NTNU – Trondheim
Norwegian University of
Science and Technology

Micromanagement in StarCraft using Potential Fields tuned with a Multi-Objective Genetic Algorithm

Jørgen Bøe Svendsen
Espen Auran Rathe

Master of Science in Computer Science
Submission date: June 2012
Supervisor: Pauline Haddow, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Espen Auran Rathe and Jørgen Bøe Svendsen

Micromanagement in StarCraft using Potential Fields tuned with a Multi-Objective Genetic Algorithm

Master thesis, Spring 2012

Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Computer and Information Science

Sammendrag

Denne masteroppgaven presenterer en måte for å kontrollere Micromanagement i Real-Time Strategy (RTS) spill ved bruk av Potential Fields (PF) som er optimisert ved hjelp av Multi-Objective Evolutionary Algorithms (MOEA), nærmere bestemt Non-dominated Sorting Genetic Algorithm (NSGA-II). Den klassiske RTS tittelen *StarCraft: Broodwar* er brukt som testplattform på grunn av sin status i AI miljøet, den detaljerte informasjonen som er tilgjengelig fra tidligere forskning og prosjekter, og open-source rammeverket Brood War Application Programming Interface (BWAPI). Det foreslåtte AI'et kontrollerer sine enheter ved å plassere flere forskjellige Potential Fields på slagmarken. Vektene som brukes bak PF'ene sine kalkulasjoner er optimisert ved bruk av NSGA-II. Dette arbeidet er et forsøk på å forbedre tidligere metoder som er gjort med PF i RTS. Resultatene indikerer at Multi-Objective Optimization er en egnet metode for å optimisere PF i RTS.

Abstract

This thesis presents an approach to controlling Micromanagement in Real-Time Strategy (RTS) computer games using Potential Fields (PF) that are tuned with Multi-Objectve Optimized Evolutionary Algorithms (MOEA), specifically the Nondominated Sorting Genetic Algorithm (NSGA-II). The classic RTS title *StarCraft: Broodwar* has been chosen as testing platform due to its status in the competitive AI scene, the amount of detailed information available from previous research and projects, and the free open-source framework Brood War Application Programming Interface (BWAPI). The proposed AI controls its units by placing several types of Potential Fields onto the battlefield. The weights behind the PFs' calculations are optimized using NSGA-II. This work is an attempt to improve on previous methods done with PF in RTS. The results indicate that Multi-Objective Optimization is a suited method for optimizing Potential Fields in RTS games.

Acknowledgements

We would like to thank our supervisor, Pauline Haddow, for her steady council and feedback throughout this year, even in busy times. We also want to thank her for her open-mindedness about this project, and always being able share her expert knowledge in our discussions.

A special mention also goes to Yan Yi Look, who took the time to review our report despite having summer break.

Espen Auran Rathe and Jørgen Bøe Svendsen

Trondheim, June 11, 2012

Contents

List of Figures

Chapter 1

Introduction

StarCraft: Brood War is one of the most popular real-time strategy (RTS) games in the world. Since its release in 1998 it has gained critical acclaim and has been an important part of the electronic sports [1] scene. In South-Korea the game is still widely played, and at a point the competitive scene was so big that StarCraft matches is shown on Korean cable TV. This long term popularity has resulted in numerous patches being released over the years, making StarCraft a very balanced game. Balanced means that the different races in StarCraft (there are three) are equal in strength. This balance and popularity has caused a string of AI competitions to be aimed at the StarCraft platform. Inspired by these events, most notably the 2010 conference Artificial Intelligence and Interactive Digital Entertainment (AIIDE) [2], which first featured StarCraft in 2010, prompted the desire to explore the possibilities of this domain. The goal of this thesis is to effectively utilize Evolutionary Algorithms (EA) in an Artificial Intelligence (AI) for StarCraft: Brood War.

1.1 Background Project

To prepare for this master thesis a specialization project was written the semester prior. The project was to design an architecture for a StarCraft AI to be used in the master thesis. Six groups, composed by a total of twelve students, cooperated on the literature review part of this task, which was a large undertaking and a total of 400 articles were considered. The domains of AI, architectures and RTS games were explored and to do this

[1] Electronic sports is the general term for competitive play of video games. Other terms are e-sports, pro gaming and cybersports.

[2] Organized by the RTS Game AI Research Group at the university of Alberta http://webdocs.cs.ualberta.ca/ cdavid/starcraftaicomp/

in a thorough way the groups used a technique called structured literature review (SLR). In short the SLR technique works like this: first use a set of terms in various search engines to find a set of articles. These articles are then filtered several times with increasingly complex techniques, so that irrelevant articles are rejected early in the process to save time. The final set of articles was very broad and resulted in unique architectures for each group, designed based on the articles.

Some of the articles found during the SLR were used in this master thesis, and the rich domain knowledge gained has been useful to understand how best to approach our problem. The architecture to be used in this thesis was designed in collaboration, and the implementation was done by the other group. The architecture is further explained in Section 5.3.

1.2 StarCraft

This section is dedicated to describing the basics of StarCraft, and will assume no prior knowledge of the game.

1.2.1 General Rules

In StarCraft you have a birds eye view of the battlefield, and you can select and control your buildings or army by issuing commands such as e.g. moving an individual soldier (hereafter referred to as a *unit*) of your army to certain locations, or making a certain building produce a tank. These buildings and units have hit points (HP), which is the amount of damage they can take before being destroyed and removed from the game.

The game is won by defeating your opponent, which is much like a typical war scenario where armies defend and attack until one part is overpowered. Victories usually occur when a player concedes, which can be done at any point in the game if a player decides his chances of winning are unrealistic. The only other way of making Starcraft declare a winner is to destroy all the buildings a player has until he has none left. There are three races in Starcraft that players can choose to play as: Terran, Protoss and Zerg. These all have different buildings and units available, and their strengths and weak points differ. These differences are further elaborated in Subsection 1.2.2.

1.2.2 The Three Races

The story of StarCraft is set in space far in the future and centers around the three races Terran, Zerg and Protoss. They fight for dominion over

the galaxy, which is the perfect excuse for staging countless battles in an RTS game. While having very different strengths and weaknesses the three races are viewed as being balanced and are equally favoured when playing a match. However it requires skill when learning how to play each of the races well, and knowing one race do not automatically make you a good player in another race.

The differences and similarities are many, but the most characteristic differences are explained in the following paragraphs and shown in Figure 1.1.

Terran The Terran is a race of humans capable of travelling through space. The relative strength of Terran units are placed between Protoss and Zerg units, and is regarded as an easy race to learn. The other two races require more knowledge and skill to win a match. Terran have good defensive measures and can more safely than the other races defend the base and amass a huge army before attacking. Other special features of the Terran race is that their buildings can be moved and their gatherers can repair certain units and all buildings.

Zerg The Zerg is a race of insectoids, whose main interest is taking over and devouring everything in the galaxy. The relative strength of the Zerg units are the weakest and thus they have to employ different tactics than the Terran race. As a counter to their weaker unit they are able to produce a wast amount of units in a short period of time, quickly replenishing their army. Some Zerg units are able to hide underground, and they can also have the advantage of superior speed. These possible advantages are decided by what upgrades that have been completed.

Protoss The Protoss is a race of aliens with a high intellect and psionic powers. They have the strongest units, however they also have the highest cost. All Protoss units have shields that have to be taken down (by doing damage to the unit) before the Hit Points (HP) of the unit can be deplenished. These shields regenerate and is one of the reasons Protoss units are quite fearsome.

1.2.3 Resources and Economy

Buildings and units can be built or trained respectively using resources which are gathered at predefined spots on each map. Each race has a building where resources can be deposited, called the *main building*. The main building of each race is able to train units capable of harvesting resources.

	Terran	Zerg	Protoss
Unit Strength	Medium	Weak	Strong
Unit Cost	Medium	Cheap	Expensive
Production speed	Slow	Fast	Medium
Defensive capabilities	Strong	Weak	Medium

Figure 1.1: A comparison of the three races.

Figure 1.2 shows the unit training menu for the main building, notice it can only train one type of unit (shown in the top left corner). Each race has a different name for these units (Zerg has *drones*, Terran has *SCVs* and Protoss has *Probes*), but their functions are identical, so the general term for them is simply *workers*. These workers are able to gather resources and depositing them in the main building, thereby earning the player resources for him to spend.

The two types of resources are called *minerals* and *vespene gas*. These can be gathered from points on the map called *mineral patches* and *vespene geysers* respectively as shown in Figure 1.2. The main base acts as a collection point and is typically placed by the player as close as possible to the mineral and vespene gathering points to decrease the travelling distance of the worker unit.

The more workers you have gathering resources the higher the gathering rate you have, but only up to a certain point. The reason for this is that each resource point can only be harvested by one worker at a time, so the optimal amount of workers would be when a resource point is constantly being harvested from by some worker, and there is no queue for workers when they come to harvest. When this optimal amount of workers is achieved the mineral patch or vespene geyser is *saturated*. When all all mineral patches and vespene geyesers at a base is saturated the base is said to be saturated.

Each player starts with one main base, commonly called *the main*. The closest spot where another base can be built is called the *natural* (short for natural expansion) and the act of building a new base is called *expanding*. Since the resources are limited and the gathering rate important for the outcome of a game expanding to ones natural is usually something that a player wants to do at some point during the game to increase his income.

Figure 1.2: Screenshot taken from the main base of a Terran early in the game. The yellow text and figures annotate different elements in the game as well as the game's user interface.

1.2.4 Technology

When constructing buildings the general purpose is to use them to produce a selection of offensive units associated with the building. Buildings might also offer purchasable upgrades for your units, such as increased damage. The purpose of a building can also be to unlock higher technology. Technology, also called "tech", is the term for how advanced your available buildings, units and upgrades are, thereby unlocking higher technology means getting access to more buildings, units and upgrades.

As most units and buildings are not available in the beginning of the game, the step of unlocking technology is usually repeated throughout the game to help you fight your enemy. Little to nothing in Starcraft is free of charge, so producing units and buying upgrades always costs resources.

The order in which buildings are built is an integral part of the game since it is the recipe of how and when a player may get access to certain technology and units. The sequence of which buildings unlocks what is called the *tech tree*, an example of a hierarchical visualization is shown in Figure 1.3. A

player will typically decide a certain path in the tech tree to follow, since trying to unlock everything is too expensive and too time-consuming. The activity of unlocking technology in the tech tree is called *teching*.

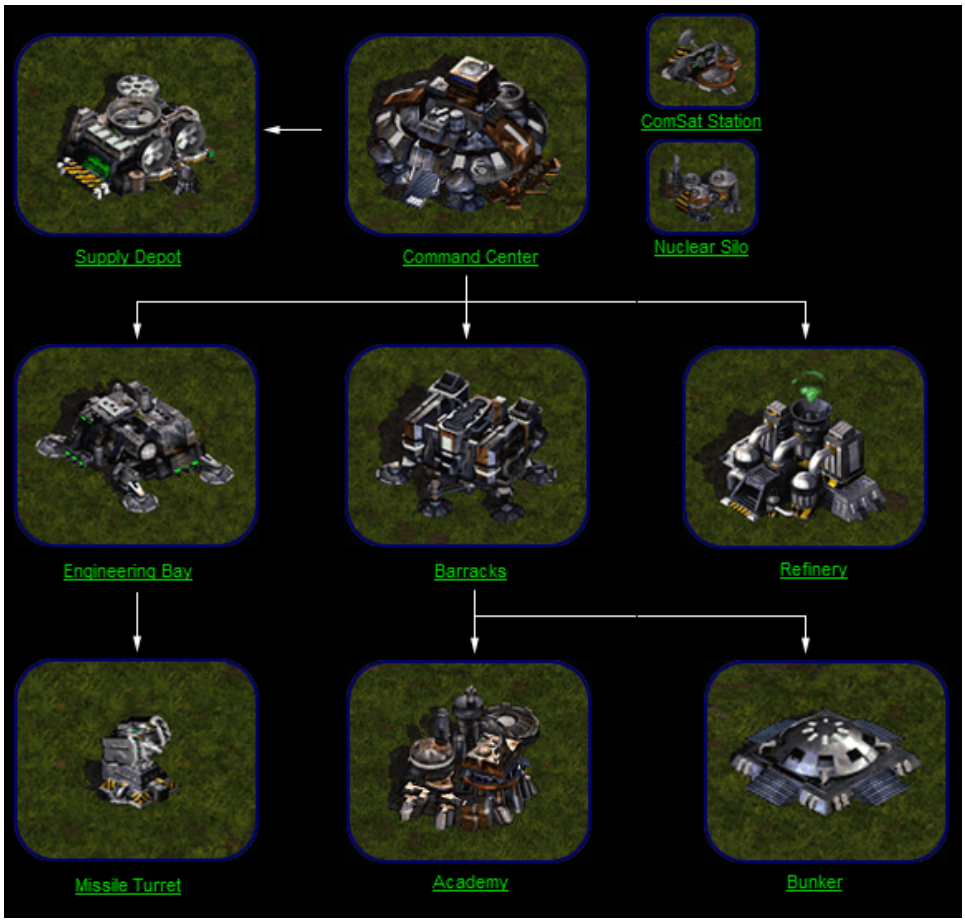

Figure 1.3: The first half of the Terran tech tree.

1.2.5 Units

The strength of a StarCraft unit is determined by its HP, armor, attack damage, cooldown, attack range and the special attacks it can do. HP and armor determines the survivability of the unit. The more armor a unit has the less HP are lost from the same attack. The combination of the attack damage and cooldown determines how much damage a unit can do per time. Attack range plays a factor in that the longer a unit's attack range the further away the unit can stand and still attack an enemy unit. If two units of opposing teams have different attack ranges the unit with the

longest attack range will be able to attack several times before the other unit is able to reach him. Special abilities can both boost survivability of one or several units and do damage to one or several enemy units. Special attacks will not be used in the implementation because of its complexity, and thus not used in the strength definition.

A unit can be defined as *strong* relative to another unit if it has better survivability and can do more damage over time. The other unit would then be defined as *weak*. This definition is made for the sake of comparing units in this report. The strength is not defined explicitly within the game, but it is reflected through the cost of the unit. However, the usefulness of a unit is determined by the situation, like what units it is facing. Since some units are strong against others having the right composition given your opponents composition might decide who wins an engagement.

If a unit kills one or more units that in total cost more than its own cost, then this unit is *cost effective*. This is achieved by good unit composition that counters the enemy units and good Micromanagement (see Section 3.2). Some units can do attacks that affect an area on the map, called *area of effect* (AoE) attacks. These attacks can do tremendous damage if fighting against a group clustered units. This is a good example of cost efficient play. The ability to do strong attacks usually comes at a price; e.g. the unit *high templar* has a very strong AoE attack, but also have very few hit points, making it very easy to kill if not positioned well (see Section 3.2)).

In order to own units a player needs something called *supply*. Supply is the amount of units you can own at any given time, and it maxes out at 200. Some units take up one supply while others can take up six. More supply is gained from building certain buildings or units. There is one building or unit for each race. If you do not have enough supply to create more units you are *supply blocked*. Putting your opponent in this state by destroying supply buildings or units is a good tactic.

1.2.6 Fog of War and Scouting

Fog of War (FOW) is a game mechanic normal in RTS games. FOW is present wherever you don't have any units or buildings placed, and you can not see enemy units or buildings in those areas. Special for StarCraft is that FOW is also present if enemy units are on higher ground than your units, thus forcing you to move your own units to higher ground, as you cannot attack anything you cannot see. It is the reason one has to do Scouting.

Scouting is the term used for finding out what your opponent is doing. Notice how black the map shown in Figure 1.2 is, this is because the player has not scouted anything and has only has map vision of his own base.

Scouting is done in several ways, depending on what race you play and how long the game has lasted. An example of scouting is moving a worker to your opponent's base to see what he is building and what number of units he has trained. As the game progresses you will gain access to units which are better suited for Scouting, e.g. flying and fast moving units. Some units are also invisible, making them good scouts. Terran have a special ability that lets them see a small area anywhere on the map for a short period of time. Using this ability in the opponents base is a good idea.

By scouting your opponent you can observe what units he plans to train and react by making units that counter them. It is also useful to decide where it is best to attack and get alerted about impending attacks. As the match progresses it is normal to try to monitor your opponents' movements constantly by placing scouts across the map.

1.3 Research Questions

Can RTS Micromanagement be perceived as a Multi-Objective Problem? Will an implementation using MOO be an improvement of one using a single-objective GA, and finally - will this be an improvement on previous work done with Potential Fields in RTS games?

Chapter 2

Theory and Methodology

In this chapter the theoretical fundamentals and concepts the solution is based on will be explained. The focus is on Potential Fields, Evolutionary Algorithms, Multi-Agent Systems and Multi-Objective Optimization.

2.1 Potential Fields

Potential Fields, also called Artificial Potential Fields (APF), is a method originally used for maneuvering robots between obstacles (Khatib, 1986). In a topological space it creates attracting and repelling fields, typically centred around a point. They can be thought off as magnetic charges working on a charged particle, each field attracting or repelling. The sum of all the fields, given the particles position, determines which direction it moves. Figure 2.1 shows a repelling Potential Field, where the repelling force is stronger closer to the center of the field.

Potential Fields have mostly been used to control robots because of their topological nature. However, they have also been used in other domains where the problem can be solved by simulating topology. Potential Fields have also been used for the deployment of mobile networks (Zavlanos and Pappas, 2007) and RTS games (Hagelbäck and Johansson, 2008).

The force (attractive or repulsive) from a Potential Field decreases the further away from the center you get. It can decrease in different ways: linearly; exponentially or discretely. The different variations represent different wanted behaviours. For example a discretely decreasing field could be used to keep the particle outside a strict boundary, useful for avoiding obstacles because you do not want to be affected by the obstacle unless you are close enough to crash. A linearly decreasing field on the other hand is better used

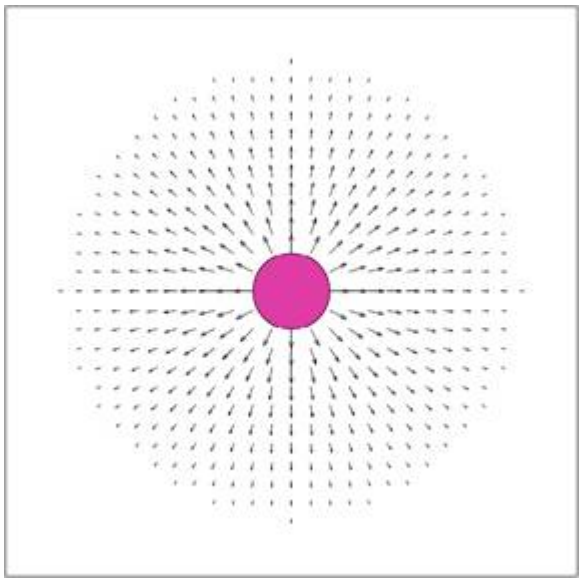

Figure 2.1: A repelling Potential Field (Safadi, 2007).

when several units are to be considered at once. When deciding where to move you will want to consider all enemy and friendly units and give more weight to the units that are closer to you. This is useful because enemy units close to you are more likely to be able to attack you, and friendly units close to you more likely to be able to protect you. However enemy units that are further away might be able to attack you if they move closer and thus have to be taken into consideration as well.

When representing Potential Fields the strength value from each Potential Field can be assigned to each pixel. An example of how this could look is shown in Figure 2.2, where both a negative and a positive linear Potential Fields are shown. A unit on a pixel position will move towards the pixel with the highest value surrounding it. Of course, one can increase the granularity and let several pixels represent one cell in a grid. This is actually a good way to lessen the computational power needed to do computations with them (Hagelbäck and Johansson, 2008). One does not have to represent such a grid explicitly. One can use the distance between the agent and the fields to create a result vector. If multiple units are using the same fields, e.g. if the fields are set by a centralized intelligence, representing them in a grid makes sense. The grid would be calculated beforehand and each unit would only need to check the values of the spaces closest to it. If each unit has their own Potential Fields calculating them this way would be impractical, as it would mean each unit would have to represent the whole grid and then decide what to do. Using the relative distance would be the best choice.

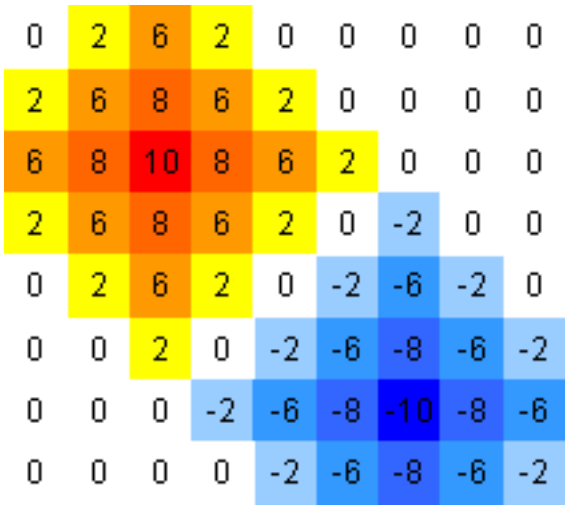

Figure 2.2: Two potential fields with strength values shown.

Potential Fields have been applied to RTS games as shown by Hagelbäck and Johansson (2008); Sandberg (2011). There are several advantages of using Potential Fields, they handle dynamic domains well and easily produce the behaviour wanted in these games (Hagelbäck and Johansson, 2008). Path-finding in a dynamic environment can be very hard. The most common path finding solution, the A* algorithm, struggles with this. Because of the nature of Potential Fields dynamic environments does not give a Potential Fields algorithm any extra work (Sandberg, 2011). Micromanagement (see Section 3.2) is a good example of wanted RTS behaviour that's possible with Potential Fields. Moving units away from enemy units, which is sometimes needed, can be done by placing negative fields on these units. Similarly making your units attack a weak target the Potential Fields can be designed to be strong on weak units.

2.2 Evolutionary Algorithms

Evolutionary Computation (EC) is a general term used for a group of stochastic search techniques who are loosely based on the Darwinian principle *Survival of the Fittest*, which they utilize by emulating evolution to optimize and improve their solutions. EC groups several techniques, the main types being Genetic Algorithms (GA), Evolutionary Strategies and Genetic Programming. All of these are also classified by the term Evolutionary Algorithms (EA), which have mechanisms in common such as reproduction, mutation, recombination and selection. An EA has a population of encoded solutions which can be manipulated by the mechanisms mentioned before,

and evaluated by a fitness function (Floreano and Mattiussi, 2008). EA requires both a fitness function as well as an *objective*, which represents a more high-level goal than the fitness function. This will be presented in Section 2.3.

An encoded individual in an EA population is called a *genome*, and is often represented as a string of bits, the encoding plans or blueprints for these individuals are called *genotypes*. The genotype is composed of several chromosomes, each of which represents a parameter for the solution. The genotype is turned into a phenotype by taking the chromosomes and decoding them into real values that can together form a testable solution candidate.

The cycle of life in an EA is depicted by Figure 2.3. The cycle on the figure starts in the bottom left corner as a set of genotypes, before the first step of evolution the first generation is often randomly generated. Once translated into phenotypes, the candidates have their fitness evaluated by testing it within the specific problem domain. The fitness of each individual determines their chance of staying in the population, and the chance of being selected as parents to breed new individuals into the next generation. Once the breeding is completed, the new genotypes are brought into the population and the cycle continues onto a new generation of individuals.

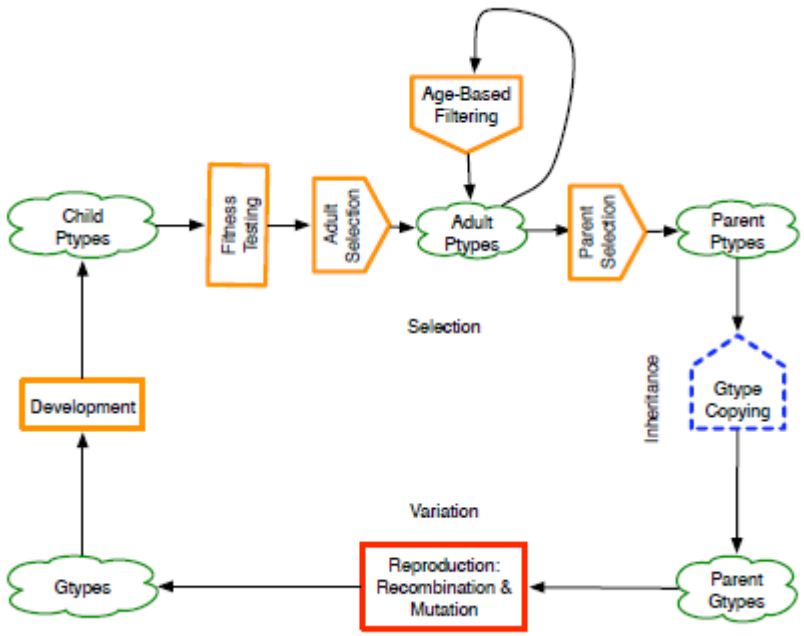

Figure 2.3: The basic EA cycle (Downing, 2010).

Evolutionary Operators (EVOPs) are techniques that are aimed at generating populations with high fitness, the three major EVOPs for EA are selection, recombination and mutation (Coello et al., 2007).

Selection decides how individuals of a population are drafted for parenthood depending on their fitness. There are several techniques available for EA implementation, one of which is tournament selection. In tournament selection the individuals are drafted into smaller groups, or tournaments, to compete against each other for parenthood using their fitness values. The size of these tournaments are chosen manually and determines the selection pressure of the EA. Selection pressure is how difficult it is to be drafted for parenthood. The higher the selection pressure the more the individuals with high fitness are favoured, as a result the convergence rate in EA is largely determined by the selection pressure. The tournaments are held until the number of parents wanted has been reached (Floreano and Mattiussi, 2008).

Elitism is the act of preserving an entity to the next generation without changing it. You do this if you want to preserve the best individuals in a population without risking that they are drastically changed during mutation or recombination. Elitism is an important part of Multi-Objective Optimization (see Section 2.3).

The Recombination EVOP focuses on how to best combine the different chromosomes of each parent to produce a new genotype that ideally takes the best properties from each partner genotype. Recombination in evolutionary algorithms is crossover, which cuts the parent genotypes at one or more given points and recombines the parts into a new child. In Figure 2.4 single-point crossover is shown. The two parts of the bars parents and children represent the chromosomes. The crossover operation switches part of each chromosome to create two new children with a part of each parent (Floreano and Mattiussi, 2008).

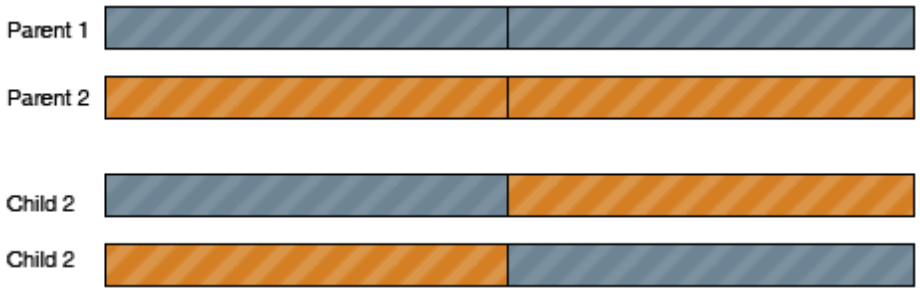

Figure 2.4: An example of a single point crossover done by cutting genotype segments and recombining pieces of each parent.

The mutation EVOP is the process of randomly changing the genome. Mutation can change one or several parts of the genome in a random fashion. How it is changed depends on the representation of the geneotype. In Figure 2.5 we show an example of mutation where the genotype is a series of bits and it is mutated by inverting a single bit on a random place in the bitstring. Changing several parts of the genome at once would make it change drastically. Mutating too rarely, on the other hand, could make the EA converge towards a local optima.

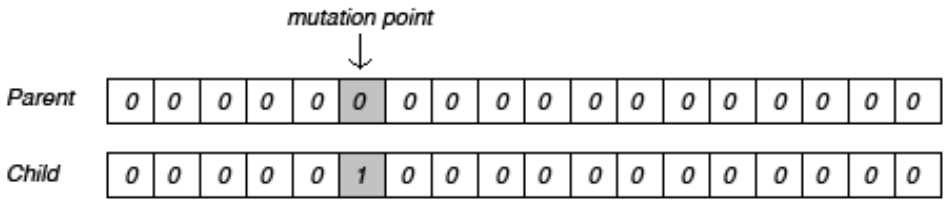

Figure 2.5: Single bit mutation

Choosing the right combination of EVOPs and their parameters is crucial for the performance of an EA. There is no domain-independent answer that is more correct. It depends on the problem at hand and how an EA represents that problem, and there is room for creativity and intuition when tuning the parameters. The parameters of an EA is mutation chance, crossover chance, selection mechanism, population size and the number of generations.

2.3 Multi Objective Optimization

Earlier we mentioned how an EA require both an objective and fitness function. The difference between these two can be vague at times. Objectives are high-level goals in the problem domain which describe what you want to accomplish. While fitness functions work in the algorithm domain to measure how well a particular solution manages to accomplish these goals, this is done by assigning a value to that solution that reflects the measured quality.

Definition 1. ***(Pareto Dominance (Coello et al., 2007))*** *A vector* $\boldsymbol{u} = (u_1, ..., u_k)$ *is said to* ***dominate*** *another vector* $\boldsymbol{v} = (v_1, ..., v_k)$ *(denoted by* $\boldsymbol{u} \preceq \boldsymbol{v}$*) if and only if* $\boldsymbol{u}$ *is partially less than* $\boldsymbol{v}$*, i.e.,* $\forall i \in \{1, ..., k\}, u_i \leq v_i \wedge \exists i \in \{1, ..., k\} : u_i < v_i$.

When a problem has multiple conflicting objectives, it becomes increasingly complex to represent the overall quality of a solution by a single fitness

function. We can define the problem of multi-objective optimization as the search for a vector of decision variables which are optimized to yield balanced and acceptable results for all the objectives. The clue to this is to find good compromises (or *trade-offs*) to satisfy all the objectives evenly, the goal being to find the *Pareto optimum*, which is defined as:

Definition 2. ***(Pareto Optimality (Coello et al., 2007))*** *A solution $x \in \Omega$ is said to be Pareto Optimal with respect to (w.r.t) Ω if and only if there is no $x' \in \Omega$ for which $\boldsymbol{v} = F(x') = (f_1(x'), ..., f_k(x'))$ dominates $\boldsymbol{u} = F(x) = (f_1(x), ..., f_k(x))$.*

Saying a vector v_1 dominates vector v_2 means that it performs better over all the objectives as Definition 1 states. This can be illustrated by looking at the solutions in *objective space* by plotting them in a coordinate space where each objective is a dimension as shown in Figure 2.6. The outer solutions in this space who do not have both their coordinates exceeded by another solution form collectively what is called the Pareto front.

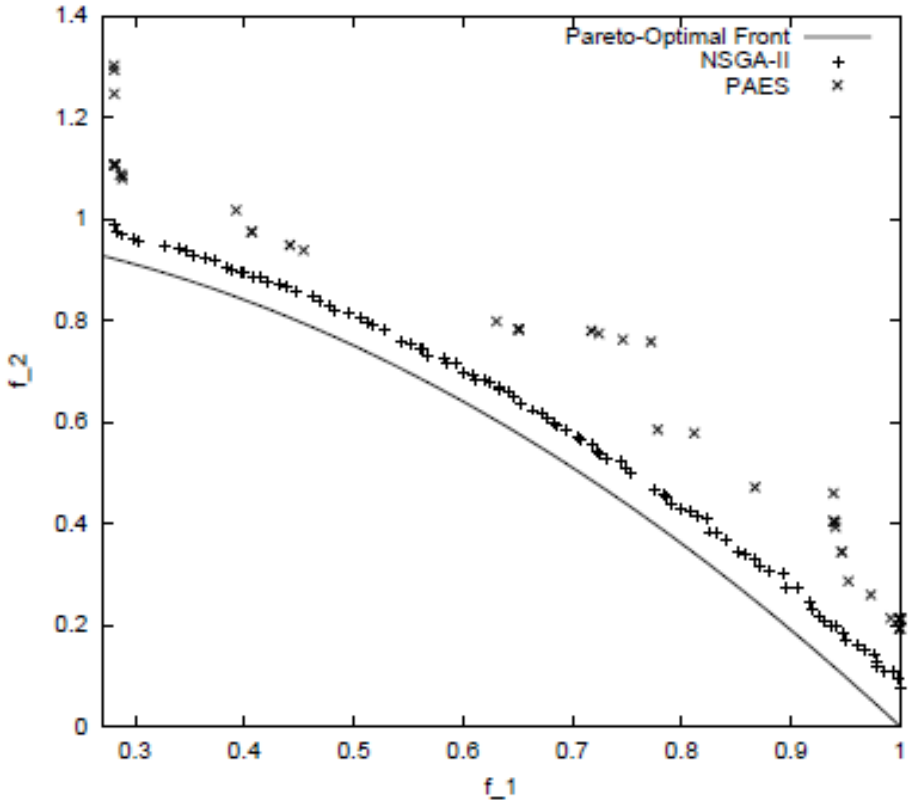

Figure 2.6: Non-dominated vectors in objective space, collectively called the Pareto front (Deb et al., 2000). This objective space has two objectives and thereby two dimensions.

Since the definition of a multi-objective problem suggests it is not possible to have a single, globally optimal solution, the ability of EA to produce a large variety of candidate solutions is very beneficial. This has motivated many Multi Objective Evolutionary Algorithms (MOEAs) to be suggested. The primary goal of these MOEAs is to utilize EAs ability to generate multiple Pareto-optimal candidates in a single run.

The problem of evaluating the performance of a Micromanagement AI can be viewed as a multi-objective problem, there are several factors to consider when evaluating the performance of the AI, and it will sometimes be difficult to compare two solutions as they might excel at different Micromanagement techniques (later discussed in Section 3.2). A multi-objective approach will make it possible to optimize the Micromanagement solution for several techniques at once, so the complex behaviour that is micromanagement can be approximated.

2.3.1 Nondominated Sorting Genetic Algorithm (NSGA-II)

One of the most adopted Multi Objective Genetic Algorithms (MOGA) is the improved non-dominated sorting genetic algorithm (NSGA-II) which is an elitist approach that does not require any additional user-defined parameters. NSGA-II combines the population of adults R_t and the population of children Q_t, and sorts them according to their non-domination rank as shown by the pseudo code in Figure 2.7. The rank is based on what pareto front the individual belongs to in accordance to who the individual dominates and who it might be dominated by.

NSGA-II attempts to promote diversity and good spread in the objective space by looping through the population and assigning a *Crowding Distance* metric to each solution as shown in Figure 2.9, this reflects how close it is to its neighbouring solutions and keeps the population diverse by making the algorithm more likely to explore solutions from lesser clustered objective space.

When creating the mating pool NSGA-II uses a selection operator $\geq_n$ which rewards objective fitness and spread by looking at the non-domination rank and crowding distance. The logic behind the operator is shown in Figure 2.8. It then continues by selecting the N best solutions for P_{t+1} according to the operator (N being the population size). Binary Tournament selection and recombination EVOPs are then applied to create a new offspring population Q_{t+1}.

```
fast-non-dominated-sort(P)
for each p ∈ P
  S_p = ∅
  n_p = 0
  for each q ∈ P
    if (p ≺ q) then                    If p dominates q
      S_p = S_p ∪ {q}                  Add q to the set of solutions dominated by p
    else if (q ≺ p) then
      n_p = n_p + 1                    Increment the domination counter of p
  if n_p = 0 then                      p belongs to the first front
    p_rank = 1
    F_1 = F_1 ∪ {p}
i = 1                                  Initialize the front counter
while F_i ≠ ∅
  Q = ∅                                Used to store the members of the next front
  for each p ∈ F_i
    for each q ∈ S_p
      n_q = n_q − 1
      if n_q = 0 then                  q belongs to the next front
        q_rank = i + 1
        Q = Q ∪ {q}
  i = i + 1
  F_i = Q
```

Figure 2.7: Pseudo code of the fast non-dominated sort algorithm used in NSGA-II (Deb et al., 2000)

The main loop of NSGA-II as shown in Figure 2.10 is repeated until a user defined condition is met, or manually terminated.

1. Non-domination rank (i_{rank})
2. Local crowding distance ($i_{distance}$)

We now define a partial order $\succeq_n$ as :
$i \succeq_n j$ if ($i_{rank} < j_{rank}$) or (($i_{rank} = j_{rank}$) and ($i_{distance} > j_{distance}$))

Figure 2.8: The selection operator as presented by (Deb et al., 2000).

```
crowding-distance-assignment(I)
l = |I|                                   number of solutions in I
for each i, set I[i]_distance = 0         initialize distance
for each objective m
  I = sort(I, m)                          sort using each objective value
  I[1]_distance = I[l]_distance = ∞       so that boundary points are always selected
  for i = 2 to (l − 1)                    for all other points
    I[i]_distance = I[i]_distance + (I[i + 1].m − I[i − 1].m)/(f_m^max − f_m^min)
```

Figure 2.9: Pseudo code for the crowding distance assignment (Deb et al., 2000).

```
R_t = P_t ∪ Q_t                           combine parent and children population
F = fast-nondominated-sort(R_t)           F = (F_1, F_2, ...), all non-dominated
                                              fronts of R_t
until |P_{t+1}| < N                       till the parent population is filled
  crowding-distance-assignment(F_i)       calculate crowding distance in F_i
  P_{t+1} = P_{t+1} ∪ F_i                 include i-th non-dominated front in the parent pop
Sort(P_{t+1}, ≥_n)                        sort in descending order using ≥_n
P_{t+1} = P_{t+1}[0 : N]                  choose the first N elements of P_{t+1}
Q_{t+1} = make-new-pop(P_{t+1})           use selection,crossover and mutation to create
t = t + 1                                     a new population Q_{t+1}
```

Figure 2.10: Pseudo code for the NSGA-II main loop (Deb et al., 2000).

Chapter 3

Game Domain and Mechanics

StarCraft is a complex game and it is best described in different parts. The game can be divided into Macromanagement and Micromanagement. This chapter will detail specifically Micromanagement mechanics, what available information is important to consider and how a computer can potentially outplay a human at Micromanagement.

3.1 Macromanagement

Macromanagement is making high level decisions during a match. These decisions entails as to what buildings to build, when to attack, when to scout, and in general strategical and tactical decisions. A big part of these decisions is what to use your accumulated resources on, and when one should try to increase the income by expanding to a new base. An example of when it is wise to expand is when attacking the enemy, so that the enemy does not have the opportunity to attack your new base while it is undefended.

While having a thought out plan about what to build at the start of the match is good, being able to adapt to new information during a match is even more important. If you are able to spot your opponent's buildings or units you can figure out what his plans are, and adapt to counter him. This is some of the reason why the game is so complex, your opponent can do something that you have never seen before, or pretending to do something else than what he is really doing. This high level of reasoning is called the *meta-game*.

In this project we are collaborating with another group, and they will be

handling the Macromanagement part of the game. This gives the opportunity to focus exclusively on Micromanagement, which is described in the following subsection.

3.2 Micromanagement

Micromanagement is the activity of controlling units during a battle. Doing this well requires a lot of skill and concentration. While it is possible to play StarCraft without prioritizing Micromanagement, doing so helps immensely because it will allow the player to be cost effective with his units (Sandberg, 2011). Positioning one unit in the wrong place can decide whether you loose or win a combat engagement, which again can decide if you win or loose the match. The game has a simple path algorithm which positions your units for you when you are moving in to attack an enemy, but the positioning is rarely optimal because the algorithm only considers finding the shortest path from A to B.

The most important Micromanagement techniques are; positioning, focusing fire, withdrawing units temporarily, and keeping units alive. Being able to execute these three puts you at a great advantage over players who can't, as you can be more cost-effective.

3.2.1 Positioning

Positioning involves where different types of units stand in relation to each other, the enemy and the environment. Examples of positioning are that weaker units should stand behind stronger units for protection, and units that want to be healed needs to stand within attack range of healing units. In the latter case the attack is a healing attack that replenishes HP, not one that does damage. Units should stand in such a way that they can attack the enemy while still having the ability to retreat. Standing on higher ground makes your units' attacks stronger.

Where your units should stand varies on the situation they are in. At some times your units should be spread to avoid area of effect attacks, at other times they should be grouped together so not to be picked off by enemy units. The numerous behaviours needed as well as the complexity of each behaviour makes it a difficult problem implementation-wise. Both the positions and the types of enemy units needs to be considered. It is also more general than the other techniques mentioned, and thus a bigger task to take on.

3.2.2 Focus Fire

Having your units attack the same enemy unit, is called focus fire, and will let you take down the enemy units more effectively. Which types of units you focus on first is important as some units can be killed fast but have very strong attacks. Taking down units that are close to dying is also important as they can keep attacking at full strength until they are dead.

The challenge here is to make sure focus firing is happening without putting your units at a greater than necessary risk. Which means that you should attack weak units but not if it means running through a a group of enemy units to do so. Of course, the pros and cons of doing any move in StarCraft are numerous.

3.2.3 Retreating

Each unit has a cooldown between attacks, which is the time it takes to reload the weapon. When a unit is on cooldown it is unable to attack until the cooldown wears off. Moving out of range of enemy units when on cooldown lets the unit avoid damage like Figure 3.1 shows. If this is exploited and your enemy fails to exploit it one has a great advantage. These techniques are very useful, but also hard to do for a human player because you usually control numerous units at a time. For example the task of selecting a single unit from a group of 50 and Micromanaging it while having to Macromanage is a difficult task. Especially since you would want to retreat several units every second for the whole duration of the battle. A computer is able to do these tasks effortlessly and with millisecond precision, this is further explained in Subsection 3.3.1.

When using a computer each of these tasks can have their own processes (see Section 5.3) and acting with millisecond precision is possible.

The challenge here is to do the move effectively. The unit should move out of range for just long enough, so that it can attack at once when the cooldown wears off. If enemy units follow it this becomes increasingly difficult, especially if the result is that the unit has to run far before the cooldown wears of. If this leads the unit to be backed up against the wall it is at a disadvantage larger than that of being on cooldown.

Micromanagement is a problem where each instance in time have almost an endless search space of possible moves and outcomes, and shows really how complex RTS games are. Brute forcing this problem is out of the question, as even determining if an action at an instance is optimal is impossible to determine. The best we could hope for would be a solution where good

Figure 3.1: These two screenshots shows a Dragoon attacking and moving away from a Zealot. Dragoon is able to exploit its superior attack range by moving away from the Zealot while the Dragoon is on cooldown.

guesses can be made in any situation. In other words a highly dynamic solution.

3.2.4 Staying alive

Keeping units alive is very important as a damaged unit can still do damage to enemy units, but a dead one cannot. Because of this one often wants to pull damaged units away if they're being attacked. If not explicitly controlled a unit will attack the closest unit of the opposing team within attack range. This means that if a damaged unit is pulled away, the enemy unit(s) attacking it might shift focus to another one of your unit. Hopefully one with more HP.

Other ways to make sure your units stay alive is attacking with undamaged units first. To do this technique an AI one would have to make sure a unit changes its behaviour when it is damaged in such a way that it would be more careful than the undamaged units. Thus staying alive for longer than it would otherwise.

3.3 Attributes

StarCraft is a highly complex domain and it is important to objectify it to understand how the methods presented in Chapter 2 can be used to effectively solve the problem of Micromanagement. Relevant information

that can be abstracted and used in Micromanagement will be discussed here.

A map in StarCraft should be perceived as a grid. The positions on this grid are represented by two dimensional coordinates x and y, where there is one coordinate for each pixel on the grid. The map can vary in size and usually contains various forms of terrain: flat terrain; low- and high ground; ramps that provide a way to move from low- to high ground and impassable obstacles. All information about the size and shape of the terrain is final and is known to every player. It is important to note that vision from low- to high ground is greatly reduced, giving advantage to anyone fighting on the high ground onto the low ground. Chokepoints are narrow areas, and a group of units within the chokepoint will be at a disadvantage against a group of enemy units of the outside. This is because the units in the chokepoint will not be able to spread out so all units can attack simultaneously. Such factors are important to analyse when entering a combat, and it is easy to located the coordinates of these choke point by using terrain analysing features in the API.

To objectify the domain we will determine the properties and variables of it. Properties are constant while variables can vary for each timestep. First the objectification of the map will be shown to give an understanding on how the basic information flow works. Like where a unit can walk and how this data is presented in BWAPI which will be presented in Chapter 5. The most important thing to take away from these properties and variables are what a position on the map means. afterwards the properties and variables directly related to Micromanagement will be presented, as they are important for understanding the solution.

- Obstacles *(property)*
- Ramps *(property)*
- Chokepoints *(property)*
- Low Ground *(property)*
- High Ground *(property)*
- Mineral Deposits *(property)*
- Vespene Geysers *(property)*
- Units *(variable)*
- Buildings *(variable)*

A property is the property of the terrain at the position in question. All positions are either defined as high- or low ground, but can only have one more property. Either that it is an obstacle, a ramp, that it is in a chokepoint

or that it is clear terrain. Mineral Deposits and Vespene Geysers area have a constant position but they gradually decrease in value (amount of remaining resources) as they are gathered from. Units and buildings are variables in the way that they are not constant in any position or in the game. All units, and some buildings, can move around and all units and buildings can be destroyed. Only one unit or building can be in the same position at once, with the exception of workers when gathering minerals.

A player will always have complete information about his own units. Information about enemy units are available as long as they are not covered by the fog of war. This includes not only their coordinates and velocity, but also a series of variables and properties that can be exploited and used in calculations to determine Micromanagement decisions or analysis. The properties and variables of special interests are:

- Armor *(property)*
- Cooldown *(property)*
- Attack damage *(property)*
- Attack range *(property)*
- Movement speed *(property)*
- HP *(variable)*
- Remaining_cooldown *(variable)*
- Position_coordinate *(variable)*

Properties of units is information that do not change, with a few exceptions that are not relevant to the solution. While variables are observed data that is unique and can change every timestep. The properties show the relative strength of the unit as discussed in Subsection 1.2.5, and the aggregated strength of enemy and friendly units is important for deciding if to attack or run away from a combat engagement. The variables are real-time information and useful for Micromanagement decisions, especially the position coordinate because it can be used to calculate the distance to the unit. When combined with attack range the distance gives information about if an enemy unit can attack the unit, if the unit can attack the enemy unit and how far it has to move for any of these to happen.

3.3.1 Actions Per Minute

Actions per minute (APM) is a common term in the RTS genre. It is a measurement for the speed or frequency at which a player is able to interact with the game. It does not directly correspond to the skill of a player, but

it can show how aware a player is, the intensity of a situation and how fast the player is able to carry out the actions that needs to be done. Typical APM ranges from 20 for beginners to about 300 for professional gamers [1]. This is a game mechanic where an AI can easily reach inhuman capabilities, some StarCraft AIs have a recorded APM of several thousand [2].

3.3.2 Functions

Section 3.3 has so far covered the detailed, low-level data that can be abstracted from the game. This subsection will elaborate on how this information can be used in computations to make decisions in Micromanagement.

As implied in Subsection 3.3.1 there is great potential for AI in RTS when it comes to multi-tasking and speed. A human player will consider and use most of the information available to a degree when doing Micromanagement, but it is impossible for him to observe all the information at the same time. An AI will on the other hand be able to measure everything precisely and control all the units with the same precision. This is the main advantage an AI has over a human player. It lacks the complex reasoning ability a human has, but can make up for it with superior oversight and unit control.

For the AI to be able to make Micromanagement decisions in StarCraft it needs to use the properties and variables related to Micromanagement. There are several key equations that will enable the AI to execute the Micromanagement techniques. One such equation uses the relationship between attack ranges and distance. To explain this MSD will be defined as the unit's maximum attack range, $eMSD$ as the enemy unit's maximum shooting distance and *distance* as the *distance* between then two units.

If $MSD - eMSD > 0$ the unit will be able to attack the enemy unit without being hit itself, as long as it pulls back after it has attacked. It can attack again when the remaining cooldown is 0. This is the retreating technique (see Subsection 3.2.3). On the other hand if $MSD - eMSD < 0$ the unit will be in the opposite position and risk getting hit far more times than the opponent. If $MSD << eMSD$ the retreating technique cannot be executed and attacking without retreat is the best tactic, because the unit will not be able to take advantage of the MSD difference. Using the *distance* variable, found by calculating the vector between two coordinates the following equation can be created:

$$distance - eMSD > 0 \tag{3.1}$$

[1] http://wiki.teamliquid.net/starcraft/Actions_per_Minute

[2] Recorded footage of AIIDE StarCraft AI Competition contestants are freely available on YouTube

This function can be used as a boolean expression, outputting True if the unit is outside the enemy's attack range and False if it is inside it.

The *remaining_cooldown* variable is important in relation to what actions the unit should take. If on cooldown ($remaining\_cooldown > 0$) the unit should pull away from the enemies' attack ranges to avoid getting hit unnecessarily. When on cooldown the unit should attack an enemy unit as soon as possible, since not doing this right away reduces the total damage output of the unit. This implies that if a unit is to pull away from enemy attack ranges it still has to be close enough so it can attack once the cooldown is finished. The following function can be used to distinguish between being on cooldown and not being on cooldown:

$$remaining\_cooldown > 0 \tag{3.2}$$

The HP variable can be used to determine which enemy to take down in an engagement, as well as determine if the unit needs to be more careful (if it's HP is low). The less HP an enemy unit has the fewer attacks are needed to destroy it and the sooner it dies the sooner it is no longer able to attack. If several friendly units were to focus on the same enemy it would also be destroyed quicker. Looking at the HP of a friendly unit on the other hand can be used to determine if the unit should put more focus on staying alive, as discussed in the previous section. To create functions to represent the HP of friendly and enemy units the following variables are defined: HP as the unit's HP, eHP as the enemy's HP and $max(X)$ as the maximum value of variable X.

$$1 - \frac{eHP}{max(eHP)} \tag{3.3}$$

$$1 - \frac{HP}{max(HP)} \tag{3.4}$$

By $\frac{HP}{max(HP)}$ you get the percentage of the remaining HP of the unit. So each of the equations' output increases as the HP of the unit decreases. This is useful to emphasize low HP.

Equation 3.3 returns a high value when the enemy unit's HP is low. This can be used to target enemy units with low HP, as described in Subsection 3.2.2.

Equation 3.4 returns a high value when the unit's HP is low. When a friendly unit's HP is low measures should be taken to make it stay alive, as discussed in subsection 3.2.4.

This chapter has covered the more advanced aspects of StarCraft, the next chapter will cover related research in this domain and how previous StarCraft AI's has used some of some of the information presented here.

Chapter 4

Related Work

In this chapter we will mention research related to our work and further on detail some StarCraft bots that have had significant importance in our own research and implementation.

The idea of evolutionary computation (EC) was made widely known in the 1960s by Rechenberg (1965); Fogel et al. (1966); Holland (1975), and was aimed at tackling problems in software and hardware systems that were too difficult for the current available analytical methods, by exploiting the mechanics of natural evolution. Today it is applicable in many fields such as machine learning, system optimization, hardware design, computer-assisted manufacturing, material production technologies, and robotics (Floreano and Mattiussi, 2008, p. 1). Evolutionary algorithms, a subset of EC, has the potential to discover novel, innovative solutions that differ greatly from human-designed solutions. A good example of this statement is the NASA X-band antenna that was designed by the help of an evolutionary algorithm (Floreano and Mattiussi, 2008, p. 40).

Evolutionary Algorithms have been used for several types of games and while not in widespread use it has proven useful in certain domains. Among these are first-person shooter games (Priesterjahn et al., 2006) and real-time strategy games (Fernandez-Ares et al., 2011). It is the most effective when used to optimize decision mechanisms that are already in place, like state machines (Fernandez-Ares et al., 2011), behaviour trees (Lim et al., 2010) or Potential Fields (Hagelbäck and Johansson, 2008)

Multiple objectives in problems has been a troubling issue for optimizing solutions. Classical optimization methods have suggested converting the multi-objective optimization problem to a single-objective optimization problem by emphasizing one particular Pareto-optimal solution at a time (Deb et al., 2000). However, due to the EAs ability to search for multiple

solutions concurrently in a single run, several multi-objective optimization evolutionary algorithms (MOEAs) have been suggested over the years (Zitzler and Thiele, 1998). One of the most famous algorithms to date that has shown promising results (Deb et al., 2000) is the nondominated sorting genetic algorithm (NSGA-II), which we will be using as part of our implementation. NSGA-II is an improved version of NSGA suggested by (Srinivas and Deb, 1994), which reduces the complexity of the original algorithm from $O(mN^3)$ to $O(mN^2)$.

This work is largely based on Johan Hägelback's research on using multi-agent Potential Fields in real-time strategy games (Hagelbäck and Johansson, 2008) and adjusting these fields by the help of the optimization techniques of genetic algorithms (Sandberg, 2011). The idea of using artificial Potential Fields (APF) in multi-agent systems (MAS) is not a new one however, where the earliest ideas of this roots back to 1986 robotics (Khatib, 1986). Newer research shows more applicable areas for APF's, Howard et.al presented a potential-field based approach to deploying a mobile sensor network (Howard et al., 2002), and the concept was first brought to the game domain by Johannson et.al (Johansson and Saffiotti, 2001) who used Artificial Potential Fields (APF) in a bot for the Robot Soccer Cup in 2001. The same year (Vadakkepat et al., 2001) also applied Evolutionary Artificial Potential Field on the Robot Soccer platform, adjusting the weights of the APF with multi-objective evolution.

Multi-Agent Systems has been used in games such as the board game Diplomacy (Kraus and Lehmann, 1995), Johansson has also defined a general architecture of this kind aimed at board games (Johansson, 2006). Combining this self-organized type of system with APF was first presented by (Hagelbäck and Johansson, 2008), in which they showed a step-by-step methodology for designing Multi-Agent Potential Field (MAPF) solutions for RTS games. This work was later followed up by (Sandberg, 2011) who explored the possibility of adjusting the weights of MAPF's with evolutionary optimization (see Section 4.1 and Section 4.2). Sandberg also chose StarCraft: Broodwar as the testing platform to develop the AI for, arguing for the balanced gameplay and interesting high complexity.

Numerous bots have been created for StarCraft and similar RTS games. Few of them have been documented well, however some have thorough scientific articles describing them. These have given us a good deal of information and is the closest we get to knowing what is good solutions without testing it ourselves.

4.1 MAPF bot

In a series of articles Johan Hagelbäck (Hagelbäck and Johansson, 2008) uses Potential Fields in RTS games, among them StarCraft. The whole game is controlled by Potential Fields but a lot of the focus is directed towards Micromanagement. His articles are the inspiration for Sandberg (2011) which is further explored in 4.2.

Hägelback uses Multi-Agent Potential Fields (MAPF) for his solution. In MAPF each agent has their own Potential Fields where each agent is a unit in the game. Potential Fields are also assigned to objects and to abstract concepts like reacting to opponents movements (called *forces*). When designing the MAPF solution one goes through several steps to make sure the solution is good enough (Hagelbäck and Johansson, 2008).

1. Identify the objects of the game.
2. Identify the driving forces in the game.
3. Assign fields to objects.
4. Identify the granularity and space in the enviornment.
5. Identify the agents.
6. Design the supporting architecture.

In this solution the Potential Fields are calculated once every frame and they are shared among all the units in one group. Each unit then looks at the 8 tiles around him and the one he is standing on to determine in which direction to move. It moves in the direction where the value is the strongest. If the strongest pixel is the one the unit is on the unit will stay still.

The Potential Fields for StarCraft are not discussed in detail, so equations are unknown, however the Potential Fields for other games are discussed. The important thing to know is that there are few types of fields, four in the case of the game *Tank Battle* which also is a RTS game.

Hägelback adresses the challenge of Fog Of War (Hagelbäck and Johansson, 2008). He showed that with minor additions to the algorithm the performance could become as good as without FOW. The most important part was saving the condition of the units not currently visible because of FOW. Also having perfect information about the obstacles on the map is a necessity.

Hagelbäck and Johansson (2008) also addressed possible problems one can have when using Potential Fields. Like units getting stuck in local optima and performance issues. For getting stuck in a local optima he suggests adding random noise to the calculations so the output always is a little bit

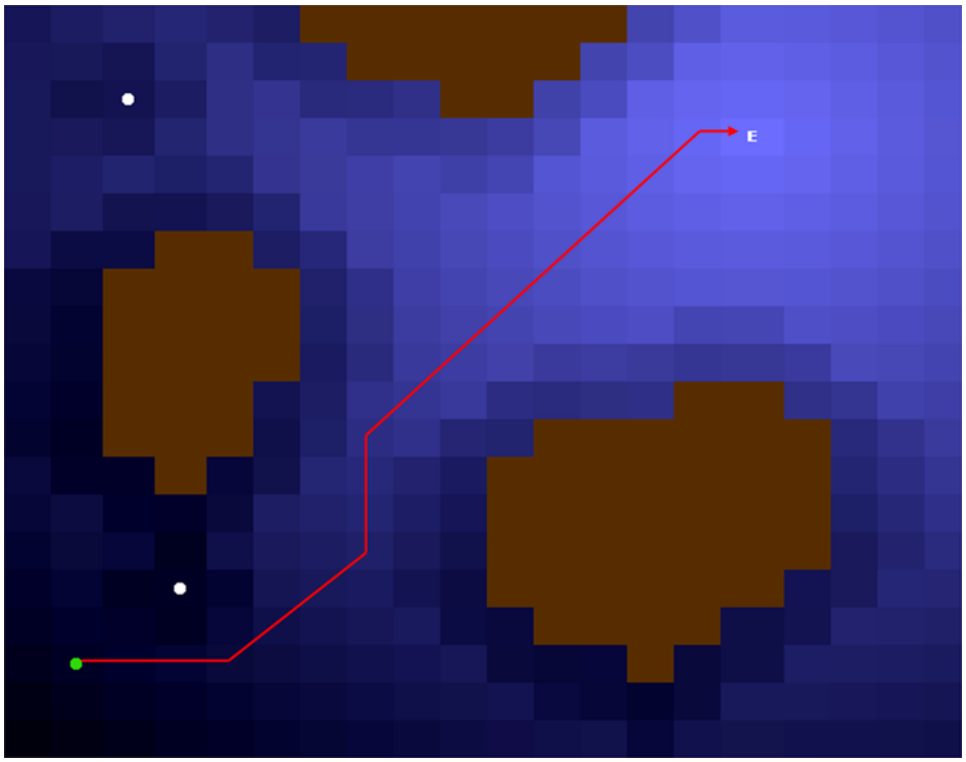

Figure 4.1: Two potential fields with strength values shown (Hagelbäck and Johansson, 2008). The lighter the blue color is the stronger the field is.

different. For performance issues he suggests increasing the granularity (see Section 2.1) and making sure the Potential Field equations are effective.

The weights for the attributes that decide how strong or weak a Potential Field is, are hard coded. However since he has few attributes he is doing OK. If one were to have more attributes, the task of finding the correct combination of weights would be very difficult. If one were to use more attributes another approach would be desirable.

The bot did OK against the best bots from the 2010 AIIDE competition, winning 33% of the matches. To improve the results Hagelbäck and Johansson (2008) suggests improving the Micromanagement and exploiting the special abilities of different units better.

4.2 EMAPF bot

Thomas Sandberg presents an AI that uses Potential Fields and Evolutionary Algorithms (Sandberg, 2011). His method is named Evolutionary

Multi-Agent Potential Fields (EMAPF) and is based on the MAPF method described in the previous section. The upgrade done by Sandberg is that he uses Evolutionary Algorithms to tune the weights of the Potential Fields functions instead of hard coding them like Hagelbäck does. He argues that tuning the Potential Fields is time-consuming and difficult, and using EA to tune them will be easier and allow for more attributes to be used.

The Potential Fields are calculated in the same was as in (Hagelbäck and Johansson, 2008), but the fields themselves are different. For one there are more types of Potential Fields, and the rationale behind this is that the EA optimization is able to handle more Potential Fields than hard-coding. The different fields are:

Maximum Shooting Distance An attractive field that attracts the unit to be within shooting distance of an enemy.

Weapon Cool Down A repulsive field on an enemy unit when the unit is on cooldown.

Centroid Of Squad An attractive field on the center of the squad of units that the unit is a part of.

Center Of The Map An attractive field at the center of the map to make units favor the center of the map over the edges.

Map Edge A repulsive field on the edge of the map so units won't walk into it.

Own Unit A repulsive field on friendly units to avoid collisions.

Enemy Unit A repulsive field on enemy units to avoid collisions.

Neutral Unit A repulsive field on neutral units to avoid collisions.

The same pipeline is used for designing the solution as it is in the MAPF solution, but tuning of the EA is added. The detailed description of how Sandberg (2011) designs his EA is useful. For example he has a large number of parameters, which is also his reason for optimizing with an EA.

He is also focused on Small Scale Combat (SSC), which essentially is the same as Micromanagement. In his experimental setup there are no buildings and thus no possibility to produce new units. Instead each player is given a set of units, and the player with units left wins. The main reason why his experimental setup is interesting is that he uses BWAPI, the same API as the one used for the experiments in this thesis. Which means that some of the challenges he faced and his solution to these challenges will be similar to those in this work.

During training he calculates the fitness after an engagement has ended. The fitness functions include the amount of units you have, how many units

you have killed and how many you have lost. Finally the remaining hit points and shields are added to the score. The experiments in this report will be done in a similar fashion, using a custom-made map and predefined units. This lets you control the enviornmental attributes you want the AI to take into account while training.

The EMAPF implementation produces good results, beating the built-in AI in StarCraft consistently, (Sandberg, 2011), which strengthens its relevance.

Chapter 5

Tools and Framework

In this chapter the API used to communicate with StarCraft will be explained, as well as a library that extends the API called BWSAL. The architecture for the AI in this work will also be presented, which contains a modular case-based reasoning (CBR) framework and that the Micromanagement AI plugs into.

5.1 BWAPI

Brood War Application Programmable Interface (BWAPI) [1] is the API that is going to be used for controlling and getting information from StarCraft: Brood War. This is the API that the StarCraft AI competitions are based on [2]. It was also the basis in the authors' Specialization Project where the research was centred around the use and possibilities of BWAPI. The API is written in and for C++, however there exists wrappers for several languages like Java and Python. Once compiled the DLL (Dynamic-link library) is injected by launching StarCraft using the third-party program Chaoslauncher [3].

BWAPI gives complete access to the game states and control over all units and buildings, as well as lookup information about different static game properties. To exemplify what unit information of high importance BWAPI provides; the current hit points (HP), position coordinates, cooldown (for the next attack), attack strength as well as a myriad of unit-type specific information. These are also examples of the type of information that will be

[1] http://code.google.com/p/bwapi

[2] AIIDE acquired a content-use license from Blizzard to host the competition in 08.2011

[3] http://wiki.teamliquid.net/starcraft/Chaoslauncher

exploited in the AI that this report presents, and will be further explored in Chapter 6.

5.2 BWSAL

The Brood War Standard Add-on Library (BWSAL) is a library for BWAPI that provides several useful features. Among these are automating building placement, scouting and additional unit information.

While the features provided by BWSAL is mainly of interest for AIs more versatile than one only focusing on Micromanagement, the *UnitGroup* and the *UnitGroupManager* classes will prove useful. They provides additional information about the position of the AI's units, as well as relevant topological positions like choke points.

There are no alternative language wrappers available for BWSAL, which means that any project using it needs to be written in C++.

5.3 Architecture

The architecture for the AI bot is based on the research done by the authors and several other students during a collaborative Specialization Project. The architecture behind the AI is structured as a module that can be plugged into a Case Based Reasoning (CBR) oriented agent architecture that is developed in parallel by a group of colleagues in connection with their thesis. The focus of the CBR agent is around Macromanagement, and the Micromanagement AI in this report will act as a plug-in that is activated as a reactive part of the system. The top-level architecture for this system is showed in Figure 5.1, and will be further explained in the following paragraphs.

The architecture is based on a layered architecture used in robotics. It has three layers, where the top one is the Planning layer, the second one is the Executive layer and the third is the Reactive layer. The Planning layer makes plans, and alters them as new information is gained.

The Executive layer will be responsible for resource management, deciding how to execute plans and in which order to do so. It passes the plans down to the reactive layer. The commands passed are composed of keywords like "Attack" and "Enemy base".

The Reactive Layer executes its orders by initializing new *Behaviours*. Behaviours are given specific unit groups or buildings that is theirs to supervise and control. A Micromanagement AI will act as a *Simple Behaviour* in this

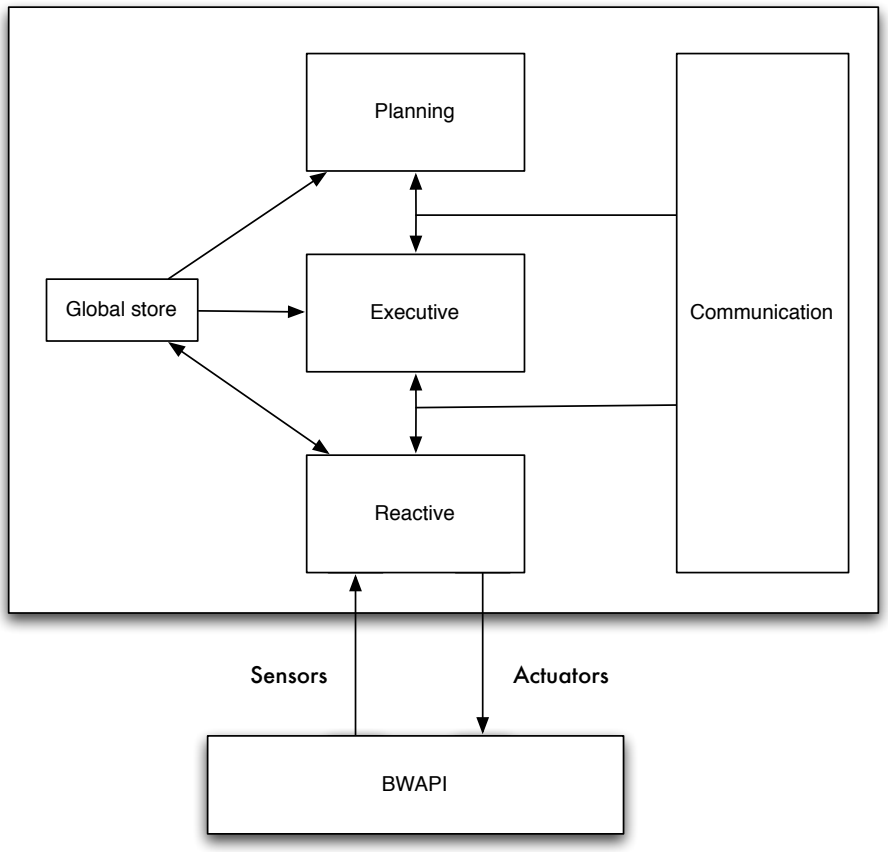

Figure 5.1: The three layered architecture.

system as shown in Figure 5.2, instantiation will come from the Reactive layer once it senses a combat situation. When instanced the Micromanagement Behaviour will be delegated control over the units that needs to be Micromanaged.

A behaviour is able to communicate its success with the Executive layer, and through that the Planning layer, with a simple communication protocol. A behaviour is usually initialized with a specific order like "Attack Enemy Base X". After this order is complete, or if the execution of this order fails, the behaviour reports back before terminating. In the case of the Micromanagement AI the order is to destroy the target within range of the group of units. If the enemy units are destroyed a "Success" is returned. If all the friendly units in the group dies then "Failed" is returned. Behaviours will live until either the behaviour itself says that it is done, or it is prematurely killed. Killing the Micromanagement behaviour may happen if the Executive layer finds that the unit group is of more use somewhere else or it predicts that the group will loose the combat engagement and should escape instead. The

Micromanagement AI makes no such predictions, and will thus continue its behaviour until either all the units in its possession are dead, or the enemy units are.

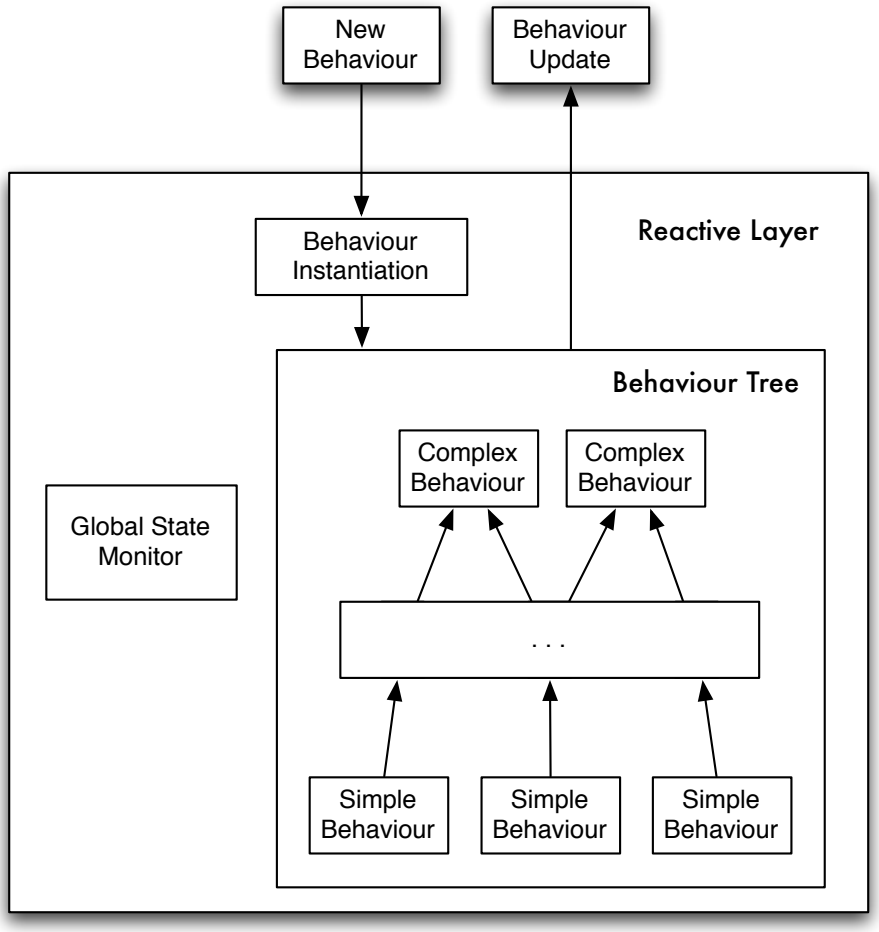

Figure 5.2: The Reactive layer.

In addition to the mentioned layers the architecture has the Global Store and the Communication Module. The Global Store works as a collective memory and is similar to a Black Board. It keeps updated information on the game state as well as variables used. An example is that if new information comes to light all parts of the architecture gets to know it at once and can change their behaviour accordingly. The Communication Module handles the communication between layers in a simplistic but consistent way and is what is used for the API as discussed in the previous paragraph. Having the Communication Module lets the different layers be independent of each other, as they do not need information about where the messages comes

from or goes to.

The Micromanagement AI as a Simple Behaviour in relation to the system architecture is depicted in Figure 5.3. The inner workings of the Micromanagement AI will be further detailed in the next chapter.

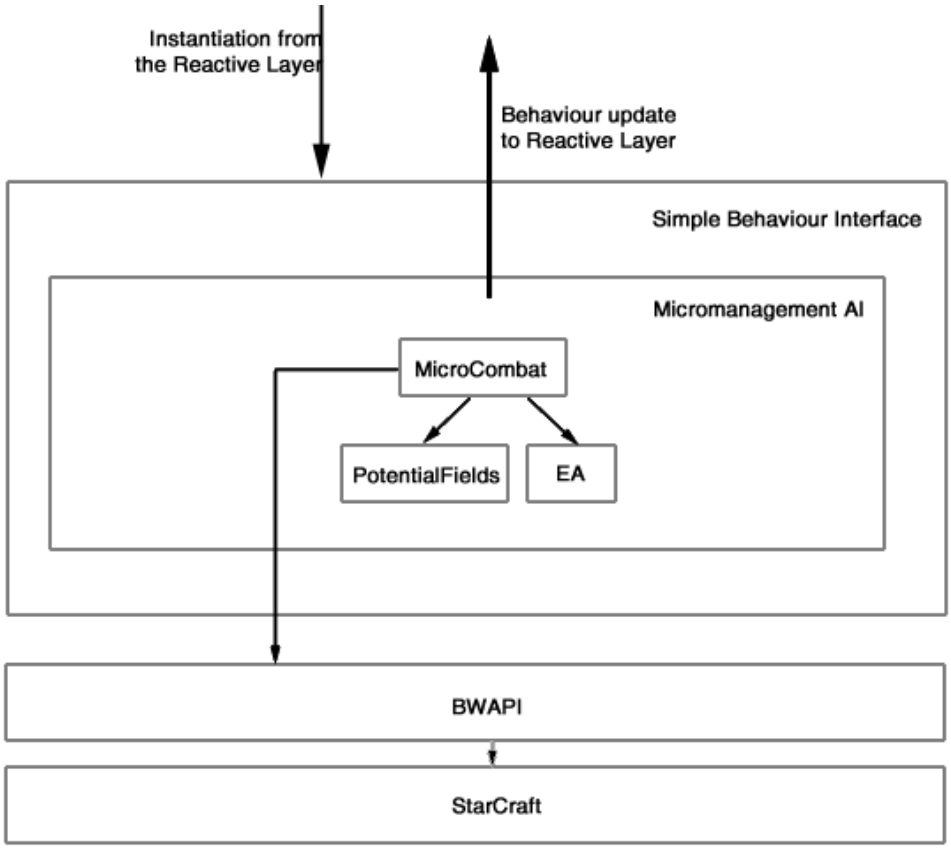

Figure 5.3: Architectural view of the Micromanagement AI implemented as a Simple Behaviour.

Chapter 6

Model

The solution to the Micromanagement problem will be Potential Fields that are tuned by a multi-objective genetic algorithm (MOGA). The Micromanagement techniques presented in Chapter 3 are the basis for the solution.

Since the solution is complex the explanation will be divided into parts addressed separately. The solution is an expansion of the ideas from Hagelbäck and Johansson (2008) and Sandberg (2011), using MOGA to optimize instead of GA.

6.1 AI Overview

The AI will be in one of two possible modes when playing. The *training mode* is for evolving solutions with a GA, and the *testing mode* is for running a match with a manually set solution. This section will focus mainly on the training mode and this is this mode that is used for finding the final solution.

At the beginning of a match the AI will have a genome that represents a set of weights used in the Potential Field functions, these weights will be used for the entirety of the match and the purpose of the match will be to test how well the AI performs with the given weights. This means that during training, as the AI will pick different weights, the AI's behaviour will only change between matches. How these weights and their Potential Fields decide the actions of the controlling units is explained in Section 6.2.

In training mode each StarCraft match is a fitness test. The data that is used to decide the fitness is gathered both during and at the end of the match, and then used in the fitness functions (see Section 6.2.1). This means that

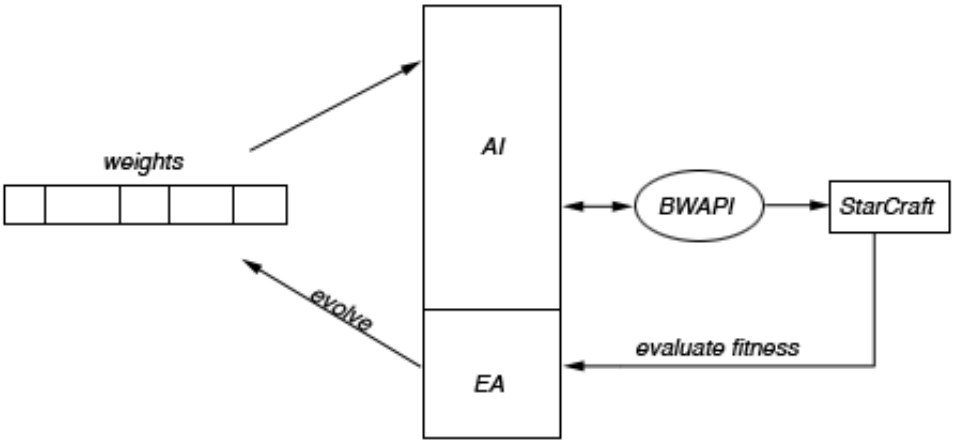

Figure 6.1: Overview of the AI in training mode.

during training the set of weights represented by the genome will evolve, and the average individuals behaviour change.

While in testing mode the AI performs quite similarly to the testing mode shown in Figure 6.1, except that it disconnects the evaluating and evolving part, and has a manually set weight array. This means that its behaviour will never change between matches. The testing mode will be used for the experiments.

6.2 Potential Fields

The algorithm will decide in which direction a unit will move or who to attack based on the Potential Field functions. They will output a value that represents a direction or a target, and is translated by the algorithm into a StarCraft command. The purpose of the Potential Fields is to execute the techniques presented in Section 3.2.

Every unit you control will have their own Potential Fields which means they have to be calculated for each unit. This is in contrast to Hagelbäck and Johansson (2008) who have the same Potential Fields for all groups of units. The fields will be placed on all enemy units in sight and at the center of the group (COG) of soldiers you control. The fields on the enemy units are for deciding who to attack and deciding in which direction to retreat. The fields placed at the center of the group are for directing the unit towards other friendly units when retreating. The Potential Fields will be updated several times each second because the environment in StarCraft is highly dynamic and fast paced.

6.2.1 Functions

The functions decide how strong a field is. There are three potential fields: One for enemy units when not on cooldown, one for enemy units when on cooldown, and one for the center of squad.

There are different fields depending on whether it is for an enemy unit, COG, or if the unit is on cooldown. The functions use the properties and variables presented in Section 3.3. Sandberg (2011) inspired the general style of the functions, especially the dependence on Maximum Shooting Distance (MSD), but they are quite different because the placement is different. Also there are fewer types of Potential Fields (only three), because this approach to the Micromanagement problem is different. There is more focus on Micromanagement techniques and less on general Micromanagement in the solution in this thesis.

The functions will use the following terminology:

$force$ The total attraction/repulsion exerted from one Potential Field.

$distance$ The distance from the enemy unit.

MSD The maximum shooting distance of the unit.

$eMSD$ The maximum shooting distance of the enemy unit.

HP The unit's HP.

eHP The enemy unit's HP.

HP_{max} The maximum amount of HP a unit can have.

w_n A weight.

$max(x, y)$ The greater of x and y.

$$force_{attack} = w_3 * (1 - \frac{eHP}{eHP_{max}}) + w_4 * \frac{1}{distance} \tag{6.1}$$

Equation 6.1 shows the function used when the unit is not on cooldown. In this mode the Potential Fields around enemy units will have an attracting force, and the unit with the highest attractive force is attacked. It is necessary that only the strongest field is the one that affects the movement of the unit to achieve focus firing. Because if a unit in StarCraft is commanded to attack in a direction it will automatically attack the closest one, but to achieve effective focus fire one will want to, at times, attack other units than the closest one.

$$force_{escape} = \begin{cases} w_0 * distance & if(distance - max(MSD, eMSD) + w_2 * (1 - \frac{HP}{HP_{max}}) > 0) \\ -\frac{w_1}{distance} & else \end{cases} \tag{6.2}$$

Equation 6.2 shows the function for the field emitted by an enemy unit when a friendly unit is on cooldown. The force of the field is repulsive if you are inside the enemy unit's maximum shooting distance in addition to a threshold, and attractive while outside. The result is that the unit is kept a certain distance outside the enemy unit's MSD, so that it can avoid being attacked by the enemy unit while being able to move in for the attack as soon as the cooldown is over. Because of this it is desirable that this distance is short, but not so short that the unit cannot escape.

$$force_{CoG} = w_5 \tag{6.3}$$

Equation 6.7 shows the function used for the COG Potential Field. It is only active while on cooldown, because attacking takes priority when the unit able to attack. However when on cooldown all the forces are summed to decide the direction of where the unit should move. The COG Field is used to guide the unit towards the other friendly units, which is a part of the positioning challenge. If gathered together units have less of a chance to be picked off. The field is constant and only dependent on the weight, which represents the importance of moving towards the COG.

Before implementing the functions they were simulated in Excel to see if the functions created the desirable Potential Fields.

6.3 NSGA-II

NSGA-II will be used to tune the weights used in the Potential Fields functions. Because of the complexity of the problem (defined in Section 1.3) several objectives are needed, and multi-objective optimization is a good way to solve problems with multiple objectives (Coello et al., 2007). It is also a way for us to improve upon the solution presented in Subsection 4.2.

The objectives represent how well the AI performs the Micromanagement techniques. These behaviours are: positioning, focus firing retreating and staying alive as discussed in Section 3.2. The NSGA-II algorithm will optimize the PF functions' weights using fitness functions based on the objectives. Parameters like mutation rate and population size will be tuned manually.

6.3.1 Objectives

The objectives are:

Focus firing attacking a weaker unit.

Positioning moving the units to their optimal positions.

Tactical retreating moving away from enemy firing range when on cooldown.

Staying alive retreating more further when unit's HP is low.

Representing the objectives directly in fitness functions is not feasible, but implicitly doing so is possible. The resulting fitness functions reward one or several of the objectives at once, but care has been taken to make sure they are not overlapping more than necessary.There has also been an attempt to make them as generic as possible to allow for new behaviours. There are a total of four fitness functions who focus on different parts of the performance.

$$fitness_{KS} = \frac{\sum_{n=0}^{noKilledUnits} valueUnit_n - \sum_{n=0}^{noLostUnits} valueUnit_n}{\sum_{n=0}^{noEnemyUnits} valueUnit_n * 2} \quad (6.4)$$

The killscore ($fitness_{KS}$) is a value calculated by StarCraft at the end of each match. It is based on how many units lost compared to units lost by the opponent, and takes into account the relative strength of each unit, giving a higher score to units who are stronger. n represents a unit, identified by a number between 0 and the total number of killed or lost units. $valueUnit$ is the value of the unit. Because the total value of the enemy units and the total value of the friendly units is not necessarily the same, the value range of $fitness_{KS}$ varies according to what units are fighting.

The killscore awards all the objectives and is a general measure of good Micromanagement. Focus firing is rewarded because focus firing kills enemy units faster than not focus firing. The other objectives are based on survivability and surviving means loosing fewer units.

$$fitness_{HP} = \frac{\sum_{n=0}^{noUnits} \frac{HP_n}{max(HP_n)}}{noUnits} \quad (6.5)$$

The HP fitness ($fitness_{HP}$) is the average HP remaining at the end of the match divided by the maximum HP possible. HP_n is the HP of unit n and $max(HP_n)$ is the highest HP the unit can have. The fitness is a measure of how many hits the units have taken and especially rewards tactical retreating. Of course, it is a general reward as all the fitness functions give off bad values given a loss. The HP fitness will output 1 if no friendly units

took any damage during the match. If the match was lost, the fitness will return the remaining enemy units' HP as a negative value, resulting in -1 if no damage was dealt by the friendly units.

$$fitness_{FF} = \frac{\sum_{n=0}^{noUnits} \frac{damage_n}{max(damage_n)}}{noUnits} \tag{6.6}$$

The focus fire fitness ($fitness_{FF}$) function takes the current damage output and divides it on the maximum damage output possible up to this point. $damage_n$ is the damage a unit n has done this far. The result will be between 0 and 1 and represent how large a percentage of the total possible damage the group of units have done.

This fitness function rewards attacking as often as possible, and though it does not reward focus firing explicitly it is likely to boost it because it makes sure everyone attacks. When everyone attacks focus firing is more likely to occur. Non-attacking unit will be heavily punished by this function. There is no explicit focus fire fitness function, but it is encouraged in the PF functions, and if it is a useful tactic it will be rewarded because fewer units die.

$$fitness_{SA} = \frac{\sum_{n=0}^{noUnits} \frac{timeAlive_n}{max(time)}}{noUnits} \tag{6.7}$$

The last fitness function records how long a unit managed to stay alive in relation to the total time. It is averaged over all units because it is the performance of the group as a whole that is relevant. Focus firing, positioning and staying alive is only effective if two or more units are cooperating. The fitness function rewards survivability and is meant to promote moving back while hurt and let other units with more HP take damage, as well as positioning correctly when on cooldown.

6.3.2 Genome Representation

The genotypes in the population are normalized bitstrings. Since the different PF weights will have different proportions as described in the previous subsection, the decoding algorithm has a set of parameters telling it the range and number of bits dedicated to each of the chromosomes. E.g. one weight might have the range [0.10] represented by 8 bits, which will allow it to differentiate up to circa 0.04 accuracy.

6.3.3 Evolutionary Operators

For the NSGA-II implementation mutation, single-point crossover and binary tournament selection was used. Mutation is done on the individual-level, meaning that an individual has one chance for one of its genes to be mutated and nothing more. Crossover has one chance in the breeding process to do a single-point crossover, if this occurs a random point between the chromosomes is chosen and recombination is done on two parents to create two recombined children as Figure 2.4 shows.

Chapter 7

Experiments

In this chapter the experimental setup and the experiments will be presented. The results will be thoroughly discussed in relation to the experimental setup and the implementation. In the experiments we wish to tune the Potential Fields to different combat scenarios consisting of different unit compositions, which is defined in Section 7.2. Each scenario will have its own training session and its separate generation data to work with. Generation data will not be exposed to a mix of testing scenarios, which means that two different scenarios will produce two different sets of potential solutions over two different training sessions. The reason for this is that different variations of unit compositions fighting each other will have different properties and therefore different potentials for what can be accomplished with Micromanagement (see Section 3.3). It is impossible to test every variation of scenarios since there are so many, but by having a set of general scenarios it will be easier to observe the difference in performance and results. The first scenario is chosen because it is the best unit composition for utilizing all the Micromanagement techniques at once. The second scenario will be one used by Sandberg (2011) so that the Potential Fields solution can be compared with theirs.

The questions these experiments attempt to answer are:

- Does the Potential Fields solution optimized by Multi-Objective Optimization (NSGA-II) perform better than one optimized by Genetic Algorithms?
- Does the AI perform better than the AI created by Sandberg (2011)?

These questions are designed to answer the research questions (see Section 1.3) using the techniques presented in Chapter 6.

The first questions will be answered by creating two Potential Field solu-

tions. One evolved by a NSGA-II and one evolved by a GA. The two will be compared by the results and the data gathered from the training process.

To be able to answer the second question the AI has to be tested in a similar setting as Sandberg (2011). In his report he describes numerous experiments, unfortunately the results of these experiments are not represented in a way that makes them easily comparable. Another problem is that reconstructing the exact same experimental environment is difficult because some of the experimental parameters are not documented, and the map used for the experiments must be reconstructed based on description and assumptions. A similar experiment will have to suffice. In the experiments by Sandberg the successfulness of a match scenario is described by units lost and units remaining, which will be used as result comparison for the second of the experiment in this report.

7.1 Experimental Setup and Configurations

This section will describe the preparations, the configurations and how the experiments were to be conducted.

7.1.1 Test Platform

All tests were run on a desktop computer with 3GHz Intel Core 2 Duo CPU, 4GB RAM and 64-bit Windows 7. The following software was installed in order to perform the experiments:

- StarCraft: Broodwar retail version patched to 1.16.1
- BWAPI 3.6.1 (revision 3769)
- BWSAL 0.9.12
- Boost C++ Libraries 1.49.0
- Chaoslauncher 0.5.4.1

To speed up testing, the system doing the experiment ran multiple instances of StarCraft. All the instances loaded and saved data from a shared XML file, allowing each instance to pick an untested individual from the population. This required that one instance acted as an *evolution master*, meaning that when all the individuals in the population were tested the *evolution master* would produce a new generation to the XML file while the other instances waited. The runtime of each test was made additionally faster with the BWAPI function *setLocalSpeed(0)*, meaning zero delay between

each game frame. Additional configurations such as disabling CPU throttling, the sound engine and user interface has also been made to increase the testing performance.

7.1.2 Evolutionary parameters

To find the most suited crossover and mutation rates the AI was evolved with six different sets of parameters. The population was evolved over 20 generations in order to distinguish and compare the result data. The reasoning behind the fixed number of solutions was based on trial and error, after looking at a couple of graphs from different mixtures of population size and number of generations it was found that 20 generations with 40 individuals was sufficient information to distinguish between evolutionary progressions with different pairs of crossover and mutation parameters.

Since the a match is not deterministic the more matches a genome is tested on the more accurate and representable the average results will be, for the results in this report each genome has been tested two times. This number comes from the experiment attributes that was used by Sandberg (2011).

It took around 2 hours to reach the 20th generation. The unit composition used in these tests were 9 Vultures controlled by the AI versus 9 Hydralisks controlled by the built-in StarCraft AI.

The collection of result data from the six different test runs can be found in Appendix A. Figure 7.1 shows a comparison of the highest performing set of parameters versus one of the lowest performing sets. Judging by the result data and the growth in fitness shown shown in the figure, evolution with 3% mutation and 15% crossover proved to be the best set of parameters.

$Fitness_{KS}$ is a strong indicator of the results of a match. The higher $fitness_{KS}$ is the more enemy units have been killed compared to how many friendly units have been lost. Thus this is the most important fitness value when deciding which solution has the best performance. The other objectives are important for the development of the NSGA-II algorithm, as they award different techniques, but not as important for deciding the best individual. A steady growth in the objectives is however important, as it indicates a non-random development and implies that they can converge towards a good value.

The results vary quite a bit and stays within a smaller value range than expected. This will be further explored in Section 7.4 and 7.5.

The NSGA-II algorithm is quite different from a normal GA algorithm in the way that it uses more elitism (see Section 2.3.1), as well as the mutation

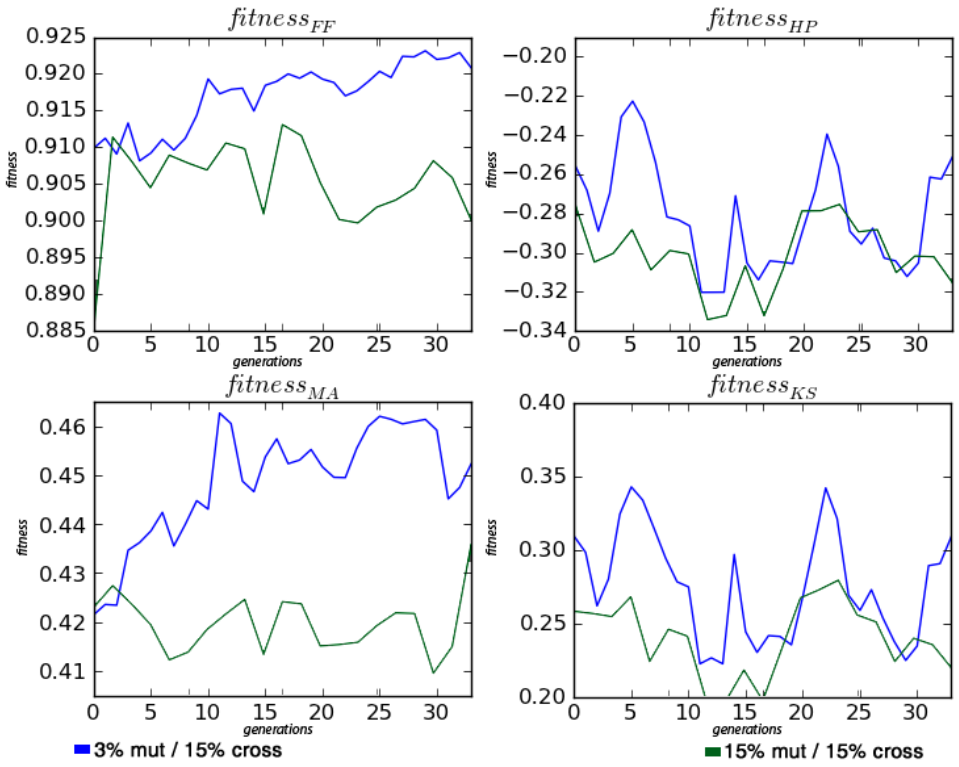

Figure 7.1: Comparison of the best results, 3% mutation 15% crossover, and one of the worst results 15% mutation 15% crossover.

EVOP being performed on the individual level. Thus a higher than normal mutation rate and crossover can be safely used to explore diverse solutions while still keeping the best solutions.

7.1.3 Potential Fields functions

For the experiments the ranges of the weights used in the Potential Field functions need to be determined. Since several functions use more than one weight, they need to be proportionally correct to each other in order to reflect their magnitude of importance.

For example in $force_{attack}$ (Equation 6.1) there are two parts of the calculation. $w_4 * \frac{1}{distance}$ is the most significant part, and will be of a higher range than $w_3 * (1 - \frac{eHP}{max(eHP)})$. At the same time the last part needs to have a relatively high maximum value in order to not be disregarded completely. w_4 will have a higher maximum value than w_3 to avoid running through enemy units as discussed in Subsection 6.2.1.

However as all weights have a minimum value of 0, the weight w_4 can still

be disregarded if it is found to be unimportant. This opens for emergent behaviour. For example it is possible that focusing down enemies with low HP does not help win a match faster and the evolved solution will move that weight towards 0.

All the weight ranges have been set in relation to each other by manually calculating the Potential Field outputs in theoretical scenarios.

The weight ranges are showed in the following table:

Weight	Minimum	Maximum
w_0	0	50
w_1	0	1000
w_2	0	10
w_3	0	400
w_4	0	1000
w_5	0	500

Table 7.1: unit data table

w_3 and w_4 are in $force_{attack}$ (Equation 6.1) which decides which enemy unit to attack. It is the only Potential Field in effect when not on cooldown and as a result the size of these two weights are only relative to each other. As mentioned, w_4 should have a higher range than w_3. The maximum value has been set to 1000 and 400 respectively.

w_0 in Equation 6.2 is the part of $force_{escape}$ that decides how much a unit should be attracted to enemy units if outside their MSD in addition to a threshold. w_2 is the weight that affects this threshold. This Potential Field is only in effect when the unit is on cooldown. The other Potential Field that is active while on cooldown is $force_{CoG}$ (Equation 6.7), which is only controlled by w_5.

During combat the unit will be in close proximity to several friendly units and each of these units will emit the attractive field (whose value is controlled by w_0). The sum of these fields will be cominded into one force vector and thus w_0 should have a small value. The maximum value chosen is 50. This is different from $force_{CoG}$ which has one field for all friendly units.

w_2 decides the threshold and $1 - \frac{hp}{max(HP)}$ decides which percentage of the threshold is to be used. Thus w_2 needs to represent the maximum threshold distance. None of these weight ranges were tested in game as they are so numerous and dependant on each other.

When $distance - max(MSD, eMSD) + w_2 * (1 - \frac{hp}{HP_{max}}) < 0$ the field $-\frac{w_1}{distance}$ is placed on the enemy unit(Equation 6.2). It does however com-

pete with the other enemy units' attracting fields (since the unit is probably outside many units' $max(MSD, eMSD)$), so the maximum value should be high to allow for this behaviour. This is shown in Figure 7.2, where one enemy unit is within the unit's MSD, and thus decides to run away. When running away it is wanted that the unit runs towards the other friendly units for protection, thus w_5's maximum value should not be enough to override w_1 but still enough to affect it.

Figure 7.2: The orange circle shows the MSD of the unit and the blue circles shows the MSD of the enemy units. In the experiments a blue line shows the direction the unit is moving in when running away

7.2 Match setup

The matches will all share the same map designed by the authors using the StarCraft Campaign Editor, which is the official program for making StarCraft maps. The map is 192x192 tiles each tile in StarCraft is a 32x32 pixels area, and it is configured for two players to start with a predefined set of units placed a short distance away from each other. One player is the Potential Fields AI while the other is the built-in AI that StarCraft uses. The terrain in this map is lower ground and there are no obstacles placed

anywhere besides the walls that encloses the edges of the map.

The different experiments will involve several unit compositions, the properties of the different units can been seen in Table 7.2.

Unit	Race	HP	Armor	Attack damage	Range	Cooldown
Hydralisk	Zerg	80	0	10	4	15
Vulture	Terrain	80	0	20	5	30
Zealots	Protoss	100	1	16	1	8
Dragoon	Protoss	100	1	20	4	30

Table 7.2: Unit data taken from the StarCraft: Broodwar Expert Guide [1].

All experiment results are stored in XML files, and simple python scripts were created by the authors to produce graphs by parsing these files. The plotting framework *matplotlib*[2] was used in these scripts.

7.3 Challenges

Due to some instability in the running process the game would occasionally crash during training. Because of this the testing platform also had a daemon script running, which would clean up after a crash and restart the training procedure, so that constant supervision over the experiments was not needed.

BWAPI is not bug-free and still under development, which has caused some instability and random crashes due to memory leaks and various bugs. The nature of EA training demanded that the game needed to be restarted a large amount of times, this caused some challenges. One of which was the actual restart mechanic in the BWAPI, which would sometimes just fail and the bot would be stuck in the score screen (the end screen after a match is complete) without the ability to restart until a supervisor could do it manually. The other major bug lies within StarCraft itself; it turns out that the game has a memory leak that is only an issue if the game plays thousands of matches repeatedly within a single instance. This bug was never reported during the 19 years of the official StarCraft maintenance since no human player would ever play that many matches continually (Sandberg, 2011). Some of these bugs were not possible to handle with the daemon script, which meant a test could stop at any time and require manual restart, although this happened rarely.

[1] http://www.gamefaqs.com/pc/75249-starcraft-brood-war/faqs/2473
[2] http://matplotlib.sourceforge.net

The C++ language also posed a challenge getting accustomed to as none of the authors had experience with the language to any extent before working on this thesis. Developing in the CBR architecture described in Section 5.3 also proved challenging since there was a lot of code to understand, and it also made the architecture of the Micromanagement AI a bit more complex than what was strictly necessary.

7.4 Experiment 1: NSGA-II vs. GA

To address the first experiment question an AI evolved with NSGA-II will be compared to one evolved with GA. A conclusion will be made by comparing the fitness rate progression as well as comparing the win rate of the fittest solution found for each method. The GA implementation uses the same binary tournament selection as NSGA-II, and will have some of the same evolutionary parameters. As Subsection 7.1.2 concluded, the best parameters proved to be 3% mutation and 15% crossover. To make the GA more comparable, $fitness_{KS}$ was chosen as its fitness function. The reasoning behind this is that $fitness_{KS}$ is based on the in-game unit value, which makes it the most representable value for the end-result of a match.

Once the different variations have been evolved, the best individual from the last generation will play ten matches under observation. During observation, the game speed will be turned down from the training speed described in Subsection 7.1.1 to a normal speed. Since the Potential Fields functions have been designed from the bottom up and the intended behaviour already defined, it should be possible to identify the cause behind the behaviour. Of course, emergent behaviour would also be very interesting to see.

For an AI to be deemed better than another it needs to win more consistently. Table 7.3 and Table 7.4 shows the NSGA-II and GA experimental set-up respecively.

Experiment parameters	
Controlled units	7 Dragoons
Enemy units	7 Zealots
Matches	10
Population	40
Generations	40
Mutation rate	3
Crossover rate	15

Table 7.3: Experiment 1: NSGA-II

Experiment parameters	
Controlled units	7 Dragoons
Enemy units	7 Zealots
Matches	10
Population	40
Generations	40
Mutation rate	3
Crossover rate	15

Table 7.4: Experiment 1: GA

7.4.1 Results

The immediate difference seen in these results is the fast improvement curve within the first five generations. The Dragoon versus Zealot unit composition is a good scenario for testing the Micromanagement techniques. A basic use of shoot and retreat will enable the Dragoon to stay alive longer while dealing damage because the attack ranges of Zealots and Dragoons are so different. This can explain the rapidly increasing curve in the first generations in $fitness_{KS}$ in Table 7.3 and 7.4 since it is less challenging for evolution to get the roughly correct weights than to perfect them.

Another interesting aspect in the results is the decrease in $fitness_{HP}$ in comparison with the parallel increase in $fitness_{KS}$. This could be the result of the units prioritizing the focus fire technique. Kill score only increases as a unit dies, not when it is damaged, while $fitness_{HP}$ represents overall HP lost. Focusing on a unit with low HP will ensure that the unit dies faster and that there is one less enemy unit on the battlefield that can damage friendly units. However, moving into position to attack a weak unit could cost time that can otherwise be used to deal damage to another enemy unit that might not die during the battle. As a result, focusing fire might increase the kill score as more units get killed, but can also decrease the damage done to surviving enemy units.

Figure 7.4 shows little progress in terms of mean fitness. Comparing it to $fitness_{KS}$ in Figure 7.3 indicates a clear improvement in favour of NSGA-II. However, this difference became less distinct when the best individuals from each end-generation played their ten matches versus the built-in StarCraft AI.

The results in Table 7.5 are quite similar and the two AIs loose most of the matches they play. The manual preliminary tests done by the authors

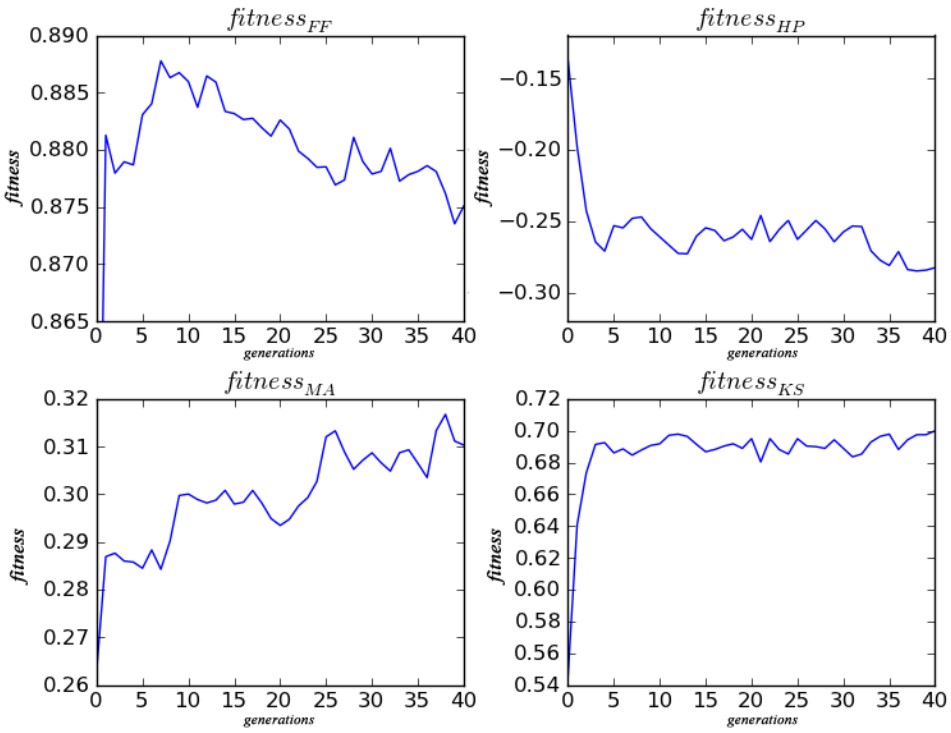

Figure 7.3: MOGA evolved 7: Dragoons versus 7 Zealots. Mutation rate 3%, Crossover rate 15%.

GA		MOGA	
Win	1	Win	2
Loss	9	Loss	8

Table 7.5: Win and loss rates in Experiment 1.

with this unit composition showed that this match up was winnable if good Micromanagement was used. This and the observations made during the tests conclude that neither of the AIs Micromanage optimally. The rest of this section will discuss the observations and explore the possible causes for the results.

During the observation of the matches the Micromanagement techniques intended were present, but also some behaviours that were not intended. When the Dragoons first engage the Zealots they move as a cluster right into the Zealots, and have trouble retreating when on cooldown, often blocking each other from moving away. This is of course not the wanted behaviour and it does not take advantage of the superior attack range of the Dragoons. However, when a few units remain and they are further apart from each other, the Dragoons improve their retreating technique radically. Alas it is

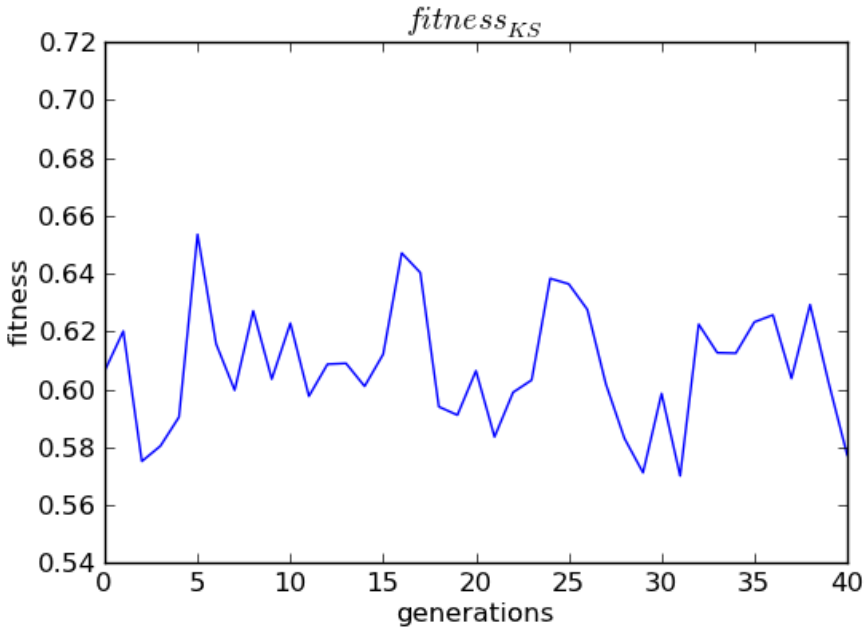

Figure 7.4: GA evolved: 7 Dragoons versus 7 Zealots. Mutation rate 3%, Crossover rate 15%.

often too little too late. The reason for this behaviour can be numerous and will be further explored in Subsection 7.5.1.

One technique that seemed to work well was focus fire. The units cooperated on destroying enemy units with low HP that were within reasonable range as designed by Equation 6.1. This behaviour is shown in Figure 7.5, where the red lines show which target a unit is going to attack. Several units are targeting the same enemy unit even if other enemy units are closer. There were also observed a few examples of a unit with high HP stepping closer to the enemy so that the enemy switched target from an escaping unit with low HP. This behaviour is due to the escaping field described by Function 6.2.

It was stated in Section 7.2 that each genome will run two times, but the more times each genome runs, the more precise the fitness will be. This is due to the fact that the built-in StarCraft AI possesses some random behaviour. At times it will suddenly halt its units' movement for no apparent reason. Randomness like this might result in a lucky win, causing noise in the fitness evaluation and making it difficult to determine if a solution is consistently high performing based purely on the fitness values.

The results produced in this experiment proved NSGA-II to have a better learning curve than GA in terms of the objective functions. However in terms

Figure 7.5: Four Dragoons focus firing one Zealot, the remaining two are on cooldown and are running away.

of win ratio it did not prove its superiority adequately enough in comparison to the GA solution. This indicates that there is something wrong with the Potential Fields part of the solution, as both AI techniques struggled more than would be expected. To further explore this we conducted a new experiment, comparing the GA solution to Sandberg's solution. As they both use the same AI (genetic algorithms) method we will be comparing the Potential Fields parts of the solutions.

7.5 Experiment 2: GA vs. EMAPF

In this experiment the Micromanagement AI described in this thesis will be compared to Sandberg (2011)'s EMAPF AI. One of the experiments in his thesis will be replicated and the results compared. The information about the experiment is shown in Table 7.6.

Sandberg (2011) did this experiment with three different solutions. With the best solution he lost only one unit in the engagement. The success of the experiment will be determined in relation to his result. The best result

Experiment parameters	
Controlled units	9 Vultures
Enemy units	9 Hydralisks
Matches	10
Generations	40
Mutation rate	3
Crossover rate	15

Table 7.6: Experiment 2

from the 10 matches will be used for comparison, as this is what Sandberg does in his experiments.

Since the map and the unit placement is different from Sandberg's, the experiment won't be completely fair, and this will be taken into consideration.

7.5.1 Results

Solution	GA	EMAPF
Remaining units	5	8

Table 7.7: Remaining units in Experiment 2.

In Sandberg's best run he has 8 of 9 units alive after the match, while the GA has 5 units remaining in the best run. While the difference is not huge it is big enough to determine that the Potential Fields solution in this thesis is not as good as Sandberg's EMAPF solution.

9 Vultures vs 9 Hydralisks	
Win	3
Loss	7

Table 7.8: Win and loss rates in Experiment 2

Although this experiment produced better results than the previous experiment, the results are still not satisfactory when looking at the average win rate. A highlight is, however, that the AI did very well the few times that it won; the best match left the AI with five remaining units, the second best with four. The results enforce the implication that the Potential Fields part of the solution is lacking and is not as good as the EMAPF solution in its current form.

Sandberg has shown the match in question in a Youtube video[3] which makes it possible to compare the behaviour observed during the experiment with the behaviour of his AI. During the match his units behave in the following fashion: the units (Vultures) move towards the middle of the map. When they encounter the enemy units (Hydras) the Vultures spread out and surround them, keeping distance between the friendly units. They move in to attack and and then retreat over and over again, getting attacked less than if they were standing still. A snapshot of this behaviour is shown in Figure 7.6. Focus firing is also observed during the test. There is no collision between friendly units in the video.

Figure 7.6: The units in the EMAPF solution are spread around the enemy units.

The biggest difference between the behaviours of the EMAPF and the GA is in the beginning of the match, where the units cluster together instead of spreading out and surrounding. There are several possible reasons for this:

- Center of Group attraction.
- The enemy unit repulsion is too weak.

[3]http://www.tinyurl.com/9vulturesvs9hydralisks

- No collision avoidance.
- The thresholds are too small.

The potential issue with Center of Group attraction is if the center of the group is too close to the enemy units. Then it will encourage the units to stay close to the group instead of moving as far away from the enemy units to avoid getting attacked. It is also a problem if the units are surrounding the enemy. Then the COG will be placed in the middle, encouraging the units to move towards the enemy instead of running away.

The potential issue that the enemy unit repulsion is too weak is linked to the issue with Center of Group attraction. The attraction is meant to steer the unit towards other friendly units when retreating because of the fields around the enemy units. However, if the fields around the enemy units are not strong enough the Center of Group attraction becomes more powerful than intended.

During the experiments we observed collisions between units, especially at the start of a match when they were clustered together. Even if some units attempted to run away from the enemy they were not able to because other friendly units were blocking their exit path. This is something that Sandberg (2011) does not struggle with (at least not from what is seen in the Youtube video), since Sandberg has small repulsing fields around each friendly unit to avoid these collisions.

As mentioned in Section 7.1.3 the threshold weight is the weight that decides how far away from the $max(MSD, MSD_e)$ value the unit should go when retreating, relative to it's remaining HP. If this threshold was larger, the units would move even further away, making sure there would be a safe distance between the unit and the enemy units.

Each of these potential causes can be addressed. The center of group attraction can be removed altogether, or given a much smaller maximum value, by changing the value range of w_5. Conversely, weak enemy unit repulsion can be addressed by increasing the weight range of w_4.

Collision avoidance can be handled by creating a new Potential Field to be added on friendly units. The implementation is designed in such a way that this addition is simple and should only take about an hour to implement and test. To address the threshold issue, the threshold weight's (w_2) maximum value can be increased.

7.6 Discussion

To answer the experiment questions the results need to be discussed, and to explain the results the data from the experiments conducted needs to be understood.

7.6.1 Objective functions

The graphs for the development of the mean objective fitness is a good indicator as to how well the EA performed. However there is some steady growth in the different objectives. This is especially true for $fitness_{FF}$. This is the objective function that rewards focus firing, a technique seen executed quite well during the experiments.

$fitness_{MA}$ also grows steadily. This implies that the bot becomes increasingly good at keeping its units alive for a longer time. This is both an indication of generally better play, the tanking behaviour seen in the experiments and that the units are better at retreating from enemy fire. While the latter is not optimal during the whole match, there is a clear improvement from an unevolved solution.

$fitness_{HP}$ and $fitness_{KS}$ are very tied together and output quite similar values, especially visible in Figure 7.1. These values vary the most and is also the ones prone to instability due to the non-deterministic nature of the StarCraft game. Especially since these graphs are from the Vultures vs. Hydras scenario, which is a more random and difficult set-up than the scenario run in Experiment 1. Also the lacking performance in the Potential Fields causes these values to vary more than expected because it is impossible for the Micromanagement AI to develop optimal behaviour.

7.6.2 Potential Fields functions

The weights have a tremendous importance and is the backbone of the whole solution. The weights had to be constrained to different ranges because of the nature of the PF functions and to reduce the search space for the EA. The functions were structured simplistic and normalized so it should be easier to identify the important factors, and rather let the weights connected to the factors decide just how important they are in proportion to each other. Because of the time used to evolve a solution it is difficult to tell if the ranges of the weights are correct, or if they should be larger or smaller. Incorrect weight ranges could be a reason that the Potential Fields did not perform optimally.

Except for the two weights in question (w_4 and w_2), the weight ranges seems to have been set correctly based on observation made from the experiments. The reason for the limited performance is likely to be the implementation of the Potential Fields. The solution improves rapidly in early generations, but quickly stagnates to a mediocre performance with behaviours that are not intended according to the PF designs. Despite supervising the early PF outputs during development, these flaws were not properly identified until late into the experiments.

7.6.3 MOGA vs GA

The training data showed that MOGA evolved better and were able to optimize all four objectives at once. The GA solution evolved more randomly and was not able increase its average fitness in the population. Despite the indication of increased performance with NSGA-II over GA the experimental results were not enough to prove this claim.

Chapter 8

Conclusion

Success builds character, failure reveals it.

This thesis presented a Potential Fields based Micromanagement AI for the RTS game StarCraft, optimized using NSGA-II. The goal as the authors started working on this thesis was to find a way to improve already existing results presented by previous RTS Micromanagement bots using Potential Fields. There was no previous solutions utilizing Multi-Objective Optimization to develop PFs, so developing one would be a step forward in the domain.

In the end the experiments yielded disappointing results in terms of practical performance and the AI did not perform better than previous bots. However as discussed the AI learns rapidly up until a certain point in its performance. The question whether this problem lies in the design or the implementation of the Potential Fields was raised. Which would explain the lacking performance of the AI regardless of its weight values.

After evaluating the results of the conducted experiments, the authors revised the possible causes of error suggested in Subsection 7.5.1 and created two new revisions of the AI with a mixture of increasing weight thresholds, removing the Center of Group field and adding a repulsive collision avoidance field at each friendly unit. The new revisions were given a short ten generations to evolve, and then tested to check for improvements. Unfortunately neither of the revisions showed any improved performance, but they did help to empower the suspicion that the fault does indeed lie in the fundamental implementation of the Potential Fields, and not within the design itself.

8.1 Further work

For further work it would be interesting to see a different design approach to the Potential Fields controlling the Micromanagement. Fields that expand behaviour like collision avoidance as well as taking more detailed aspects of the game into the calculations like terrain, obstacles, and enemy unit cooldown in the case of StarCraft.

It would also be interesting to see a more thorough training procedure. If more time is available each individual can be tested more than two times for higher accuracy. Co-evolution is also a possibility and would probably have a great impact on the behaviour of an optimal solution. Another approach could also be training the AI versus an experienced human player, because the built-in StarCraft AI performs far from optimally and employs few different tactics in Micromanagement.

The source code for the EMAPF bot created by Sandberg (2011) is available on-line, extending it to using NSGA-II with the objectives presented in this report would prove more direct and conclusive results of how MOO affects the performance.

The results in this report indicate that Multi-Objective Optimization is indeed a valid approach worth exploring for tuning Potential Fields in RTS games. StarCraft has proven an excellent test platform, but the methods are applicable for almost any RTS game because of the generic nature of the genres domain rules and environment. It would be interesting to see this approach attempted in a different RTS game.

Appendix A

Results: mean fitness

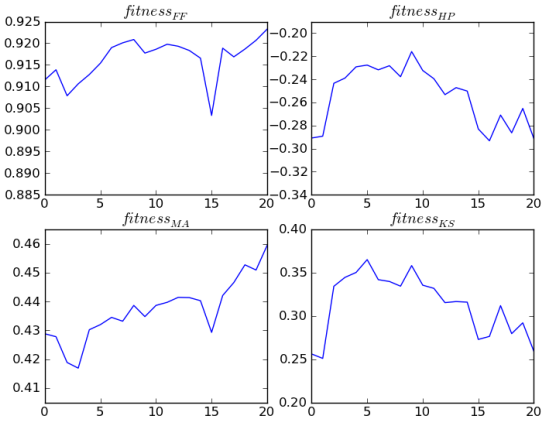

Figure A.1: 20 generations. 9 Hydralisk versus 9 Vultures. Mutation rate 3%, Crossover rate 3%.

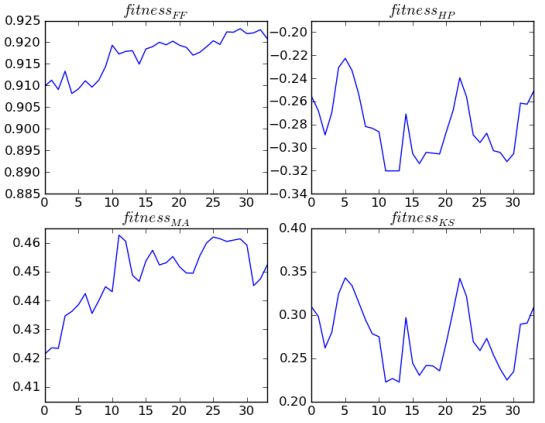

Figure A.2: 20 generations. 9 Hydralisk versus 9 Vultures. Mutation rate 3%, Crossover rate 15%.

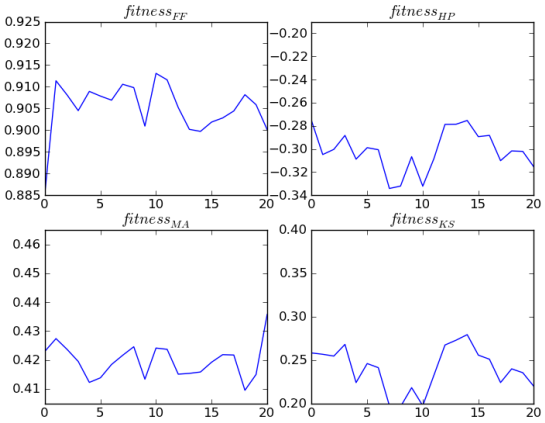

Figure A.3: 20 generations. Mutation rate 15%, Crossover rate 15%.

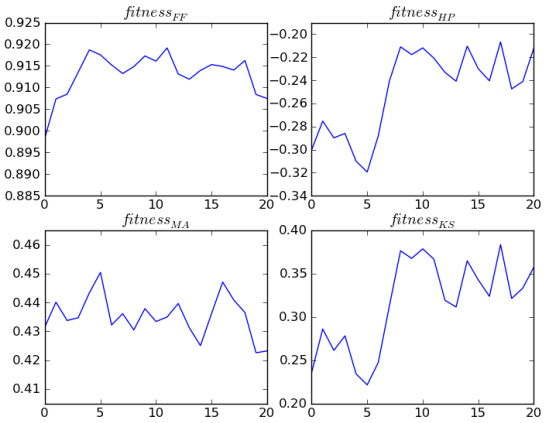

Figure A.4: 20 generations. Mutation rate 15%, Crossover rate 50%.

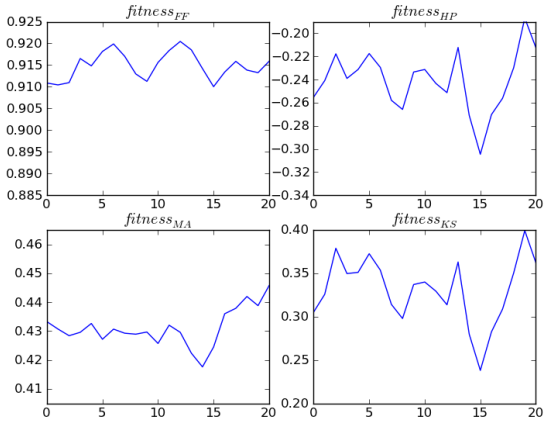

Figure A.5: 20 generations. Mutation rate 50%, Crossover rate 50%.

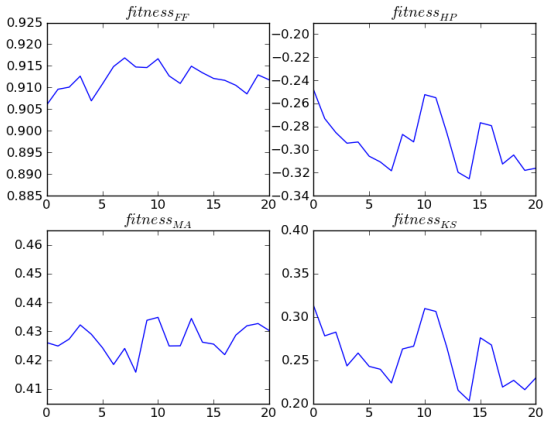

Figure A.6: 20 generations. Mutation rate 50%, Crossover rate 100%.

Bibliography

Coello, C., Lamont, G., and Van Veldhuizen, D. (2007). *Evolutionary algorithms for solving multi-objective problems*, volume 5. Springer-Verlag New York Inc.

Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. *Lecture notes in computer science*, 1917:849–858.

Downing, K. L. (2010). Introduction to evolutionary algorithms.

Fernandez-Ares, A., Mora, A., Merelo, J., Garcia-Sanchez, P., and Fernandes, C. (2011). Optimizing player behavior in a real-time strategy game using evolutionary algorithms. In *Evolutionary Computation (CEC), 2011 IEEE Congress on*, pages 2017–2024. IEEE.

Floreano, D. and Mattiussi, C. (2008). *Bio-inspired artificial intelligence: theories, methods, and technologies*. The MIT Press.

Fogel, L., Owens, A., and Walsh, M. (1966). Artificial intelligence through simulated evolution.

Hagelbäck, J. and Johansson, S. (2008). Using multi-agent potential fields in real-time strategy games. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pages 631–638. International Foundation for Autonomous Agents and Multiagent Systems.

Holland, J. (1975). *Adaptation in natural and artificial systems*. Number 53. University of Michigan press.

Howard, A., Mataric, M., and Sukhatme, G. (2002). Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. In *Proceedings of the 6th International Symposium on Distributed Autonomous Robotics Systems (DARS02)*, pages 299–308. Citeseer.

Johansson, S. (2006). On using multi-agent systems in playing board games. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 569–576. ACM.

Johansson, S. and Saffiotti, A. (2001). An electric field approach to autonomous robot control. *RoboCup 2001*, (2752).

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research*, 5(1):90–98.

Kraus, S. and Lehmann, D. (1995). Designing and building a negotiating automated agent. *Computational Intelligence*, 11(1):132–171.

Lim, C., Baumgarten, R., and Colton, S. (2010). Evolving behaviour trees for the commercial game defcon. *Applications of Evolutionary Computation*, pages 100–110.

Priesterjahn, S., Kramer, O., Weimer, A., and Goebels, A. (2006). Evolution of human-competitive agents in modern computer games. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 777–784. IEEE.

Rechenberg, I. (1965). Cybernetic solution path of an experimental problem.

Safadi, H. (2007). Local path planning using virtual potential field. `http://www.cs.mcgill.ca/~hsafad/robotics/index.html`.

Sandberg, T. (2011). Evolutionary multi-agent potential field based ai approach for ssc scenarios in rts games.

Srinivas, N. and Deb, K. (1994). Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary computation*, 2(3):221–248.

Vadakkepat, P., Lee, T., and Xin, L. (2001). Application of evolutionary artificial potential field in robot soccer system. In *IFSA World Congress and 20th NAFIPS International Conference, 2001. Joint 9th*, pages 2781–2785. IEEE.

Zavlanos, M. and Pappas, G. (2007). Potential fields for maintaining connectivity of mobile networks. *Robotics, IEEE Transactions on*, 23(4):812–816.

Zitzler, E. and Thiele, L. (1998). Multiobjective optimization using evolutionary algorithms—a comparative case study. In *Parallel Problem Solving from Nature—PPSN V*, pages 292–301. Springer.