraith.units]
Terran_Wraith = 1
```

Listing A.6: Unit compositions for Scout squads

A.3 Intention and reason messages

All intention and reason messages that could be used during a game. Again, it will use a random message from intention and reason.

```
[config]
; How many seconds must pass until the same intention can be used
; again. This can be overriden by each intention by specifying
; interval_min = seconds in the intention or reason.
default_intention_interval_min = 30.0


; ---------------------- INTENTIONS ----------------------
[intention.AlliedAttackFollow]
message = "I will join up with you"
message = "I'll come and help with you attack"
message = "I'll meet upp with your attack"
message = "your wish is my command, at least attack command"
message = "wait for me, I'll join the attack"

[intention.AlliedAttackFollowNot]
message = "sorry, will not join you"
message = "I can't join you right now"
message = "sorry, I can't join you"
message = "can't join you at the moment"

[intention.BotAttack]
message = "I feel like attacking"
message = "Leeeeroooooooooy Jenkins"
message = "I will attack here"
message = "I am going to attack here"
message = "now is the time to attack"
message = "sending out an attack"
message = "I will attack the enemy"
message = "I am attacking"

[intention.BotAttackMerged]
message = "added more units to the attack"
message = "reinforcements sent to the attack"
message = "reinforcing the attack"
message = "the attack was reinforced"

[intention.BotAttackMergedNot]
message = "I will not add more units"
message = "can't add more units to the attack"
message = "units the attack will not add"
message = "cannot reinforce the attack"

[intention.BotAttackNewPosition]
message = "finding a new place to attack"
message = "I will attack in a new place"
```

```
message = "changed attacking spot"
message = "I found a new attack spot"

[intention.BotAttackNot]
message = "I cannot create a new attack"
message = "a new attack is not possible now"
message = "not possible for me to attack now"
message = "I will not attack now"
message = "sorry, can't attack right now"

[intention.BotComingToAid]
message = "I will help you"
message = "hang on, I will be there in a sec"
message = "shit, I'll come and help you"
message = "I will send my army to you"
message = "wait for me, I'll help you"

[intention.BotComingToAidNot]
message = "sorry, I can't come and help you"
message = "I can't help you atm"
message = "shit, I cannot help"
message = "sorry, you're on your own"
message = "shit, you have to deal with that yourself"
message = "oh no, I can't help you"

[intention.BotDrop]
message = "I will drop here"
message = "drop incoming"
message = "i hope the enemy likes my drop present"
message = "i will send a distracting drop"
message = "drop aquired"

[intention.BotDropNewPosition]
message = "finding a new position to drop"
message = "hey, changing position for my drop"

[intention.BotDropNot]
message = "I can't create a drop right now"
message = "drop not available"

[intention.BotExpand]
message = "I'm getting ready to expand"
message = "I will expand here"
message = "hey, I will start an expansion soon"
message = "I'm gonna expand"
message = "expansion coming up soon"

[intention.BotExpandNot]
message = "I can't expand right now"

[intention.BotNeedHelp]
```

```
interval_min = 60
message = "I need help"
message = "help me"
message = "help"
message = "hey, can you help me"
message = "need some help here"

[intention.BotRetreat]
message = "falling back"
message = "retreating"
message = "I will retreat for now"
message = "I will retreat back to my base"
message = "falling back to base"
message = "hey, I will retreat"

[intention.BotScout]
interval_min = 240
message = "I will scout around a bit"
message = "scouting for the enemy"
message = "hey, I will send out a scout"
message = "I'll check where the enemy is"

[intention.BotScoutNot]
message = "I cannot scout at the moment"
message = "cannot scout now"
message = "I will not scout now"

[intention.BotTransitionLate]
message = "I'm transitioning to the late game now"
message = "I will be in late game now"
message = "I will transition to late game"
message = "hey, I transition transition to late game now"

[intention.BotTransitionMid]
message = "I will transition to the mid game now"
message = "hey, I'm going to the mid game"
message = "I will start mid game structures and units now"

[intention.BotTransitionNot]
message = "I can't transition any more"
message = "No available transitions"
message = "It's not possible to transition any more"

[intention.WeShouldRetreat]
message = "it's probably good if we retreat"
message = "now is a good time to retreat"
message = "hey, I think we should retreat"
message = "hey, it might be good to fall back"


; ---------- REASONS ----------
```

```
[reason.AlliedAttackDisappeared]
message = "I lost track of your attack"
message = "because you wondered of somewhere"
message = "had a hard time keeping up with you"
message = "you disappeared from my sight"

[reason.AlliedAttackNotExist]
message = "there is no attack to join"
message = "I can't find an attack to follow"
message = "because you don't have a squad"

[reason.AlliedExpanding]
message = "while you are expanding"
message = "because your setting up an expansion"
message = "I see you're building an expansion"
message = "because you're expanding"
message = "you're expanding"

[reason.AlliedMovingToAttack]
message = "because you're moving out to attack"
message = "while you're attacking"
message = "it seems you're moving to attack"
message = "because I think you're going to attack"

[reason.BotAttacking]
message = "because I am attacking"
message = "the attack will cover it"
message = "because I am attacking the enemy"
message = "my attack will cover it"
message = "because my attack will lure the enemy"

[reason.BotAttackSuccess]
message = "because my attack was successfully completed"
message = "my attack goal was completed"
message = "because my attack completed"
message = "I have destroyed the specific enemy buildings"
message = "because close enemy buildings were destroyed"

[reason.BotDidNotAttack]
message = "because I did not attack anything"
message = "I did not attack"
message = "because I did not find any enemy"
message = "I did not find any enemy"
message = "because I did not attack an enemy"

[reason.BotDropTimedOut]
message = "because I could not drop anywhere"
message = "because the drop took too long time"
message = "because I could not drop"
message = "it took too long time to drop"
message = "I could not drop"
```

```
[reason.BotExpanding]
message = "because I'm expanding"
message = "it's good when I'm expanding"
message = "because I'm going to expand"
message = "because I will expand"
message = "because I will setup a new expansion"

[reason.BotExpansionRunningLow]
message = "one of my expansions are running low"
message = "an expansion is soon out of minerals"
message = "because I'm an expansion is low on minerals"
message = "all minerals in one expansion are soon depleted"
message = "almost no minerals left in one expansion"

[reason.BotExpansionsSaturated]
message = "because my expansions are saturated with workers"
message = "the mineral patches are saturated"
message = "because I have enough workers on my bases"
message = "because the mineral patches are saturated"
message = "because all expansions are saturated"

[reason.BotHighOnResources]
message = "because my resources are high"
message = "I had too much minerals and gas"
message = "I had plenty of resources"
message = "because my resources was sky-rocketing"
message = "I had plenty of minerals and gas"

[reason.BotIsUnderAttack]
message = "because the enemy is attacking me"
message = "fending off an attack here"
message = "because I'm pinned down by the enemy"
message = "the enemy is in my base"
message = "my base is under attack"
message = "because I'm under attack"

[reason.BotNotEnoughUnits]
message = "I don't have enough units for that"
message = "little low on units right now"
message = "because I don't have enough units"
message = "because I am low on units"

[reason.BotTooManyAttacks]
message = "I have too many attacks already"
message = "because I can't handle more attacks"
message = "because I have too many attacks"
message = "too many simultaneous attacks"

[reason.BotTransitionNoMore]
message = "I'm already in the late game"
```

```
message = "late game is the last transition"
message = "I'm already at the last transition"

[reason.BotUpgradeSoonDone]
message = "because an ugrade is almost done"
message = "my upgrade is soon done"
message = "I have an uprade that will finish soon"
message = "because my upgrade will finish soon"
message = "because an upgrade is soon done"

[reason.EnemyTooStrong]
message = "enemy is too strong"
message = "because I will die if I stay here"
message = "because the enemy is too strong"
message = "enemy has too many units"
message = "enemy force is too big"
message = "because the enemy force is too strong"
```

Listing A.7: Intention and reason messages ini-file

Appendix B

Doxygen documentetanion

Since the doxygen documentation would be several hundreds of pages and be hard to search for, we refer to the online documentation found at: `http://bats.senth.org/doxygen/`. If the page is down and you want the documentation, please send an email to matteus.magnusson@gmail.com and he will send you a new working link.

Bibliography

[1] Aswin T. Abraham and Kevin McGee. AI for dynamic team-mate adaptation in games. In *Proc. IEEE CIG 2010 IEEE*, pages 419–426. IEEE, 2010.

[2] Dan Adams. The State of the RTS, 2006. http://pc.ign.com/articles/700/700747p1.htm, accessed 2012-04-06.

[3] Brian Ashcraft. *Why Is StarCraft So Popular in Korea.* Kotaku, 2010.

[4] Sander C.J. Bakkes, Pieter H.M. Spronck, and H. Jaap van den Herik. Opponent modelling for case-based adaptive game AI. *Entertainment Computing*, pages 27–37, 2009.

[5] BioWare. Mass Effect, 2007. http://masseffect.bioware.com/, accessed 2012-04-05.

[6] Blizzard Entertainment. StarCraft: Brood War, 1998. http://us.blizzard.com/en-us/games/sc/, accessed 2012-04-05.

[7] Blizzard Entertainment. StarCraft Manual, 1998.

[8] Blizzard Entertainment. WarCraft 3: The Frozen Throne, 2003. http://us.blizzard.com/en-us/games/war3/, accessed 2012-04-05.

[9] Blizzard Entertainment. StarCraft 2: Wings of Liberty, 2010. http://us.blizzard.com/en-us/games/sc2/, accessed 2012-04-05.

[10] BWAPI, 2011. http://code.google.com/p/bwapi/, accessed 2012-09-12.

[11] BWTA, 2011. http://code.google.com/p/bwta/, accessed 2012-09-12.

[12] Capcom. Resident Evil 5, 2009. http://www.residentevil.com/5/, accessed 2012-04-05.

[13] Capcom. Lost Planet 2, 2010. http://www.lostplanet2game.com/, accessed 2012-04-05.

[14] CIG StarCraft RTS AI Competition, 2011. http://ls11-www.cs.uni-dortmund.de/rts-competition/starcraft-cig2011, accessed 2012-04-06.

[15] AIIDE Starcraft AI Competition. AIIDE Starcraft AI Competition. http://webdocs.cs.ualberta.ca/ cdavid/starcraftaicomp/index.shtml, accessed 2012-04-06.

[16] EA Los Angeles. Command & Conquer: Red Alert 3, 2008. http://www.commandandconquer.com/en/games/bygameid/ra3/, accessed 2012-04-05.

[17] Electronic Arts. FIFA, 2011. http://www.ea.com/soccer/, accessed 2012-04-05.

[18] Lindsay Fleay. The Historical RTS List, 2012. http://www.rakrent.com/rtsc/rtsc_gameslist2.htm, accessed 2012-04-05.

[19] Gearbox Software. Brother in Arms: Road to Hill 30, 2005. http://www.gearboxsoftware.com/games/bia/, accessed 2012-04-05.

[20] Johan Hagelbäck. BTHAI, 2011. http://code.google.com/p/bthai/, accessed 2012-08-12.

[21] Johan Hagelbäck and Stefan J. Johansson. Measuring player experience on runtime dynamic difficulty scaling in an RTS game. In *Proc. IEEE CIG 2009*, pages 46–52. IEEE, 2009.

[22] Ryan Houlette. Player Modeling for Adaptive Games. In *AI Game Programming Wisdom 2*, pages 557–566. Charles River Media, 2003.

[23] Infinity Ward. Call of Duty: Modern Warfare 2, 2009. http://modernwarfare2.infinityward.com, accessed 2012-04-05.

[24] Inofficial StarCraft wiki, 2012. http://starcraft.wikia.com/, accessed 2012-09-05.

[25] T.J.A Jansen. Player Adaptive Cooperative Artificial Intelligence for RTS Games. B.Sc. thesis, Universiteit Maastricht, 2007.

[26] Froduald Kabanza, Philipe Bellefeuille, Francis Bisson, Abder R. Benaskeur, and Hengameh Irandoust. Opponent behaviour recognition for real-time strategy games. In *AAAI Workshops*. AAAI Publisher, 2010.

[27] Kevin McGee and Aswin T. Abraham. Real-time team-mate AI in games: a definition, survey, & critique. In *Proc FDG 2010*, pages 124–131. ACM, 2010.

[28] Donald A. Norman. *The Design of Future Things*. Basic Books, 2007.

[29] Open Real-Time Strategy, 2010. http://skatgame.net/mburo/orts/, accesses 2012-04-05.

[30] Sean Plott. Day[9]. http://day9.tv, accessed 2012-09-15.

[31] Zhou Pucheng and Shen Huiyan. Multi-agent cooperation by reinforcement learning with teammate modeling and reward allotment. In *Proc. IEEE FSKD 2011*, pages 1316–19. IEEE, 2011.

[32] John Reynolds. Team member AI in an FPS. In *AI Game Programming Wisdom 2*, pages 207–215. Charles River Media, 2003.

[33] Frederik Schadd, Sander Bakkes, and Pieter Spronck. Opponent modeling in real-time strategy games. In *Proc. GAME-ON 2007*, pages 61–68. EUROSIS-ETI Publications, 2007.

[34] Bhuman Soni and Philip Hingston. Bots trained to play like a human are more fun. In *Neural Networks, 2008. IJCNN 2008*, pages 363–369, june 2008.

[35] Spring engine, 2012. http://springrts.com, accessed 2012-04-05.

[36] Ensemble Studios. Age of empires 3, 2005. http://www.ageofempires3.com/, accessed 2012-04-05.

[37] Penelope Sweetser and Peta Wyeth. GameFlow: a model for evaluating player enjoyment in games. *ACM CIE Volume 3 Issue 3*, 2005.

[38] Gabriel Synnaeve and Pierre Bessiere. A bayesian model for opening prediction in RTS games with application to StarCraft. In *Proc. IEEE CIG 2011*, pages 281–288. IEEE, 2011.

[39] Ubisoft Montreal. Tom Clancy's Rainbow Six Vegas 2, 2008. http://www.ubi.com/US/Games/Info.aspx?pId=6243, accessed 2012-04-05.

[40] Wargus, 2011. http://wargus.sourceforge.net/, accesses 2012-04-05.