Applying Fuzzy Set Theory and Case-Based Reasoning Approach for Managing Strategical and Tactical Reasoning in StarCraft

by

Pedro Antonio Cadena Vázquez

Thesis

Submitted to the Graduate Program in the Division of Mechatronics and Information Technologies in partial fulfillment of the requirement for the Master of Science degree in Intelligent Systems

Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Monterrey

May 2011
This thesis is dedicated to my parents who have given me the opportunity of an education from the best institutions and support throughout my life.
Acknowledgements

It is a pleasure to thank those who made this thesis possible. I am grateful to my advisor Ph.D. Leonardo Garrido, whose encouragement, guidance and support from the initial to the final level enabled me to develop an understanding of the subject. I owe my deepest gratitude to my parents who were always supporting me and encouraging me with their best wishes. I would also like to thank the Research Chair grant on Intelligent Autonomous Agents CAT 179 of the Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Monterrey and the Conacyt scholarships. And overall I would like to thank God for giving me health, wisdom, courage and strength to reach my goals.

PEDRO ANTONIO CADENA VÁZQUEZ

Instituto Tecnológico y de Estudios Superiores de Monterrey
May 2011
Applying Fuzzy Set Theory and Case-Based Reasoning Approach for Managing Strategical and Tactical Reasoning in StarCraft

Pedro Antonio Cadena Vázquez, M.Sc.
Instituto Tecnológico y de Estudios Superiores de Monterrey, 2011

Thesis Advisor: Leonardo Garrido, Ph.D.

This thesis aims at describing, analyzing, implementing and discussing fuzzy set theory and Case-Based Reasoning (CBR) for strategical and tactical management in the real-time strategy game of StarCraft. In order to play a complete match of StarCraft this thesis divides the problem in four categories: resource management, strategical management, tactical management and micro organization. Case-based reasoning is a problem solving AI approach that uses past experience to deal with actual problems. A new problem is solved by finding a similar past case, and reusing it in the new problem situation. Fuzzy set theory is used in case representation to provide a characterization of imprecise and uncertain information. In this thesis, the combination of fuzzy sets and case-based reasoning is called Fuzzy Case-Based Reasoning (FCBR). CBR was applied to reason about strategies while FCBR was applied to deal with tactical reasoning. The resulting system was victorious in 60% of the games, it was defeated in 25% of the games, getting ties in 15% of the matches. The results revealed that our system can successfully reason about strategies and tactics, defeating the built-in AI of StarCraft. The principal conclusion was that FCBR can reason with abstract information and a large space of actions. Moreover, the resulting system shows its potential to incorporates human knowledge and can effectively adapt to varying conditions of the map.
## Contents

Acknowledgements v

Abstract vi

List of Figures x

Chapter 1 Introduction 1
  1.1 Motivation .................................................. 2
  1.2 Problem Definition ........................................ 4
  1.3 Hypothesis .................................................. 5
  1.4 Objectives and Methodology ............................... 6
  1.5 Organization ............................................... 6

Chapter 2 Background 8
  2.1 Case-Based Reasoning ...................................... 8
    2.1.1 Case-based problem solving ......................... 9
    2.1.2 Learning in Case-based Reasoning .................. 9
    2.1.3 The CBR cycle ....................................... 10
    2.1.4 Advantages of CBR .................................. 12
  2.2 Fuzzy Theory and its Convergence with CBR ............. 12
  2.3 StarCraft: Brood War ...................................... 14
    2.3.1 Overview .............................................. 15
    2.3.2 Game Play ............................................ 15
    2.3.3 Replays .............................................. 16
  2.4 BWAPI: An API for Interacting with StarCraft: Brood War 16
  2.5 BWSAL: Standard Add-on Library for BWAPI .......... 17
  2.6 BWTA: Brood War Terrain Analyzer .................... 18
  2.7 Summary .................................................. 18

Chapter 3 Implementing Fuzzy Sets and CBR approach in StarCraft 19
  3.1 Overview .................................................. 19
    3.1.1 Resource Management ................................. 20
Chapter 3

3.1 Strategic Management
3.1.2 Strategical Management
3.1.3 Tactical Management
3.1.4 Micro Management
3.1.5 Protoss Race, built-in AI Enemy and The Map of Python
3.1.6 The Fog of War

3.2 Strategical Reasoning
3.2.1 Strategy Selection
3.2.2 Case Representation
3.2.3 Case Extraction and Case Base
3.2.4 Retrieve Process and Matching Treat
3.2.5 Reuse Process
3.2.6 Revise Process and its Implementation
3.2.7 Summary

3.3 Tactical Reasoning
3.3.1 Space Abstraction
3.3.2 Fuzzy Case Representation
3.3.3 Implementing Fuzzy Sets in StarCraft
3.3.4 Case Extraction and Case Base
3.3.5 Retrieve Process and the Fuzzy Matching Method
3.3.6 Reuse Process
3.3.7 Revise Process and its Implementation
3.3.8 Summary

3.4 Micro Management Improvement
3.4.1 Chokepoint Defense
3.4.2 Target Path
3.4.3 Attack Coordination
3.4.4 Summary

3.5 Results
3.5.1 Selected FCBR AI Initial Position vs Selected Built-in AI Initial Position
3.5.2 Selected FCBR AI Initial Position vs Random Built-in AI Initial Position
3.5.3 Random FCBR AI Initial Position vs Random Built-in AI Initial Position
3.5.4 General Discussion
3.5.5 Summary

Chapter 4 Related Work
4.1 CBR for macromanagement in RTS games
4.1.1 Case-Based Goal Formulation
4.1.2 Case-Based Planning ................................................. 58
4.1.3 CBR for Build Order ................................................ 59
4.1.4 Building a Player Strategy Model ............................... 60
4.2 CBR for micromanagement in RTS games ....................... 61
4.2.1 Improving Micromanagement .................................. 61
4.2.2 Learning Continuous Actions Models ......................... 61
4.2.3 Transfer Learning .................................................. 62
4.3 Other Approaches in RTS games ................................. 62

Chapter 5 Conclusions ..................................................... 64
5.1 Research Questions Revisited .................................... 65
5.2 Contributions .......................................................... 66
5.3 Future Work ............................................................ 67

Vita ................................................................. 72
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>The CBR Cycle</td>
<td>10</td>
</tr>
<tr>
<td>3.1</td>
<td>Case representation for strategical reasoning</td>
<td>24</td>
</tr>
<tr>
<td>3.2</td>
<td>Case representation for strategies in the game</td>
<td>25</td>
</tr>
<tr>
<td>3.3</td>
<td>Lost Temple map divided into regions</td>
<td>30</td>
</tr>
<tr>
<td>3.4</td>
<td>Case representation for tactical reasoning</td>
<td>31</td>
</tr>
<tr>
<td>3.5</td>
<td>Case representation for tactics in the game</td>
<td>32</td>
</tr>
<tr>
<td>3.6</td>
<td>Membership functions for <em>area</em></td>
<td>34</td>
</tr>
<tr>
<td>3.7</td>
<td>Membership functions for <em>ckokepoints</em></td>
<td>34</td>
</tr>
<tr>
<td>3.8</td>
<td>Membership functions for <em>military presence</em></td>
<td>35</td>
</tr>
<tr>
<td>3.9</td>
<td>Membership functions for <em>combat intensity</em></td>
<td>35</td>
</tr>
<tr>
<td>3.10</td>
<td>Membership functions for <em>lost units</em></td>
<td>37</td>
</tr>
<tr>
<td>3.11</td>
<td>Membership functions for <em>numeric state</em></td>
<td>37</td>
</tr>
<tr>
<td>3.12</td>
<td>Intersection area between two membership</td>
<td>38</td>
</tr>
<tr>
<td>3.13</td>
<td>Results of <em>FCBR-Bot</em> in custom start locations</td>
<td>45</td>
</tr>
<tr>
<td>3.14</td>
<td>Scores of <em>FCBR-Bot</em> in custom start locations</td>
<td>46</td>
</tr>
<tr>
<td>3.15</td>
<td>Results of <em>FCBR-Bot</em> in custom start location vs random start location</td>
<td>48</td>
</tr>
<tr>
<td>3.16</td>
<td>Scores of <em>FCBR-Bot</em> in custom start location vs random start location</td>
<td>49</td>
</tr>
<tr>
<td>3.17</td>
<td>Results of <em>FCBR-Bot</em> in random start location vs random start location</td>
<td>52</td>
</tr>
<tr>
<td>3.18</td>
<td>Scores of <em>FCBR-Bot</em> in random start location vs random start location</td>
<td>53</td>
</tr>
<tr>
<td>3.19</td>
<td>Comparison between <em>FCBR-Bot</em> and <em>EISBot</em> in Protoss vs Protoss games</td>
<td>56</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Nowadays the term Artificial Intelligence (AI) has become popular. The main purpose of artificial intelligence is the creation of intelligent machines and intelligent agents. An important field that contributes to the development of AI is the video game context. Video games constitute an excellent medium to develop and test new and existing AI techniques. In role-playing games is common to represent the characters like computer controlled agents, in first-person shooter games the enemies are computer-controlled agents that combat against the player, in board video games the computer simulates an agent to make moves. In all games, the behavior, the actions, and the decision making by agents is determined by the game AI. For artificial intelligence researches, the development of game AI of complex modern games is a truly challenging application. Complexity in video games has increase with the development of better graphics, faster processors and better network connections. Now is easier to design dynamic worlds and more sophisticated characters that need to perform well in their environments. Recently, researches have developed several good techniques that give the agents the ability to learn. Those approaches are machine learning techniques. There exist a huge variety of intelligent algorithms that can be applied to the development of intelligent agents in video games. Reinforcement Learning methods, Neural Networks, optimization algorithms, path finding algorithms are some examples of intelligent techniques applied to video game AI. One genre of video games that has increase importance in the AI field is the real-time strategy type. Real-time strategy video games is a genre of games that do not progress incrementally in turns, the evolution of the game is in continuous time. In this type of games the decision process for strategy selection must be made in real time, involving to deal with incomplete information and a wide variety of data that represent the world in that moment. Different research works have been proposed to develop learning agents that play in RTS video games at different levels. Some of these approaches include Case-Base Reasoning (CBR). CBR is an approach to problem solving and learning that is able to utilize specific knowledge, of previously experienced problem situations. It is inspired by the human capability to solve problems by generalizing over previous observations in a restricted problem domain. When new problems appear the information of similar past problems is used
to solve them. The purpose of this research is to develop a system based on case-base reasoning approach with fuzzy attributes that can be able to play a real-time strategy game and defeat hard-coding agents. Our system is decomposed into two competencies: strategic and tactical management.

1.1 Motivation

Inside the AI community the RTS games genre offers a large variety of fundamental AI research problems. Artificial intelligence researches find the development of game AI of modern RTS games a truly challenging application. According with Buro and Furtak [2003] some important research problems are:

- Adversarial real-time planning. In real-time simulations the presence of a large space of objects and actions requires the use of agents that can deal with abstractions of the world state. Such abstractions of the state must allow to reason in a manageable abstract space and to translate found solutions back into action sequences in the original state space. Real-time environments present features that exhibit dynamic, hostile and smart behaviors. Adversarial real-time planning approaches need to be investigated in these environments in order to anticipate to enemy actions and accomplish reasoning in real-time.

- Decision making under uncertainty. RTS games enforce incomplete information through “the fog of war”. At the beginning of the game, players are not aware of the enemies’ base locations and intentions. To gather information about enemy actions, players must send scout units to explore the terrain and draw conclusions to adapt. Plausible hypotheses have to be formed and acted upon if there is no available data about opponent locations and actions.

- Opponent modeling, learning. Actual RTS game AI systems are coded with rule-based algorithms, AI agents with such approach are not appropriate to learn quickly. In contrast, human players only need few games to identify the weakness of the opponent and to exploit them in future games. Current machine-learning approaches that are based on statistics are inadequate for quick learning.

- Spatial and temporal reasoning. In RTS games terrain analysis is of utmost importance; static and dynamic analysis of the terrain as understanding temporal relations of actions is required to complex reasoning in RTS environments. Current AIs largely ignore these issues and lead to simple common-sense reasoning [Forbus et al., 2005].

- Resource management. RTS games propose several categories that divide the game play; players start the game by gathering resources to build up defenses and
armies, then players start to upgrade weaponry and to climb up the technology tree. During game, players have to balance the resources they spent in each category. Resource management is an important task to the development of bases and attack forces and it is crucial in any successful strategy. Therefore, resource manager is an important challenge to the development of AI in real-time strategy games.

- Collaboration. The actual development of technology enables RTS games to join players form different regions of the world. In RTS games groups of players can join forces and intelligence. Therefore, a challenging research problem is the player coordination to execute actions effectively by communication among the parties. It is common in mixed human-AI teams that the AI player behaves awkwardly because it does not take into account the actions of the human player. Moreover, AI players cannot infer human’s intentions and fail to synchronize attacks.

- Pathfinding. One of the challenges in RTS games is the need to find high-quality paths over the terrain of the map. A time ago, only a small fraction of the CPU time could be devoted to AI tasks, of which finding shortest paths was the most time consuming. Nowadays, the development of technology has made possible better graphic accelerators, which allow programs to spent more time on AI tasks. Nevertheless, RTS games present a large amount of moving objects and the need of more realistic simulators in RTS environments make it necessary to improve and generalize pathfinding algorithms. The simple problem of finding shortest paths becomes complicate because of the dynamism of the units over the map, enemy influence, inertia, and fuel consumption.

The understanding of fundamental AI problems is increased by the results of RTS game research, such as opponent modeling and adversarial real-time planning. Moreover, RTS game AI research has significant impact on the real-time control domain in general and the computer game industry in particular which needs to create reliable computer controlled agents [Buro and Furtak, 2003].

Other arguments in favor of AI research in RTS games, according with Buro [2004], are:

- Actual RTS games are well test-bed environments where researches can conduct experiments and offer a diversity of objective ways of measuring performance.

- RTS games can be configured to focus on different categories of the game such as how to win local fights, how to do a successful scout process, how to defend friend bases, how to conduct attacks over terrain enemy, how to deploy units across the map, among others.
• Future commercial games with a strong game AI will make a difference over games with a reduced AI because graphics improvements are beginning to saturate. Additionally, smarter AI enemies and allies are accessible at any hour of the day and can add a better game experience.

The creation of real-time systems capable of outperforming human experts in RTS games is a challenging task. Because search space abstraction, real-time planning, and temporal and spatial reasoning are central to many other problems, the scope of applications in RTS games seems endless.

Nowadays there are open source applications that enable to interact with an RTS game engine making possible for the user to retrieve information about game state as well as issue commands to units. These applications open the doors for creating custom AI’s and motivate the RTS AI research development

1.2 Problem Definition

The goal for players of RTS games is to build up armies capable of defeating enemy bases while simultaneously defending one’s base against enemy attacks. The object and action complexity, combined with real-time multi-scale play, provide a uniquely challenging domain for game playing agents [McCoy and Mateas, 2008].

A large number of unique domain objects and unique actions are presented in RTS games. In the category of domain objects it can be mentioned different type of mobile units, different type of buildings with varying defensive and unit production capabilities, modification of buildings, and resources that must be managed to construct buildings and units. In the field of actions it can be mentioned different kind of building and unit construction orders, choosing upgrades and tech development for units, resource management, and the actions to employ unit capabilities during battle. In an RTS game, actions can occur at multiple scale levels. High-level strategy decisions involve which type of buildings and units need to be produced, intermediate tactical decisions include how to deploy group of units across the map and low level micro-management decisions about individual unit actions. The complexity of RTS games grows for successful players that must engage in multiple, simultaneous, real-time tasks. It is typically that in the middle of a game, a player may be managing the defense and production capacities of several bases while being simultaneously engaged in one or more battles. Finally, incomplete information is enforced by RTS games in the form of “the fog of war” that hides most of the map. To actively gather information about enemy activities, the player who can only see areas of the map where he has unit, requires the deployment of scout units across the map [McCoy and Mateas, 2008].

According to the features of an RTS game mention above McCoy and Mateas [2008] state that these attributes of the domain argue for a game playing agent archi-
architecture that can combine strategic reasoning with real-time reactivity, and can incorporate human-level decision-making about multiple simultaneous tasks through multiple levels of abstractions. In addition, Buro [2004] mention that it is in these domains where the current commercial RTS AI systems fail because their absence of reasoning and adaptation while the human abilities to abstract, generalize, reason, learn, and plan shine. Moreover, Cheng and Thawonmas [2004] say that in RTS games, one of the principal challenges involves the numerous objects, incomplete information, micro actions, and fast paced actions presented in such environments. Also, Buro and Furtak [2003] state that AI in the RTS games domain is less than satisfactory because computer opponents do not smartly adapt to adversaries, do not learn from their mistakes, can not reason in abstract search spaces, can not deal with spatial and temporal object relation, and do not collaborate and communicate well. Additionally, Buro et al. [2008] say that some examples of challenges in RTS games involves the interaction of a large number of objects, spatial and temporal reasoning, acting under uncertainty in dynamic environments and the need for abstraction and planning.

Due to the attributes and complexity mention above, it is necessary to incorporate human knowledge for decision making in RTS environments. Moreover, it is required to reason with a large number of objects and actions in form of abstract search spaces. Additionally, in order to play an entire game and success defeating the opponent, it is indispensable to combine strategic and tactical reasoning. Finally, it is also crucial to accomplish reasoning with incomplete information in RTS games.

1.3 Hypothesis

An agent architecture based on Case-Base Reasoning (CBR) approach can deal with strategical reasoning in RTS games. Moreover, an agent architecture based on Fuzzy sets and Case-Base Reasoning (FCBR) is suitable to deal with tactical reasoning in RTS environments. Consequently, the use of CBR and FCBR enhances agent-based systems to defeat rule-based coded agents while playing entire matches in complex, uncertain and dynamic environments of an RTS game like StarCraft.

This leads to the formulation of the following research questions:

- How can CBR approach incorporate human knowledge for decision making?
- Is it feasible to treat the attributes of an RTS game like StarCraft as fuzzy variables?
- How can CBR approach deal with incomplete information?
- How can we combine strategic and tactical reasoning for an RTS game in CBR technique?
Is it the combination of fuzzy theory and CBR a good approach in the development of RTS game AI?

1.4 Objectives and Methodology

The main objective of this thesis is to build an agent-based system that incorporates case-based reasoning with fuzzy attributes to win the RTS game of StarCraft. In order to achieve this task, the subsequent methodology is followed:

- Implement case-base reasoning to reason about strategies in StarCraft
- Investigate and implement a suitable matching technique for case retrieving in strategical reasoning.
- Implement CBR with fuzzy attributes to reason about tactics in StarCraft
- Find an appropriate representation of game variables in form of fuzzy sets and implement this approach in our CBR system
- Investigate and implement a fuzzy matching method for case retrieving in tactical reasoning
- Implement a process to extract human information from game replays in order to build our case bases that will be used by our system
- Implement and link up to our system the necessary competencies, like resource management and micro management, needed to play a complete match of StarCraft
- Experiment with selected start locations of the map of the game
- Experiment with selected vs random start locations of the map of StarCraft
- Experiment with random start locations of the map of the game
- Analyze and document the experimental results

1.5 Organization

This thesis is divided into five chapters. An introduction to case-based reasoning and the real-time strategy game of StarCraft are presented in chapter 2. First, CBR is presented like a problem solving AI approach that uses past experience to deal with actual problems. Then CBR cycle and its processes of retrieve, reuse, revise
and retain are discussed. Later, the advantages of CBR are stated. Afterwards, the discussion of fuzzy theory and its convergence with CBR are reviewed. Subsequently, the RTS game of StarCraft, its overview and game play are presented. Finally, the Brood War Application Programming Interface (BWAPI), the BWAPI Standard Add-on Library (BWSAL) and the Brood War Terrain Analyzer (BWTA), tools to interact with StarCraft, are shown.

The implementation of case-based reasoning with fuzzy attributes in StarCraft is presented in chapter 3. First, the division of the game in the categories of resource management, strategical management, tactical management and micro management, is presented. Then, the implementation of strategical reasoning using CBR is stated. This involves the strategy selection, the case representation, the process to build the case base, and the explanation of retrieve, reuse and revise processes with the matching method and its implementation. Afterwards, the implementation of tactical reasoning using fuzzy sets and CBR is stated. This involves the description of the space abstraction, the fuzzy case representation and the description of the fuzzy sets that enhance abstract reasoning. It also involves the explanation of the process to build the case base and the retrieve, reuse and revise processes with the fuzzy matching method and its implementation.

Several approaches that have been proposed to deal with the large number of objects and actions involved in RTS games are presented in chapter 4. First, some proposes in real time strategy games based in CBR that focuses in macromanagement are stated. Then, some proposes in real time strategy games based in CBR that focuses in micromanagement, are shown. Finally, other approaches that are not based on CBR and that have been applied to RTS games are reviewed.

Lastly, chapter 5 presents the conclusions of this thesis. It presents the research questions revisited, the contributions section and the direction of future work.
Chapter 2

Background

Case-Base Reasoning is a problem solving AI approach that uses past experience to deal with actual problems. Several authors have applied CBR to different areas of RTS games. Such areas include planning, build order execution, goal formulation, among others. In the present chapter, we present a description of CBR approach. Then, an explanation of the convergence between fuzzy theory and CBR is presented. Also, we give a description of our test bed game and the libraries used to interact with the game. In chapter 4 we describe the related work where we present some research related with the application of CBR technique to different performance tasks in RTS games.

2.1 Case-Based Reasoning

According with Aamodt and Plaza [1994] case-based reasoning is a problem-solving paradigm that is different in many features to other major AI approaches. CBR is able to use the specific knowledge of previous experience of concrete problem situations. In comparison with other approaches that relies only on general knowledge of a problem domain, or making associations along generalize relationships between the descriptions and conclusions of the problems, CBR can deal with specific knowledge and uses past experience in the reasoning process. In the CBR process, a new problem is solved by finding a similar past case, and reusing it in the new problem situation. In CBR a new experience is retained each time a problem has been solved and is saved in a case base, making it available for future problems. This is an important characteristic that makes CBR an approach to incremental and sustained learning. Over the last few years, the CBR field has grown rapidly, as seen by its increased of papers at major conferences, available commercial tools and successful applications of CBR in daily use.

It is common in human reasoning to solve problems reusing past experience, CBR resembles human reasoning in the use of past experience in the form of cases.
2.1.1 Case-based problem solving

A case usually denotes a problem situation in CBR terminology. Past case, previous case, stored case, or retained case, are ways that denote a previously experienced situation, which has been captured and learned in a way that it can be reused in the solving of future problems. Similarly, the description of a new problem to be solved is referred as a new case or unsolved case. Case-based reasoning is a cyclic and integrated process of solving a problem, learning from this experience, solving a new problem, and so on [Aamodt and Plaza, 1994].

Problem solving is not necessarily the finding of a concrete solution to an application problem; it may be any problem put forth by the user. Therefore, the term problem solving is used in a wide sense, coherent with common practices within the area of knowledge-based systems in general. For example, some problem solving situations are to justify or criticize a solution proposed by the user, to interpret a problem situation, to generate a set of possible solutions, or to generate expectations in observable data [Aamodt and Plaza, 1994].

2.1.2 Learning in Case-based Reasoning

The coupling to learning is a very important feature of case-based reasoning. Machine learning community has influence case-based methods that are the base of CBR, consequently case-based reasoning is also regarded as a subfield of machine learning. Case-based reasoning does not only denote a particular reasoning method, it also denotes a machine learning paradigm that enables sustained learning by updating the case base after a problem has been solved. The product of problem solving is the base of the natural learning in case-based reasoning [Aamodt and Plaza, 1994].

Case-based reasoning supports learning from experience. This is possible because it is usually easier to learn by retaining a concrete problem solving experience than to generalize from it. In order to extract relevant knowledge from experience it is required an effective learning that incorporates a well worked out set of methods. Moreover, an effective learning in CBR requires integrating a case into an existing knowledge structure and indexing the case for later matching with similar cases. All case-based reasoning methods have to deal with five central tasks in order to execute the reasoning process. Such tasks are to identify the current problem situation, find a past similar case to the new one, use that case to suggest a solution to the current problem, evaluate the proposed solution, and update the system by learning from this experience. Nevertheless, how this is done, what part of the process that is focused and what type of problems that drives the methods varies considerably [Aamodt and Plaza, 1994].
2.1.3 The CBR cycle

At the highest level of generality, Aamodt and Plaza [1994] describes the general CBR cycle by the following four processes:

- RETRIEVE the most similar case or cases
- REUSE the information and knowledge in that case to solve the problem
- REVISE the proposed solution
- RETAIN the parts of this experience likely to be useful for future problem solving

![Figure 2.1: The CBR Cycle](image)

Figure 2.1 illustrates the CBR cycle composed by four processes: retrieve, reuse, revise and retain. In the CBR cycle a new problem is solved by retrieving one or more experienced cases. Then, the retrieved case is reused in one way or another and it is saved in the solved case. Subsequently, the solved case is revised by the revise process based on reusing a previous case. Finally, the new experience is retained by
incorporating it into the existing knowledge base in form of a new case. Following, according with Aamodt and Plaza [1994], a more detailed description of the processes is mentioned.

**Retrieve.** The retrieve process starts with the description of the problem and ends when a best matching case has been retrieved form the case base. In the retrieve process a new problem is received and its description is used to perform the matching treat over the cases that form the case base. The matching process uses the features of the new problem to search the cases that best match the new problem. The goal of the matching task is to return a set of cases that are sufficiently similar to the new case. As a result of the retrieve process a case or a set of cases that best match the new problem are return. The subtasks of the retrieve process are referred to as Identify Features, Initially Match, Search, and Select, executed in that order.

**Reuse.** The reuse process focuses on two aspects: the differences among the past and current case and what part of a retrieved case can be transferred to the new case. When the differences between the retrieved and the new case are considered non relevant while similarities are relevant, the reuse process becomes a simple classification task and the solution class of the retrieved case is transferred to the new case as its solution class. Nevertheless, other systems have to take into account differences between the retrieved and the new case, and thus the reuse process cannot transfer directly the solution to the new case but requires an adaptation process that takes into account those differences. There are two main ways to reuse past cases: reuse the past case solution, and reuse the past method that constructed the solution.

**Revise.** The revise process consists of two tasks: evaluate the case solution generated by reuse. If successful, learning from the success (case retainment), otherwise repair the case solution using domain-specific knowledge. The evaluation task consists in applying the solution of the solved case in the real environment. Such environments can be simulators of the real world or other software computer programs. This is usually a step outside the CBR system since it involves the application of a suggested solution to the real problem. Depending on the environment, the results from applying the solution may take some time to appear. The repair task involves detecting the errors of the current solution and retrieving or generating explanations of them. Then, explanations of errors are used to modify the solution in order to avoid future errors.

**Retain.** Case retainment is the process that extracts the relevant information from a new problem solving experience and incorporates such knowledge in the case base.
The outcome of the revise process and its adaptation task trigger the learning from success or failure of the proposed solution. The retain process involves selecting which information from the case to retain, in what form to retain it, how to index the case for later retrieval from similar problems, and how to integrate the new case in the memory structure. The retainment process adds information to the case base in form of new cases that have been adapted learning form new experiences.

2.1.4 Advantages of CBR

CBR offers several benefits for intelligent systems. Kolodner [1992] gives a broad list of CBR advantages. Some of these are summarized below:

- When classifications are ill-defined, case-based approach for interpretation can be more precise that a generalization-based method.
- The knowledge found in the case base allows case-based reasoning to propose solutions to problems quickly, avoiding the time necessary to derive those answers from scratch.
- Case-based reasoning offers a medium of evaluating solutions when there is not an available algorithmic method for evaluation.
- The knowledge of previous experiences is particular useful to alert systems to take actions that enable them to avoid repeating mistakes.
- Cases can help systems to focus its reasoning process on important parts of a problem, by describing what features of a problem are the important ones.
- The structure of the features found in the description of the problems, allows CBR to propose solutions in domains that are not understood completely.

2.2 Fuzzy Theory and its Convergence with CBR

According with Wang [1996], Fuzzy systems are knowledge-based or rule-based systems. The core of fuzzy systems is a knowledge base that is composed by a collection of fuzzy IF-THEN rules. Fuzzy rules are linguistic IF-THEN structures that have the general form “IF A THEN B” where A and B are (collections of) propositions containing linguistic variables that are characterized by continuous membership functions. A is called the premise and B is the consequence of the rule. The use of linguistic variables and fuzzy IF-THEN rules takes advantage of the tolerance for imprecision and
uncertainty. Therefore, fuzzy systems resemble the human mind ability to summarize data and focus on relevant information in the decision process.

Fuzzy system consists of four components: fuzzy rule base, fuzzy inference engine, fuzzifier and defuzzifier [Wang, 1996]. The process of reasoning in a fuzzy system can be explained in three categories: input, processing and output stage. The input stage maps input variables to the appropriate membership functions and truth values using the fuzzifier component. The processing stage uses the fuzzy rule base and the fuzzy inference engine components to revise each appropriate rule and generates a result for it, then combines the results of the rules. Finally, the output stage converts the result back into a specific output variable using the defuzzifier component.

Fuzzy Logic (FL) as a decision-making technique has several advantages. Following and according with Sahar et al. [2010], some advantages of FL are shown:

- Rigorous mathematical modeling is not needed when using FL.
- FL resembles human decision-making when imprecise concepts are handled.
- FL is a suitable approach to infer from imprecise information.
- The knowledge representation can be improved using FL to represent the information in terms of linguistic variables.
- Complex and non-linear systems can be modeled using FL.

According with Sahar et al. [2010], two well-known techniques for the implementation of intelligence systems are fuzzy logic and case-based reasoning. There are common concepts shared by FL and CBR, both involve selection, ranking, and aggregation of several alternatives for solving a particular problem. Nonetheless, there are differences between the two approaches and have their own advantages and weaknesses. Fuzzy logic employs the concept of linguistic variables to simplify the process of knowledge representation. A linguistic variable is a variable that can assume linguistic values. Such variables are implemented using fuzzy sets in such a way that reduce the knowledge base of the system, as an entire range of parameter values can be compactly represented by a single fuzzy set. Hence, the system’s knowledge base is defined using linguistic variables as a collection of fuzzy IF-THEN rules. The linguistic interface and the simplified knowledge representation make FL an attractive choice for intelligent system implementations.

However, one problem when using FL for intelligent systems is the difficulty of knowledge acquisition. The domain knowledge for FL-based systems is obtained from domain experts that prepare the rules in the system’s knowledge base. Nevertheless, it is not easy to include the expert’s knowledge in the rules of the system. In comparison, CBR gets around the knowledge acquisition problem by keeping a historical repository
of experience. Intelligent systems based on CBR have a case base that is composed of cases, which are formed by input parameters encountered in the past and the corresponding system output. The decision of any new input parameter configuration is obtained by comparing with all the existing cases and using the most similar case to guide the output decision. The input and output values are saved in the case base for use in future decision making. The knowledge base accuracy grows with experience but a growth in the knowledge base size also means a growth in the system complexity. This means more computational time and memory requirements. As each case is represented by its own set of crisp values for the parameters, an exponential growth in the size of the knowledge base is required to handle all possible cases [Sahar et al., 2010].

The use of a combination of CBR and FL techniques can result in systems that are more efficient and more manageable than the standalone approaches. FL can be used in case representation to provide a characterization of imprecise and uncertain information; in case retrieval, to evaluate partial matches by means of fuzzy matching techniques and in case adaptation to modify the selected case by using the concept of gradual rules [Dubois and Prade, 1992]. Therefore, FL can be used to build CBR systems with a tolerance for imprecision, uncertainty, approximate reasoning, and partial truth, in order to achieve tractability, robustness, low solution cost, and closer resemblance to human decision making [Barletta, 1991].

The use of combined FL/CBR systems goes back to the early 1990s, when CBR systems with fuzzy attributes using fuzzy pattern matching were introduced. One of the earliest hybrid CBR/FL systems is the ARC system [Plaza and de Mantaras, 1991], which uses fuzzy features to represent a prototype class of cases. The ARC system uses a fuzzy pattern matching algorithm to retrieve the most similar class to the input case. The CARS system [Bonissone and Ayub, 1993, 1994] uses fuzzy attributes to represent cases and problems. In the retrieve process, the CARS system calculates the fuzzy similarity measure between attributes using fuzzy algebra. The BOLERO system [Lopez and Plaza, 1991] integrates case-based and rule-based knowledge representation for medical diagnosis. BOLERO system stores past knowledge of solved instance using linguistic terms represented by fuzzy sets.

2.3 StarCraft: Brood War

StarCraft is a real-time strategy (RTS) computer game introduced by Blizzard Entertainment [Blizzard, 2011] in 1998. The game provides functionality to save replays for further review and analysis. Several international competitions are held for StarCraft; South Korea even has a professional league devoted to the game [Wiki, 2011]. The popularity of StarCraft, combined with the ability to save replays, has resulted in large collections of game logs that are available for analysis. StarCraft is praised for
being a benchmark of RTS for its depth, intensity, and balanced races.

The main storyline of the game turns around a war between three galactic species: the protoss, the zerg and the terrans. Protoss are considered a race of humanoid religious warriors, they are technologically advanced and rely on psionic abilities and cybernetics in battle. The zerg are vile insect-like aliens that share a collective consciousness. Finally, the terrans are descendants of human prisoners from Earth and are a young technology species with psionic potential.

We use StarCraft as a test bed to implement and develop our case-based reasoning approach with fuzzy attributes. A variety of libraries allow us to interact with the game in order to execute the processes of our AI module.

2.3.1 Overview

StarCraft was the best selling computer game in 1998 [IGN, 2011a] and in 2009 the Guinness Book of World Records confirmed StarCraft as the best-selling RTS game ever at 9.5 million copies sold [IGN, 2011b]. In November of 1998, Blizzard released an expansion pack called \textit{StarCraft: Brood War}.

The \textit{StarCraft: Brood War} expansion is a StarCraft enhancement that introduces new units for each race. Moreover, it also adds a new campaign for each race, continuing the story begun in StarCraft, plus some new tech advancements, new music tracks and new map features.

The game also includes multiplayer gaming on Blizzard’s own Internet gaming service Battle.net. Players can battle against opponents free of any charge beyond the original purchase of the game and local Internet access fees.

StarCraft has achieved a cult-like status in the computer gaming world. StarCraft game offers a variety of complex strategic possibilities and remains very popular, especially in its online multiplayer form, even years after its original release.

2.3.2 Game Play

StarCraft improved upon other RTS games, by introducing asymmetry between the units and technologies available to its three races (Protoss, Terran, and Zerg). The unit types available to each race define its racial identity. The Protoss can field powerful and expensive warriors and machinery, while the Zerg count on numbers and speed to overwhelm their opponents. The Terrans are the versatile and flexible alternative to both races, with an emphasis on specialization and combined arms. This can make it difficult to create maps that are fair for all races.

The game play of StarCraft is military combat, which contains a lot of rapid unit creation, control, and upgrade, resource management, and attack tactics.
During a match players must collect resources in order to construct a base, building structures and training units, and ultimately leading attacks to conquer opponents. Sending units to explore (scout) is essential to identify enemy regions and get information about the enemy.

### 2.3.3 Replays

StarCraft enables the player to record a game and save it as a *replay*, which can then be viewed with any other copy of StarCraft, displaying the entire course of the game. Actually, there are many websites that host *replays* of players with different skill levels, though pro-level *replays* are relatively rarely released, for reasons of team secrecy.

StarCraft *replays* contain sequential logs of player actions including mouse actions to select or move selected units, and player’s keyboard command to create, build and control units. StarCraft players can use the StarCraft main program to simulate original games by reading log-based *replays*. *Replays* are used by players to review games and extract information about strategies and tactics selection.

### 2.4 BWAPI: An API for Interacting with StarCraft: Brood War

The Brood War Application Programming Interface (BWAPI) [BWAPI, 2011] is a free and open source C++ framework for creating AI modules for StarCraft: Brood War. BWAPI offers methods to gather game information on players and individual units in StarCraft. Moreover, with BWAPI programmers can issue a wide variety of commands or orders to units in real-time game, enabling the possibility for custom AIs that can use different AI approaches to deal with micro and macro management of the game.

BWAPI enforces incomplete information through the fog of war. By default, BWAPI only reveals the visible part of the game to AI modules. Therefore, information of units that have disappeared into the fog of war is not accessible by the AI. Such feature enables programmers to write competitive non-cheating AIs that must plan and operate under partial information conditions. Other default option that delimits the intervention of the human player is disabling the StarCraft General User Interface (GUI). BWAPI uses this option to ensure that the winner of AI vs AI matches is determined exclusively based on the programming and algorithms in the AI module itself, and not by human intervention.

The use of the cheat flags is controlled by BWAPI at the beginning of a match. AI modules have the ability to enable one or more cheat flags that increase the functionality
of BWAPI. In the actual version of BWAPI there exist two cheat flags: complete map information and user input. The first flag, complete map information, makes all units accessible to the AI module; the AI module has information about all units in the game, visible and not visible. The second flag, user input, enables the interaction of the player with the StarCraft GUI, allowing the user to issue order to units along with the AI module. These flags allows programmer to write AI modules that can use the complete map information taking advantage of all units information as well as make hybrid system where human players interact with custom AIs to augment their performance.

BWAPI makes possible to:

- Write competitive AIs for StarCraft: Brood War by issuing orders to individual units.
- Retrieve information on the unit types, upgrades, technologies and weapons used by players.
- Retrieve all information of the game state.
- Analyze replays to extract trends, build orders, and common strategies, in order to perfume micro and macro reasoning.
- Study and research real-time AI algorithms in a robust commercial RTS environment.

We use BWAPI to interact with the game, getting information of the game state and issuing orders to units. Our coded agents that process the CBR approach are linked to BWSAL agents and cooperate with them to achieve our goals using BWAPI and BWTA methods.

2.5 BWSAL: Standard Add-on Library for BWAPI

The BWAPI Standard Add-on Library (BWSAL) [BWSAL, 2011] is a library that develops several add-ons for BWAPI that can be useful for programmers to write a wide variety of AIs. BWSAL offers a series of methods included in a list of managers that makes possible to build up armies and manage bases without having to worry about all the little details in BWAPI.

The actual version of BWSAL includes several managers. These managers are linked and interact to develop some task of the game. The build order manager can accept input orders like training, construction, upgrade and tech development. Build order manager selects the proper manager to execute the orders. These managers are construction manager for building structures, production manager for training of
units and morph manager for morphing units. Worker manager can organize worker type units to gather resources like minerals and gas. Other managers are included in BWSAL library.

We use BWSAL to enhance BWAPI and we use their managers to control resource management and to execute the build or train orders. In this way, our CBR approach delegates order to BWSAL managers to construct or train units, and is BWSAL that selects the appropriate units to execute the order and selects the proper allocations. Some changes were made in the managers of BWSAL in order to improve their performance.

2.6 BWTA: Brood War Terrain Analyzer

Brood War Terrain Analyzer (BWTA) [BWTA, 2011] is an add-on for BWAPI, which analyzes the current StarCraft map and returns the set of expansion locations, regions, and choke points. This library includes methods to read map properties and to access such properties. The analysis of BWTA includes a division of the map into regions. A polygonal area bound every region and the paths to adjacent regions are called chokepoints. With this methods, programmers can identify regions, resource locations, base locations, chokepoints and more properties of the map.

Our CBR agents and BWSAL agents use BWTA methods to analyze the map and extract valuable information that assists the execution of other methods. Our fuzzy approach is based in the division of the map into regions. BWTA analyzes the map of the game and divides the terrain into regions. This information is saved by BWTA in specific files. Our system interacts with BWTA to obtain the information of the terrain. Then, our system uses the information to develop the reason process.

2.7 Summary

An introduction to case-based reasoning approach was presented in this chapter. CBR is a problem solving AI approach that uses past experience to deal with actual problems. An explanation of the processes of the CBR cycle was stated. Retrieve, reuse, revise and retain are the processes that compose the CBR cycle. Later, a brief explanation of fuzzy rule-based system and its convergence with CBR was presented. Subsequently, this chapter presented an introduction to the real-time strategy game StarCraft. An overview of the game was presented, then an introduction to the game play and replays were shown. Finally, this chapter presented an introduction to the Brood War Application Programming Interface (BWAPI), the BWAPI Standard Add-on Library (BWSAL) and the Brood War Terrain Analyzer (BWTA). These tools are add-on libraries that interact with StarCraft and can be useful with a variety of AIs.
Chapter 3

Implementing Fuzzy Sets and CBR approach in StarCraft

StarCraft is a real time strategy game with a large space of actions. The game play consist in gathering resources to build an infrastructure that habilitates the production and training of combat units with the objective of engage in battle and conquer opponents. The possible actions involve the production of every type of units including buildings and military units, choosing every type of upgrades and tech developments, that improves the units abilities and give units new and stronger attacks. Furthermore, actions also include every order that can be issue to units in order to perform the resource gathering, the deployment of units in the map and the organization of units during battle.

Therefore, the combination of all possible actions that a player can take is so large. To deal with the vast space of actions that a match of StarCraft proposes, we divide the game in four categories: resource management, strategical management, tactical management and micro management. The development of all four categories is necessary to be able to play a complete match.

In the next sections we give a wider description of the four game categories. Also, we explain in a deeper way our CBR systems with fuzzy attributes adapted to StarCraft. Moreover, we describe our additions to improve micro management.

3.1 Overview

We divide a match of StarCraft in four categories. The division of the game into categories gives the possibility to deal with every category using different AI approaches. Hence, we can apply CBR to some categories and use other techniques to deal with other categories. We describe each category in the present section. Also, in this section, the selected race for our system, the enemy, the selected map, and the fog of war are presented.
3.1.1 Resource Management

Resource management is the activity of gathering resources and uses them to build bases and armies. Minerals are placed in different positions of the map in the form of crystals and are collected in a resource depot. Gas can be extracted in specific positions where exist a geyser, a refinery must be constructed there to start collecting gas. Worker units are responsible for collecting resources.

In this thesis, resource management is handle by a worker agent included in BWSAL library. This agent calculates the number of workers needed to optimize the resource gathering and assigns the order with a priority level to a build order agent. The selection of the priority level and the number of workers needed was modified to improve their performance. The worker agent of BWSAL offers a method to select the priority level to train worker units. Using such method, a high priority level of 81 was assigned to the construction of workers. This ensures that worker units can be trained with a higher priority than other type of units. The number of workers needed was modified in the code of the worker manager, the optimal worker count sums one worker unit for each mineral and three worker units for each geyser in the active bases of the player.

3.1.2 Strategical Management

Strategy is a plan of action designed to achieve a goal [Heuser, 2010]. Strategy involves high-level decisions about which type of buildings and units to produce. Several factors affect the strategy during game. It is common to select a general strategy at the beginning of the game and modify this plan according with the circumstances, like the discoverer of new enemy technology. Strategy leads the evolution of the game and is the base to develop other areas of the game.

In the present thesis, we propose a Case-Based Reasoning (CBR) agent to deal with strategical reasoning. Our agent interacts with BWSAL agents to issue orders of construction and training of units that leads the strategy of the game. The strategy is caught in its case base, composed by cases that represent the state of the game and the type of units to create.

3.1.3 Tactical Management

Tactics are planned actions to achieve a specific goal [Heuser, 2010]. We refer to tactics like intermediate decisions about how to deploy groups of units across the map and execute attack orders. The movement of military units takes importance in game to discover new regions, to cover terrain and to send units to attack. Tactics leads the armies and can vary along the game according with the objectives.
In the present thesis, we propose applying fuzzy sets and case-base reasoning to deal with tactical reasoning. In this thesis, the combination of fuzzy sets and case-based reasoning is called Fuzzy Case-Based Reasoning (FCBR) and is applied in an agent that performs tactical management. Our agent interacts with our micro management agent to issue orders to units and accomplish the tactical movement of units and their attack. Tactics are saved in a case base, composed by fuzzy cases that represent the abstract space of the game and the actions.

3.1.4 Micro Management

Micro management involves low-level decisions about individual units. Inside a battle, managing the actions of military units is crucial to decrease the damage to our army and increased the damage caused to the opponent. Micro management is an important task to take into account when playing RTS games.

In the present thesis, we develop a hard-coding micro management agent to deal with low-level decisions. Our agent interacts with our tactical management agent to from groups of units, to improve their deployment across the map and to improve the assignment of orders to groups of units.

3.1.5 Protoss Race, built-in AI Enemy and The Map of Python

To decrease the large space of actions presented in StarCraft caused by different features of races we restrict our system to games of the kind Protoss versus Protoss. Our coded AI plays like Protoss and the enemy AI plays like Protoss. The Protoss are a race of humanoids in the StarCraft series. The Protoss are depicted as a physically strong species with access to advanced psionic abilities. The Protoss are considered the most technologically advanced race of the series.

The proposed system presented in this thesis is tested against the built-in AI of StarCraft. Built-in AI is composed by several intelligences with different capabilities and different kind of attacks. For Protoss race, eight tribes compose enemies: Shelak tribe, Akilae tribe, Sargas tribe, Ara tribe, Auriga tribe, Furinax tribe, Velari tribe and Venatir tribe. For each match the game engine selects a random tribe.

The maps of StarCraft vary in dimension, shape of the regions, accessible points and paths. To decrease the large space of actions presented in StarCraft caused by different map features, we restrict our system to the map of Python 1.3. This is a four players map, where exist four possible initial positions for players, assigned randomly at the beginning of the match. Python 1.3 map has been used widely in professional tournaments of StarCraft.
3.1.6 The Fog of War

StarCraft is an RTS game that enforces incomplete information by means of the fog of war. The fog of war refers to the lack of vision and information on areas of the map that have not been explored by friendly units, or areas that were explored but have since been abandoned. In the former case, the area is shown as being totally black. In the latter case, only the terrain and buildings in their last known state (e.g. being built, burning/bleeding) are shown covered by a gray ‘fog’. Updated information on such areas can be gathered by sending a unit into it.

To deal with the fog of war we use the scout manager agent included in BWSAL library. We define an early time at the beginning of the match to send a unit to explore all the possible start initial positions. In Python 1.3 map, there exist four possible positions where players can initiate the game. But, scouting needs to be done frequently to update information about our enemy. Hence, we define a period at a long time in the game where we send several scout units to explore the base locations where an enemy resource depot could be found.

3.2 Strategical Reasoning

Strategical management involves reasoning about what kind of structures to construct and what kind of units to produce. Strategy is presented at all time along the game. Players can have an initial strategy and continue with the same strategy throughout the game, can vary the initial strategy to adapt it to new problems, can change the initial strategy radically or can change the strategy.

To deal with strategical reasoning we propose a Case-Based Reasoning (CBR) agent that handles the information and decisions about what kind of structures and units to produce depending on the state of the game. Our agent executes the CBR cycle including the retrieve process, the reuse process and the revise process. Retain process is not included in our agent. The reason why retainment is not included in our approach is because retaining new information in the case base augments the complexity of the algorithm. Retainment becomes complex due to the process of selection of the information to retain, in what form to retain it, how to index the case and how to integrate the case in the case base.

3.2.1 Strategy Selection

With the help of StarCraft players, we select the strategy for our agent. When playing against built-in AI a good strategy consist in leading massive attacks against the enemy. Our strategy relies on the production of two types of combat units: Zealots and Dragoons.
Zealots are the basic Protoss infantry units. Their high hit points and excellent base ground attack with normal-type damage ensure that they are useful until late in the game, and they are arguably the best breakthrough unit due to their cost and lack of need for micromanagement compared to other units.

Zealots are one of the few small units with a high movement speed (only if the Leg Enhancements are upgraded). Zealots are virtually effective against all sorts of land units especially when supported by Dragoons, High Templar and air units. They are also capable of destroying structures very quickly. Effective counters to back down zealot groups are Reavers and any air unit capable of attacking ground forces. Zealots are also fairly vulnerable to faster units.

As one of the few Protoss ground units with the ability to strike both land and air targets, the Dragoon is an essential element for a well-balanced Protoss force. When used in conjunction with Zealots, Dragoons are very effective scoring hits against enemy forces that are tied up in hand-to-hand combat. If the enemy targets the Dragoons for attack, the Zealots can get “free” hits at close range. Dragoons also provide vital support for Zealots during any aerial attack.

The most important upgrade for the Dragoon is their Singularity Charge. The increased range of the Dragoon’s weaponry that this upgrade grants allows them to concentrate their firepower much more effectively against approaching targets.

Our strategy consists in training groups of Zealots and Dragoons and then sending groups to attack enemy regions. Units are produced continuously so we can attack with a good frequency. The strategy also involves training Observers. Observers are Robotic spies for Protoss forces on the battlefield. Their huge advantage is the ability to stay permanently cloaked. Observers are used for deep-space exploration and are also used as detectors. Observers support our combat units to detect hidden enemy units and to explore enemy regions.

Following, we describe the structure and information of the cases, the processes of the CBR cycle adapted to StarCraft and its implementation.

### 3.2.2 Case Representation

Cases are composed form a description of the problem and its solution. The description of the problem is the representation of the state of the game. The solution part is the building and training actions executed in such state of the game. Hence, we can see our cases composed by two sections, the state of the game and the strategical orders to execute. In figure 3.1 we can see a typical case that composes our case base for strategical reasoning.

We represent the state of the game like a vector of 27 elements. Each element of the vector represents a feature of the game. Our features are each unit type, including buildings and combat units, that Protoss race can produce. The first 13 features denote
combat unit types; the other 14 features denote building unit types. The value of each element represents the number of units, of that type, produced since the start of the game. So, for example in figure 3.1(a), we can see in the first cell of the vector, that five units of type Zealot have been produced since the beginning of the game.

Figure 3.2 illustrates the case representation for strategies in StarCraft. The top left side of the figure shows a partial state of the game composed by a vector with numerical values that represent the number of units constructed since the beginning of the game. In this example, there exist four units of type Zealot and five units of type Dragoon.

The solution part of the case consists in the actions to perform. Actions represent the type of unit that must be constructed or trained. Actions also involve the kind of upgrades and tech development to complete. In figure 3.1(b) we can see that the actions correspond to upgrade the Leg Enhancement, to train a Dragoon and a Zealot, to construct an Observatory and to train a Zealot. We define a length of five actions to perform for every case. This length was empirically chosen. This means that our planning window has a length of five actions. Five actions are executed each cycle of the CBR algorithm. Our agent interacts with BWSAL library to perform the CBR cycle and execute the actions. Like an example, in figure 3.2 the actions that correspond to
the solution of the case were to build a Gateway, Citadel of Adum, Dragoon, Zealot and Robotics Facility.

### 3.2.3 Case Extraction and Case Base

Case-Based Reasoning incorporates human knowledge in form of cases in the case base. RTS games like StarCraft feature numerous objects, incomplete information, micro-actions, and fast paced actions [Cheng and Thawonmas, 2004]. It is in these domains where the human abilities to abstract, generalize, reason, learn, and plan shine [Buro, 2004]. We incorporate human knowledge for strategical reasoning in our case base.

Following the strategy mentioned before, a human experienced StarCraft player was committed to play a match against the built-in AI of the game. The game was saved in a format that enables the replay of the game. Once the replay was saved, we saw again the game and using BWAPI we got the information about the strategy. The features that represent the state of the game and the strategical actions, of training units and constructing buildings, executed in every state of the game were captured and saved in form of cases and added to the case base. Hence, we build our case base from one replay where a human player fights against the built-in AI of StarCraft, following the strategy of massive attacks described in subsection 3.2.1. Our case base for strategy is actually short and is composed by 13 cases. The case base is saved in text file format where each line represents a case. This file is added to the StarCraft folder and is loaded by our agent using a coded method.

In every cycle of the CBR process a new case is retrieved and its solution (build
orders) is applied to the game. The features of the cases represent the total number of units produced until that moment for every type of unit. When the game becomes extensive in time in comparison with the duration of the replay where we got the information, it is common that our agent keeps retrieving the last case in the case base and executes the last actions.

3.2.4 Retrieve Process and Matching Treat

Retrieve is the process to select a case from our case base that is the most similar to the actual state of the game, or problem that we want to solve. Our CBR agent creates a new case with the actual state of the game. Then, the agent compares the new case with every case inside the case base and selects the case that is the most similar to the new case.

To compare the similarity between the new case and the cases of the case base, we use the Euclidean Distance. We have two states: one represents the state of the new case and the other represents the state of the case extracted from the case base. Each state is a vector filled with integers that represents the number of different type of units that have been produced since the beginning of the match. If we symbolizes the two states like \( p = (p_1, p_2, \ldots, p_n) \) and \( q = (q_1, q_2, \ldots, q_n) \) then the Euclidean Distance from \( p \) to \( q \) or form \( q \) to \( p \) is given by equation 3.1.

\[
d(p,q) = d(q,p) = \sqrt{(q_1-p_1)^2 + (q_2-p_2)^2 + \ldots + (q_n-p_n)^2} = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2} \tag{3.1}
\]

This distance denotes the similarity between cases. Cases with shorter Euclidean Distances are more similar than cases with larger Euclidean Distances. We obtain the Euclidean Distance between the new case and every case of the case base and then we select the case with the shortest distance, the most similar one.

3.2.5 Reuse Process

Our CBR agent keeps the description of the problem to solve in a new case structure. Most of the time, the state of the retrieved case is not equal to the state of the new case; we mean that the description of the problem extracted from the case base is not equals to the description of the problem we want to solve. The differences between the new case and the retrieved case are presented because the conditions of the game can vary a lot, and because the retrieved case is the most similar to the new case but not exactly the same.
Our agent resembles a classification task, the differences between cases are considered non relevant while similarities are relevant. Hence, the solution of the retrieved case is transferred to the new case as its solution. Then the new case with its solution is called the solved case.

The difference between the new case and the retrieved one indicates the difference in the number of units that has been produced since the beginning of the game. To construct and train some units in StarCraft first, the player has to construct some other type of units. To deal with this dependency of construction, we use a feature included in BWSAL library that enables the construction of the units that need to be completed before constructing the target unit. Hence, if the solution of the case implies to train a unit that needs a specific building to be trained, the library gives the build order to the proper manager to build the structure and then to train the unit. This feature lets our agent reason without concern about the differences between the actual problems and the retrieved solutions.

3.2.6 Revise Process and its Implementation

The revise process consists in applying the solution of the solved case in the real-time strategy environment; this means the application of the suggested solution to the real problem. The revise process of our CBR agent takes the solution of the solved case and applies this solution to StarCraft. The solution consists of orders for train, construct, upgrade and develop technology of units.

Our CBR agent interacts with BWSAL library to execute the orders of the solution obtained by the retrieve process. The solution consists in a list of five actions or orders. These actions are given in a row to the build order manager of BWSAL library with a decreasing priority level. Build order manager manage the construction, the training and the upgrading orders. When the order involves constructing a *Nexus*, the action is given to the base manager of BWSAL library.

*Nexus* is the resource depot type unit for Protoss race. Resource depots are used to collect minerals and the construction of a resource depot means the expansion of the player in the game. The number of bases with working resources is one of the decisive factors for the game’s outcome. In several of the StarCraft matches, base count is the active goal of the game. Base count and expansion timing is one of the key factors in any StarCraft game.

Our CBR agent verifies the orders given to the build order manager and base manager. When the orders have been executed and the corresponding units and upgrades have been produced, our agent executes the CBR cycle again. A new case with the actual state is compared with the cases in the case base, a case is retrieved, its solution is copied to a solved case and the solution is applied to the game again. The cycle is executed all along the game.
3.2.7 Summary

In the present section, a description of the CBR approach adapted to StarCraft was presented. The strategy that our CBR agent follows relies on massive attacks to enemy regions. This strategy leads the construction and training of units along the game. The strategy is saved in a case base composed by 13 cases, each case is composed by a description of the state of the game and the orders executed in such state. Human knowledge is represented in the cases of the case base. The retrieve process, the reuse process and the revise process are executed by our CBR agent which interacts with BWSAL library to produce, train and upgrade units while playing a game. The structure of CBR approach lets our agent to incorporate human knowledge to reason about strategical decisions. Moreover, CBR approach enhances our agent to deal with imperfect information presented in the fog of war of the game. With CBR, our agent is able to accomplish strategical management and, interacting with other agents, to play a match of StarCraft against built-in AI enemies.

3.3 Tactical Reasoning

Tactical management involves the decisions about how to deploy groups of units across the map. Tactical management is reserved to military units. Tactics are the art of organizing an army, the techniques for using weapons and military units in combination for engaging and defeating an enemy in battle. Tactics can vary along the game and are influenced by the strategy and the information about the enemy.

To deal with tactical management we propose the combination of fuzzy sets and case-based reasoning into an agent that handles the information and decisions to deploy units across the map. In this thesis, the combination of fuzzy sets and case-based reasoning is called Fuzzy Case-Based Reasoning (FCBR). The union of fuzzy theory and case-based reasoning simplifies the process of knowledge representation and enables the knowledge acquisition practice using a case base [Sahar et al., 2010]. The use of linguistic variables to represent the features of the cases reduces the large space of actions and objects that an RTS game proposes. This is possible because an entire range of parameter values can be compactly represented by a single fuzzy set. Moreover, keeping a historical repository of experience, the case-based reasoning approach can acquire the knowledge that is used in the reasoning process. Therefore, using a FCBR approach, we can deal with the vast space of actions presented in an RTS game and incorporate human knowledge in the reasoning process. Our FCBR agent is based on CBR approach and uses a fuzzy representation of the cases. This technique allows dealing with abstract and incomplete information. Furthermore, an abstract representation of the state of the game is more like human thinking. Human players tend to think in a form that is fluid or approximate rather than fixed and exact, fuzzy theory...
deals with such kind of reasoning.

Our agent executes the CBR cycle including the retrieve process, the reuse process and the revise process. Retain process is not included in our agent. The reason why retainment is not included in our approach is because retaining new information in the case base augments the complexity of the algorithm. Retainment becomes complex due to the process of selection of the information to retain, in what form to retain it, how to index the new case and how to integrate the new case in the case base.

Following we describe the space abstraction, the structure and information of the fuzzy cases, the processes of the CBR cycle adapted to StarCraft and its implementation.

3.3.1 Space Abstraction

To accomplish tactical reasoning and to deal with the large amount of actions and information that an RTS game like StarCraft propose we divide the state of the game into regions. Using BWTA library, an analysis of the map can be done. This analysis computes the regions, chokepoints and base locations. A region is a partition of the map with a polygon boundary, and is connected to other regions via chokepoints. A chokepoint is the line that divides two regions; a chokepoint connects exactly two regions. A base location is a position on the map where it makes sense to place a base.

Our FCBR agent uses regions to perform tactical reasoning. Instead of having a game state that includes information about the entire map, we have a game state for every region of the map. Therefore, each region has its own fuzzy state description. Individually regions are independent of other regions. This means that we have a state description of the game for each single region of the map. Hence, a typical case represents the state of some region and the actions executed in such state for such region. With this division, we can handle the information of the map executing the CBR cycle in every region. Thus, the retrieve process, the reuse process and the revise process can be executed individually for every region. Our FCBR agent processes the CBR cycle in regions where exists friendly combat units.

In figure 3.3 a typical division of a StarCraft map into regions process by BWTA is presented. This example shows the Lost temple map divided into regions. Chokepoints are the connections between adjacent regions. This information is used by our FCBR agent to process tactical management.

3.3.2 Fuzzy Case Representation

Analogously to the strategical representation of cases, tactical cases are composed from a description of the problem and its solution. The description of the problem is the representation of the state of some region. The solution part involves the tactical actions
executed over the units of such region to move them across the map. Our FCBR agent handles three tactical actions: move, attack-move and attack-unit. StarCraft game has a large list of orders that can be issue to units. Orders like follow, attack, move, patrol, stop and hold position, are a typical set of actions that a player can use. But moving units and attacking enemy are the principal ones for our agent. These actions represent the solution part of cases in tactical reasoning. In figure 3.4 we can see a typical case that composes our case base for tactical reasoning.

We represent the state of a region like a vector of 31 elements. Each element of the vector represents a feature of the region. Each feature has a range of values characterized by linguistic variables. For example, in figure 3.4(a) the first element of the vector represents the feature of “area”, the area is characterized by a set of linguistic variables like “small”, “medium”, “big”, etc., that represent the geographical size of the region. The first two elements of the vector denote the geographical features of area and chokepoints. The area symbolizes the size of the region, and the chokepoints symbolize the number of connections with other regions. The next three elements represent the military features of military presence, combat intensity and lost units. The military presence refers to terrain possession, the combat intensity denotes the
Figure 3.4: Case representation for tactical reasoning

(a) Problem representation

<table>
<thead>
<tr>
<th>Geographic Features</th>
<th>Military Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area</td>
<td>Corsair</td>
</tr>
<tr>
<td>Distance</td>
<td>Dark Templar</td>
</tr>
<tr>
<td>Military Presence</td>
<td>Zealot</td>
</tr>
<tr>
<td>Combat Intensity</td>
<td>Dragon</td>
</tr>
<tr>
<td>Lost Units</td>
<td>High Templar</td>
</tr>
<tr>
<td>Corsair</td>
<td>Archon</td>
</tr>
<tr>
<td>Dark Templar</td>
<td>Shuttle</td>
</tr>
<tr>
<td>Zealot</td>
<td>Scout</td>
</tr>
<tr>
<td>Dragon</td>
<td>Arthor</td>
</tr>
<tr>
<td>High Templar</td>
<td>Carrier</td>
</tr>
<tr>
<td>Archon</td>
<td>Reaver</td>
</tr>
<tr>
<td>Shuttle</td>
<td>Observer</td>
</tr>
</tbody>
</table>

Fuzzy Numbers for
Friend combat unit types

none less none none none none none none none none none

(b) Problem representation (continuation)

Fuzzy Numbers for
Enemy combat unit types

(c) Solution representation

<table>
<thead>
<tr>
<th>AttackUnit</th>
<th>Protoss Zealot</th>
<th>Protoss Dark Templar</th>
<th>AttackUnit</th>
<th>Protoss Dragoon</th>
<th>Protoss Photon Cannon</th>
</tr>
</thead>
</table>

Actions
strength of combat between rivals and the lost units feature represents the number of killed friend units in the region. The next 13 features represent the number of friend combat units in the region and the last 13 elements of the vector denote the number of enemy combat units in the region. The numeric state feature is used in the last 26 elements of the vector to characterize the number of units in the region for every type of combat unit.

Every feature of the region (every element of the vector) is characterized by a set of linguistic variables. Linguistic variables represent crisp information in a form and precision appropriate for the problem and are central to fuzzy theory.

Figure 3.5 illustrates the case representation for tactics in StarCraft. The top left side of the figure shows a partial state of the game composed by a vector with linguistic variables that represent the fuzzy features of the state of the region. In this example, the region is big and it has one path that communicates with other regions. Moreover, there are a lot units of type Zealot.

The solution part of the case is composed by the tactical actions to perform. Tactical actions are issued to friend combat units in the region. Such actions involve move or attack-move other regions, attack enemy units in the same region or attack enemy units that are in other regions. Actions are composed by the order name, the type of unit that executes the action, the type of target unit and the description of the target region. The number of actions in a case is variable. The number of actions varies according to the change of the state of the region where we extract the information. The actions are executed each cycle of the CBR algorithm, if the proper units are present and orders can be issued to them. Our agent interacts with BWSAL library to perform the CBR cycle.
Following we describe the membership functions that we defined for our linguistic variables.

### 3.3.3 Implementing Fuzzy Sets in StarCraft

Linguistic variables associate a linguistic condition with a crisp variable. A crisp variable is the kind of variable that has an absolute value. A linguistic variable, on the other hand, has a proportional nature: they are represented by fractional values in the range of 0 to 1. Linguistic variables are represented by fuzzy sets. According with Zadeh [1965] a fuzzy set is a class of objects with a continuum of grades of membership. Such a set is characterized by a membership function that assigns to each object a grade of membership ranging between the real unit interval $[0, 1]$.

We used trapezoidal membership functions to represent fuzzy sets. These membership functions are denoted by functions with trapezoidal shape. Subsequent we describe the membership functions that define our fuzzy sets that characterize the state of the regions in our cases.

For the feature of area we have small, medium, big, large and immense membership functions. The horizontal axis represents the area of the region and the vertical axis represents the membership degree. Figure 3.6 shows the fuzzy sets for area.

For chokepoints feature, we have one, some and several membership functions. The horizontal axis represents the number of chokepoints of the region and the vertical axis represents the membership degree. Figure 3.7 shows the fuzzy sets for chokepoints.

For military presence feature, we have enemy, hostile and friendly membership functions. The horizontal axis represents the rate between friend units in the region and all units in the region (including friend and enemy units). A rate of zero means that there are only enemy units in the region, a rate of one means that there are only friend units in the region. The vertical axis represents the membership degree. If there are no units in the region then the military presence feature is classified like unknown. Figure 3.8 shows the fuzzy sets for military presence.

For combat intensity feature, we have low, moderate and high membership functions. The horizontal axis represent the rate between friend military units that are engage in battle and total friend military units. A rate close to zero means that a little portion of the army in the region is battling and a rate close to one means that a big portion of the army in the region is battling. The vertical axis represents the membership degree. If there are no military units battling then combat intensity is classified like none. If there are no military units in the region then combat intensity is classified like not soldiers. Figure 3.9 shows the fuzzy sets for combat intensity.

For lost units feature, we have less, few, some, many and a lot membership functions. The horizontal axis represents the number of friend units lost in battle in the region. The vertical axis represents the membership degree. If no units are lost then
Figure 3.6: Membership functions for *area*

Figure 3.7: Membership functions for *chokepoints*
Figure 3.8: Membership functions for military presence

Figure 3.9: Membership functions for combat intensity
lost units is classified like *none*. Figure 3.10 shows the fuzzy sets for lost units.

For *numeric state* we have *less, few, some, many*, and *a lot* membership functions. The horizontal axis represents the number of units, for every type of unit, in the region. The vertical axis represents the membership degree. If there are no units for one type of unit then numeric state is classified like *none*. In a case we have 26 numeric state features, 13 to represent every type of military friend units and 13 to represent every type of military enemy units. Figure 3.11 shows the fuzzy sets for numeric state.

### 3.3.4 Case Extraction and Case Base

Analogous to strategical reasoning, our FCBR agent incorporates human knowledge for tactical reasoning in its case base. The information to build the tactical case base was acquired using BWAPI from the *replay* played by a StarCraft player mentioned in subsection 3.2.3. The features that represent the state of the game and the tactical actions to deploy groups of units across the map were captured and saved in the form of cases and added to the case base. Hence, we build our case base from one *replay* where a human player fights against the built-in AI of StarCraft, following the strategy of massive attacks described in subsection 3.2.1. Our case base for tactical management is composed by 147 cases. The case base is saved in text file format where each line represents a case. This file is added to the StarCraft folder and is loaded by our agent using a coded method. We have two case bases, one for strategical management, described in section 3.2, and one for tactical management.

According with subsection 3.3.1 every region of the map executes the CBR cycle independently. During a CBR cycle, the solution of the case is executed if there are available units in the region and such units satisfy the description of the actions. Since regions can have similar states along the game, our FCBR agent retrieves cases from the case base during all the match.

### 3.3.5 Retrieve Process and the Fuzzy Matching Method

Retrieve is the process to select a case from our case base that is the most similar to the actual state of the region, or problem that we want to solve. Every region of the map executes the retrieve process individually. Our FCBR agent creates a new case with the actual state of the region. Then, the agent compares the new case with every case inside the case base (tactical case base) and selects the case that is the most similar to the new case. Our case base has cases of all regions. Hence, all regions retrieve cases from the same tactical case base. Our FCBR agent handles the CBR cycle for all regions. We empirically defined a constant of time to execute the CBR in every region.

We built a library with several methods to fuzzify the variables that represent the
Figure 3.10: Membership functions for *lost units*

Figure 3.11: Membership functions for *numeric state*
state of the regions. These methods receive the numeric variables and using the fuzzy sets described in subsection 3.2.2, the methods fuzzify the values and return the proper fuzzy variables (linguistic variables). Then, the state of the new case is characterized with fuzzy variables.

To compare the similarity between the new case and the cases of the case base (tactical case base), we use the intersection area between membership functions.

The overlapping area plays an important role in the computation of the similarity measure between cases. It is clear that as the overlapping area gets bigger the similarity between the cases increases. In other words, when the two cases have similar membership functions, that indicates complete similarity between the two cases. When the overlapping area is zero, then there is no similarity between the two attributes. The process of finding the overlapping area is a simple procedure in which all the intersecting points are computed, and connected again thus creating the new area [Dvir et al., 1999]. This idea is depicted in figure 3.12.

![Intersection area between two membership functions](image)

Figure 3.12: Intersection area between two membership functions

According with Dvir et al. [1999], we assume that $A_1$ is the area associated with one membership function and $A_2$ is the area associated with the second membership function. Also, we assume that the overlapping area is denoted as $OA$. Then the similarity between the two attributes is defined as:

$$S_{IA} = \min\left(\frac{OA}{A_1}, \frac{OA}{A_2}\right) \quad (3.2)$$

Using the intersection area we obtain the overall similarity between two cases using a weighted Euclidean Distance. We have two fuzzy states; one represent the state of the new case and the other represents the state of the case extracted form the case base. Each state is a vector filled with linguistic variables that represent a fuzzy set and a membership function. If we symbolizes the two states like $\mathbf{p} = (p_1, p_2, \ldots, p_n)$ and $\mathbf{q} = (q_1, q_2, \ldots, q_n)$ then the weighted Euclidean Distance from $\mathbf{p}$ to $\mathbf{q}$ or form $\mathbf{q}$ to $\mathbf{p}$ is given by equation 3.3.
where \( w_i \) is a weight attached to the \( i \)-th variable and \( S_{IA}(p_i, q_i) \) is the intersection area between the membership functions of the \( i \)-th components of the states of the cases. This distance denotes the similarity between cases. Cases with shorter weighted Euclidean Distances are more similar than cases with larger weighted Euclidean Distances. We obtain the weighted Euclidean Distance between the new case and every case of the case base and then we select the case with the shortest distance, the most similar one.

### 3.3.6 Reuse Process

Similar to our CBR agent seen in subsection 3.2.5, our FCBR agent resembles a classification task, the differences between cases are considered non relevant while similarities are relevant. Hence, the solution of the retrieved case is transferred to the new case as its solution. Then the new case with its solution is called the solved case.

The difference between cases indicates that the actual state of the region is not exactly the same as the state of the region represented by the retrieved case. In spite of the differences, similar cases represent similar states of the region and the solution of the retrieved case can be applied to solve the new case.

### 3.3.7 Revise Process and its Implementation

The revise process involves applying the solution of the solved case in the RTS environment; this means the application of the suggested solution to the real problem. Our FCBR agent takes the solution of the solved case and applies this solution to StarCraft. The solution is composed by attack-move, attack and move orders issued to combat units.

Our FCBR agent interacts with our Micro Management (MM) agent to execute the orders of the solved case. The micro management agent keeps a list with the actual groups of units that our system has during a match. During the revise process the FCBR agent communicates with the micro management agent to see the groups of units that are available. According with the description of the order and the unit types involved in the order, a search over the groups is performed and if a group satisfies the description then the action is issued to the group. The orders given by our FCBR agent are objectives to our MM agent. Once the orders are issued to groups, MM agent leads the execution of the order. The description and functionality of MM agent is presented in section 3.4.
To execute the revise process we divide the match time and use some limits to restrict the application of the solutions proposed by solved cases. What we do is to split the match in three phases based on time, the first one is an early time of game, the second one is a middle time of game and the third one is a large time of game. Every case of the case base has a time stamp. The time stamp corresponds to the time of the game when the actions of the case were executed. The time stamp for every case is obtained during the creation of the case base. Therefore, each case has a time stamp that indicates the time of the game when the actions of the case were performed. We use the time stamp of cases and the similarity measure of cases to restrict the application of them. In the first phase of game the restrictions are stretched. In the second phase, the restrictions are relaxer than the first phase. Then, in the third phase of the game the restriction of time stamp disappear an only a relax restriction of similarity between cases is held. Restrictions let us leading tactics in such a way that in early times of the game the tactics can be executed more similar to the strategy saved in the replay were we got the human knowledge. And, in larger times of the game tactics can have more variety.

3.3.8 Summary

In the present section, a description of the FCBR approach adapted to StarCraft was presented. The tactics executed by our FCBR agent are based on human knowledge that is presented in the tactical case base built from a replay. The case base for tactical reasoning is composed by 147 cases, each case is formed by a fuzzy description of the state of a map’s region and a list of tactical actions to execute over the friend combat units in the region. The retrieve process, the reuse process and the revise process are executed by our FCBR agent, which interacts with our micro management agent to execute the orders and achieve the proper deployment of units across the map. The structure of fuzzy case-based reasoning approach lets our agent to incorporate human knowledge to reason about tactical decisions. Moreover, FCBR approach enhances our agent to deal with imperfect information presented in the fog of war of the game and enables our agent to perform spatial and temporal reasoning using a fuzzy representation of the game. With FCBR, our agent is able to accomplish tactical management and, interacting with other agents, to play a match of StarCraft against built-in AI enemies.

3.4 Micro Management Improvement

In RTS games like StarCraft, the space of actions that a match proposes is large. As we mentioned before, to be able to play a complete match of StarCraft, we divide the game in four categories: resource management, strategical management, tactical
management and micro management. Micro management is concerned with low-level decision about individual units. Micro management describes minor, detailed gameplay elements that must be manually addressed by the player. In StarCraft micro management is an important task that must be addressed to improve gameplay and defeat the opponents.

To accomplish micro organization of units we created a Micro Management (MM) agent that handles the supervision of units at micro level. Our agent adds methods for improve certain behaviors of our units in order to perform well in battle. Micro management agent interacts with our tactical management agent (FCBR agent). The MM agent has a list with the available friend combat groups of units and a description of them. Such description contains the name of the actual order that the group must accomplish. Groups without orders have an idle state. When performing the CBR cycle, the tactical management agent, access the list of groups from MM agent, if there are available groups that satisfies the description of the orders, then the orders are issued to the groups. Orders issued by tactical management agent are objectives for MM agent. MM agent interacts with BWTA and BWAPI libraries to achieve the orders of the groups of military units. Following we describe the behaviors added by MM agent to improve micro management in gameplay.

3.4.1 Chokepoint Defense

During the first few minutes of a game, the biggest threat from Protoss players comes in the form of Zealots. It is usual that enemy’s tactics at this stage consists in build Zealots and overwhelm the opponent. This is called The Zealot Rush. In most Protoss on Protoss encounters, the first few minutes of the game will usually consist of Zealot Rushing. To deal with this difficult and properly defend our base against enemy rushers, our micro management agent adds the chokepoint defense behavior.

This behavior consists in sending our combat units to defend the chokepoints of the region where we start the match. Sending units to protect the chokepoints is a good practice to guard the entrances to our base. When friend units protect the chokepoints, enemy rushers are weaker. This is possible because enemy units have not free paths to achieve its target and are delayed in time. Moreover, chokepoint defense increase the damage caused to our opponent. If our combat units cover the area of the chokepoint then they have a wider range of attack than the opponent units that enter the chokepoint.

Chokepoint defense behavior is applied only during the first few minutes of the game where enemy rushers are a common attack. Then, the MM agent drops this behavior and no more units are committed to guard chokepoints.
3.4.2 Target Path

The organization of groups of units to attack enemy bases is essential to increase the damage to enemy units and structures and succeed in battle. Massive attacks consist in building a large quantity of soldiers and send them to attack enemy regions. Groups of units are more effective attacking when units are closer. To achieve better results during attacks, our micro management agent adds the target path behavior.

Target path behavior consists in leading the path following by our armies in order to achieve coordination between units of the same group and between groups of different type of units to accomplish a better deployment of groups across the map and increase the damage inflicted to enemy armies and enemy bases. Every type of unit in StarCraft has special properties. One of these properties is velocity. Ground and air units have different velocity according to the type of unit. Moreover, is possible to upgrade the velocity for certain type of units. The difference in velocity causes that faster units achieve its targets in shorter time. The delay in movement between types of units decreases the effectiveness of attacks.

What target path does is to get the target position of groups and calculates a “path” to follow. The path consists in a series of positions that groups must reach before reach its target position. Target path leads the movement of groups according with the proximity of the center of the group to its actual target position and according to the closeness between individual units. We empirically defined the optimal distance of the center of the group to its actual target position. Also, we empirically defined the optimal closeness of units of the same group based on a dispersion measurement.

In addition, target path behavior handles the coordination between groups of different type of units. Groups with the same order and same target position, have the same “path” to follow. Target path coordinate groups of different type of units with the same order to be able to reach the final target position at same time increasing the damage caused to the opponent. Target path behavior is presented at all time during a match.

3.4.3 Attack Coordination

To improve coordination of attack orders, micro management agent can issue attack orders to groups that are in idle state. Using FCBR approach, our tactical management agent issues orders to groups of units. These orders are handle by our micro management agent. When micro management agent detects that an attack order than has been issued to a group has as target a previously identified enemy region, MM agent searches other groups with idle state and assign them the same order.

With this behavior micro management agent can build a better offensive while attacking enemy regions. Groups of units that are standing idle are used to support
groups that have attack orders to go into battle and attack enemy terrain.

3.4.4 Summary

In this section we presented a description of the improvements coded in micro management agent adapted to StarCraft to accomplish micro organization of units. Micro management is an important task in gameplay and is necessary to play an entire match in real-time strategy games like StarCraft. The improvements addressed by our micro management agent consist in three behaviors: chokepoint defense, target path and attack coordination. Such behaviors enhance the coordination of groups of units performing better results in battle. With such behaviors, our micro management agent is able to accomplish micro organization and interacts with other agents to play a match of StarCraft against built-in AI enemies.

3.5 Results

Fuzzy case-based reasoning is a novel approach founded upon case-based reasoning with the enhancement of fuzzy theory and fuzzy sets. We implemented CBR and FCBR approach to accomplish strategical and tactical management inside a real-time strategy environment. FCBR lets reasoning with large abstract actions spaces that enforce incomplete information. Moreover, FCBR lets incorporate human knowledge in the form of cases. Such features result in a system that take advantage of human knowledge to reason about new problems based on the solutions of similar past problems.

To deal with the challenges that a real-time strategy environment as StarCraft proposes, we divided the problem of play a complete match in four categories: resource management, strategical management, tactical management and micro management. We focused on strategical and tactical management building a case-based reasoning agent to accomplish strategical reasoning and building a fuzzy case-based reasoning agent to accomplish tactical reasoning. The development of these agents and the addition of other ones resulted in a bot (a software program that imitates the behavior of a human) that plays against built-in AI enemies in StarCraft. The resulting bot is called FCBR-Bot.

We developed our experiments in the map of Python 1.3. This is a four players map and has four possible start positions for players. We performed our experiments in a Protoss vs Protoss version of the game where our bot and its opponent engaged in battle.

Following, we describe the experiments that demonstrate the application of our bot in StarCraft. We divided the experiments in three parts according with the start location of our bot and the enemy.
3.5.1 Selected FCBR AI Initial Position vs Selected Built-in AI Initial Position

In this experiment we select the initial position for our bot and the initial position for our enemy. The knowledge used for our agents is contained in the case bases. This knowledge was obtained from a replay played by a human player following the strategy described in subsection 3.2.1. In this replay the enemy starts in a certain position of the map and the player start in another certain position of the map. To resemble replay conditions, we delimit our system to select the start position for our bot to be the same start position of the player (human) in the replay. Also, we delimit our system to select the start position of our enemy to be the same start position of the enemy in the replay, where we extracted the information.

With these conditions we performed 100 experiments, it means 100 matches, where our bot played against the built-in AI of the game. The results of the games are shown in figure 3.13.

Analysis

According with figure 3.13(a) our approach implemented in StarCraft was victorious 60% of the games. Our bot loses 28% of the games and obtained 12% tie games. The application of case-based reasoning for strategical management and fuzzy case-based reasoning for tactical management in addition with other improvements is able to reason in StarCraft and to defeat built-in AI opponents with a rate for more than half of the played games. Taking into account that our bot uses only one strategy saved into the knowledge of the case bases and that the built-in AI of StarCraft consists in random strategies according with the enemy selected by the game engine, we can state that our system can successfully adapt to different kind of strategies followed by the opponent. Furthermore, resembling the conditions of the start locations of the replay where we extracted the information, our system is able to successfully reason with incomplete information and a large space of actions. The evidence relies on the fact that under the same conditions, the enemy only wins almost half of the times than our system. This shows that our system performs better than the built-in AI of StarCraft.

Figure 3.13(b) shows the game scores obtained by our system and its opponents. In this figure, the scores of all games are presented. Blue area corresponds to the scores obtained by our system while red area corresponds to the scores obtained by the opponent. We can see that blue area has more presence than red area because the number of victories achieved by our system is more than the number of enemy victories. Players with higher scores are commonly victorious. The scores are calculated bases on the internal scores assigned by StarCraft to the construction and destruction of units. Different types of units have different build score and destroy score. Player's
Figure 3.13: Results of FCBR-Bot in custom start locations
(a) Win scores

(b) Lose scores

(c) Tie scores

Figure 3.14: Scores of FCBR-Bot in custom start locations
total score is composed by the sum of the scores of build units and destroy units, build structures and destroy structures, harvest minerals and gas produced. A larger number of victories demonstrate the better performance of the player. Hence, our FCBR-Bot performed better than the opponent. The strategical and tactical reasoning approach in our system was able to reason in a nondeterministic environment and defeat its opponent most of the times.

The scores acquired by our system in win, lose and tie games are shown in figure 3.14. Win game scores are the scores obtained by our bot in games where it was victorious, lose game scores are the scores obtained by our bot in games where it was defeated by its opponent and tie game score are the scores where no player achieved victory. The tie games represent the matches were there was not a winner not a loser player. We consider tie games when the game engine of StarCraft does not design a winner. In order to designate a winner player, the game engine of StarCraft checks if all structures of one player are destroyed.

Comparing figure 3.14(a) and figure 3.14(b), it can be observed that our approach performed better than the built-in AI in most of the games because FCBR-Bot obtained higher scores winning in 60 of 100 games and losing in only 28 games with lower scores. Figure 3.14(c) shows the scores for tie games. In certain games the structures of the players are not destroyed at all. Even when one player has a higher or lower score if not all structures from one player are destroyed, the game engine of StarCraft does not designate a winner and the game is considered like a tie.

### 3.5.2 Selected FCBR AI Initial Position vs Random Built-in AI Initial Position

In this experiment we select the start position of our bot while the start position of the enemy is randomly assigned. Like previous experiment, to resemble the conditions of the replay where we extracted the human knowledge, we coded our system to select the start position of our bot to be the same start position of the player (human) in the replay. But now, in this experiment, the opponent can start in any other start location. It means that the built-in AI has a random start location. With these conditions we demonstrate that our system can process abstract information of the enemy even when the enemy start locations is random. Our FCBR approach makes an abstract reasoning due to the division of the map into regions and the fuzzy representation of the state of such regions. Then, FCBR approach can adapt to the varying conditions presented when enemy start in different locations. Such enhancement lets reasoning in large space of actions enforced with incomplete information, such as StarCraft.

With these conditions we performed 100 experiments, it means 100 matches, where our bot played against built-in AI enemies. The results of the games are shown in figure 3.15.
Figure 3.15: Results of FCBR-Bot in custom start location vs random start location
Figure 3.16: Scores of FCBR-Bot in custom start location vs random start location
Analysis

In agreement with figure 3.15(a), our approach implemented in StarCraft was victorious 58% of the times. Our bot loses 25% of the games and obtained 17% tie games. The application of case-based reasoning for strategical management and fuzzy case-based reasoning for tactical management in addition with other improvements is able to reason in StarCraft and defeat built-in AI opponents with a rate for more than half of the played games.

Taking into account that our bot uses only one strategy saved into the knowledge of the case bases and that the built-in AI of StarCraft consists in random strategies according with the enemy selected by the game engine, we can state that our system can successfully adapt to different kind of strategies followed by the opponent. Based in the fact that in this experiment the opponent can start in any region of the map (random) and the start location of our bot resembles the start location of the replay where we extracted the information, we can state that our system is able to successfully reason with incomplete information, a large space of actions and, using a division into regions and a fuzzy region state representation, our system can successfully adapt to different enemy start locations, the evidence relies on the fact that under such conditions, the enemy only wins almost half of the times than our system. This shows that our system performs better than the built-in AI of StarCraft.

Figure 3.15(b) shows the game scores obtained by our system and its opponents. In this figure, the scores of all games are presented. Blue area corresponds to the scores obtained by our system while red area corresponds to the scores obtained by the opponent. We can see that blue area has more presence than red area because the number of victories achieved by our system is more than the number of enemy victories. Players with higher scores are commonly victorious. The scores are calculated bases on the internal scores assigned by StarCraft to the construction and destruction of units. Different types of units have different build score and destroy score. Player’s total score is composed by the sum of the scores of build units and destroy units, build structures and destroy structures, harvest minerals and gas produced. A larger number of victories demonstrate the better performance of the player. Hence, our FCBR-Bot performed better than the opponent. The strategical and tactical reasoning process in our system was able to reason in a nondeterministic environment and to defeat its opponent most of the times, even when the opponent has a random start location.

The scores acquired by our system in win, lose and tie games are shown in figure 3.16. Win game scores are the scores obtained by our bot in games where it was victorious. Lose game scores are the scores obtained by our bot in games where it was defeated by the opponent. Tie game scores are the scores where no player achieved the victory. Tie games represent the matches where there was not a winner not a loser player. We consider tie games when the game engine of StarCraft does not design a
winner. In order to designate a winner player, the game engine of StarCraft checks if all structures of one player are destroyed.

Comparing figure 3.16(a) and figure 3.16(b), it can be observed that our approach performed better than the built-in AI in most of the games because FCBR-Bot obtained higher scores winning in 58 of 100 games and losing in only 25 games with lower scores. Figure 3.16(c) shows the scores for tie games. In certain games the structures of the players are not destroyed at all. Even when one player has a higher or lower score if not all structures from one player are destroyed, the game engine of StarCraft does not designate a winner and the game is considered like a tie.

3.5.3 Random FCBR AI Initial Position vs Random Built-in AI Initial Position

In this experiment our bot and its opponent can start in a random location of the map. We relax the start location feature and now both players can start in random positions of the map. With these conditions we demonstrate that our system can process abstract information even when start locations of both, our system itself and the built-in AI, are randomly assigned. Our FCBR approach makes an abstract reasoning due to the division of the map into regions and the fuzzy representation of the state of such regions. Then, FCBR approach can adapts to the varying conditions presented when it and its enemy have a random start location. Such enhancement lets reasoning in large space of actions enforced with incomplete information, such as StarCraft.

With these conditions we performed 100 experiments, it means 100 matches, where our bot played against built-in AI enemies. The results of the games are shown in figure 3.17.

Analysis

In accordance with figure 3.17(a) our approach implemented in StarCraft was victorious 61% of the times. Our bot loses 23% of the games and obtained 16% tie games. The application of case-based reasoning for strategical management and fuzzy case-based reasoning for tactical management in addition with other improvements is able to reason in StarCraft and defeat built-in AI opponents with a rate for more than half of the played games.

Taking into account that our bot uses only one strategy saved into the knowledge of the case bases and that the built-in AI of StarCraft consists in random strategies according with the enemy selected by the game engine, we can state that our system can successfully adapt to different kind of strategies followed by the opponent. Based in the fact that in this experiment our system and the opponent can start in any region of the map (random), we can state that our system is able to successfully reason with
Figure 3.17: Results of FCBR-Bot in random start location vs random start location
Figure 3.18: Scores of $FCBR$-$Bot$ in random start location vs random start location
incomplete information, a large space of actions and, using a division into regions and a fuzzy region state representation, our system can successfully adapt to different start locations for both, our bot itself and the enemy, the evidence relies on the fact that under such conditions, the enemy only wins almost half of the times than our system. This demonstrates that our system performs better than the built-in AI of StarCraft.

Figure 3.17(b) shows the game scores obtained by our system and its opponents. In this figure the scores of all games are presented. Blue area corresponds to the scores obtained by our system while red area corresponds to the scores obtained by the opponent. We can see that blue area has more presence than red area because the number of victories achieved by our system is more than the number of enemy victories. The scores are calculated bases on the internal scores assigned by StarCraft to the construction and destruction of units. Different types of units have different build score and destroy score. Player’s total score is composed by the sum of the scores of build units and destroy units, build structures and destroy structures, harvest minerals and gas produced. A larger number of victories demonstrate the better performance of the player. Hence, our FCBR-Bot performed better than the opponent. The strategical and tactical reasoning code in our system was able to reason in a nondeterministic environment and defeat its opponent most of the times, even when both players have random start locations in the map.

The scores acquired by our system in win, lose and tie games are shown in figure 3.18. The scores obtained by our bot in games where it was victorious are called win scores. The scores obtained by our bot in games where it was defeated by the opponent are the lose scores. The scores where no player achieved the victory are called tie scores. Tie games represent the matches were there was not a winner not a loser player. We consider tie games when the game engine of StarCraft does not design a winner. In order to designate a winner player, the game engine of StarCraft checks if all structures of one player are destroyed.

Comparing figure 3.18(a) and figure 3.18(b), it can be observed that our approach performed better than the built-in AI in most of the games because FCBR-Bot obtained higher scores winning in 61 of 100 games and losing in only 23 games with lower scores. Figure 3.18(c) shows the scores for tie games. In certain games the structures of the players are not destroyed at all. Even when one player has a higher or lower score if not all structures from one player are destroyed, the game engine of StarCraft does not designate a winner and the game is considered like a tie.

3.5.4 General Discussion

Analyzing the results presented in subsections 3.5.1, 3.5.2 y 3.5.3 we observe that the implementation of our approach was victorious in approximately 60% of the played games while the opponent only won approximately 25% of the total games, getting
ties in the rest of the matches. This means that a case-based reasoning and a fuzzy case-based reasoning approach that enables strategical and tactical reasoning can successfully defeat built-in AI opponents in StarCraft independently of the start locations of both players. Therefore, our system can deal with abstract information and a large space of actions and can effectively adapt to varying conditions of the map presented in terrain differences of every region.

Particularly, the results for experiment one gives a total of 60 win games, in experiment two we obtained 58 win games while in experiment three we found 61 win games. The results of all experiments are almost equal, but there is a little difference. We can observe that the highest number of win games was obtained in experiment three, when both players start in a random location of the map. The lowest number of win games was obtained in experiment two, when our bot starts in a selected location and the built-in AI starts in a random location of the map. The reason why the random-random configuration performs better than the start-start and start-random configurations can be influenced by the selection of the enemy performed by the engine of the game. To perform experiment one and two, if the players do not start in the selected locations, we force the system to restart the game assigning new locations until reach the desired ones. Such process can influence the selection of the enemy and can lead to a better performance when using a random-random configuration.

EISBot is a system proposed by Weber et al. [2010] that uses CBR to deal with strategical management. The system uses case-based goal formulation, a technique inspired by case-based reasoning, for strategy selection in the real-time strategy game of StarCraft. In this system, the goal formulation component handles strategy selection and a reactive planner handles second to second actions in the game. Playing the Protoss faction, EISBot was victorious 35% of the times, playing against the Protoss built-in AI of StarCraft on the map of Python. Comparing this results with the results obtained by our approach it can be stated that our system performs better than EISBot on the same map winning in 60% of the matches in Protoss versus Protoss games. In comparison with EISBot, our system incorporates fuzzy case-based reasoning to deal with tactical management and uses other additions to play complete games of StarCraft. Figure 3.19 shows a comparison between EISBot and FCBR-Bot results in Protoss versus Protoss games. The results obtained by our system in the three configurations mentioned before are called FCBR1 for selected vs selected initial positions, FCBR2 for selected vs random initial positions and FCBR3 for random vs random initial positions.

The effectiveness of our system relies on several facts. First, the strategy selected to battle against the built-in AI was properly chosen by human knowledge, which is incorporated in the case bases. Second, the retrieve processes for strategical and tactical management can calculate the best matches and can obtain the most representative cases, in most of the games. Then, the incorporation of restrictions in the reuse process for the use of cases in tactical management adjusts properly to game strategy and
efficiently leads the tactics during game. Finally, the map division into regions lets a proper reduction of the state of the game and enables an abstract representation of the state, leading to a proper adaptation to reason in different regions of the map. Defeating the built-in AI of StarCraft, our system shows its potential to incorporate human knowledge in the reasoning process and to add abstract reasoning of the space.

Lastly, we would like to remark the low time required to train our system to play in a particular map (versus the time required to write a handcrafted behavior to play the same map). Specifically, to record a game a player has to play a complete game (that takes between 10 and 15 minutes in the map that we used). Then, the extraction of the cases and the built of the case bases last the time required to observe the replay, which can be done with the maximum speed that the StarCraft game allows.

3.6 Summary

In this section the results obtained by the implementation of CBR and FCBR approach in StarCraft were presented. The experiments were performed in the map of Python 1.3, playing against built-in AI. Three experiments were developed, the first one selecting the start locations of our system and its opponents, the second one selecting the start location of our system and managing random start location for the built-in AI enemy. And the third one, using random start locations for both players. In all experiments encouraging results were obtained. The FCBR-Bot was victorious in most of the half of total played games. This shows the potential of introducing case-based reasoning to reason about strategical management and fuzzy case-based reasoning to reason about tactical management in real-time strategy environments enforced with incomplete information and with a large space of actions.
Chapter 4

Related Work

Several approaches have been proposed to deal with the large number of objects and actions involved in RTS games in order to make better and novel opponents. In the field of CBR, many researches have published different approaches using CBR to deal with the complexity of RTS video games. In the following sections, we present some research related with the application of CBR technique to different performance tasks in RTS video games. First, research works related with macromanagement are presented. Then, we present some research works related with micromanagement. Finally, we present certain approaches that use other techniques different to CBR.

4.1 CBR for macromanagement in RTS games

In computer RTS games, macromanagement refers to the general economy aspect of the game. This includes constructing buildings, conducting research, and producing units, among other things involving the consumption and expending of resources. Following, we present some research work in real time strategy games based in CBR that focuses in macromanagement.

4.1.1 Case-Based Goal Formulation

Case-based goal formulation is a technique for formulating new goals for an agent using a library of examples [Weber et al., 2010]. Case-based goal formulation makes use of a case library to formulate a goal state. To achieve this goal state the system computes the actions required to reach the goal state from the current state and then builds a totally ordered plan.

The system refers to the number of actions in the generated plan as the planning window size and varies the size of this window during gameplay. A larger planning window is used in the beginning of the game where the plan is unlikely to be invalidated by the opponent, while a smaller window size is used later where plans are frequently invalidated.
Case-based goal formulation is implemented in a StarCraft playing agent called EISBot. The agent consists of two components: a goal formulation component that performs strategy selection and a reactive planner that handle second to second action in the game. EISBot interfaces with StarCraft using the Brood War API.

The strategic component uses CBR approach to extract the game state, retrieve the most similar cases in the case base, formulate the goal and build a plan with the necessary actions to achieve the goal. The reactive portion of EISBot is composed of several managers that handle different aspects of gameplay. With this combination EISBot is able to play an entire match of the game.

The proposed presented in this thesis uses a similar CBR component to manage strategy during gameplay. The difference is that our system is not based on goals. Instead, our case library stores cases composed by features that represent the state of the game and the actions executed in that state. Then the retrieve case contains the actions to perform. Furthermore, our system incorporates a fuzzy CBR component to deal with tactical actions in the game. Similar to EISBot, we use BWSAL library to handle other aspects of the game, like resource management, in order to play an entire match.

Playing the Protoss faction, EISBot was victorious 35% of the times, playing against Protoss built-in StarCraft AI on the map of Python. Comparing this results with the results obtained by our approach it can be stated that our system performs better than EISBot on the same map winning in 60% of the matches in Protoss versus Protoss games. In comparison with EISBot, our system incorporates fuzzy case-based reasoning to deal with tactical management and uses other additions to play complete games of StarCraft.

4.1.2 Case-Based Planning

Case-based Tactician (CAT) is a case-based system that learns to select proper tactics to use at each state during gameplay with random opponents in a real-time strategy game [W. Aha et al., 2005]. CAT employs three sources of domain knowledge, the first source, a state lattice, is an abstraction of the state space, while the second source, a set of tactics for each state, is an abstraction of the decision space. CAT uses Wargus game as test bed. Each state corresponds to the types of constructed buildings; state changes occur when a new building is created. The third knowledge source implies using cases that map game situations to tactics and their performance.

CAT uses CBR approach for selecting which strategy to use in each state, it also use state-specific tactics libraries as case base. Cases are grouped by the building state of the game. The system performs a modified k-nearest neighbor retrieval using the Euclidian distance between eight features in case descriptions.

Therefore, CAT approach concentrates on strategic management where cases are
composed by building states and the actions to perform in such situation (build order actions).

Ontanon et al. [2007] present a real-time case based planning and execution approach designed to deal with RTS games. This approach involves learning behaviors from expert demonstrations to reduce the effort of coding the behaviors, and uses the learned behaviors inside a case-based planning system to reuse them for new situations. They divide the process in two main stages:

**Behavior acquisition.** During this first stage, an expert plays a game of *Wargus* and the trace of that game is stored. Then, the expert annotates the trace explaining the goals he was pursuing with the actions he took while playing.

**Execution.** The execution engine consists of two main modules, a real-time plan expansion and execution (RTEE) module and a behavior generation (BG) module. The RTEE module maintains an execution tree of the current active goals and subgoals and which behaviors are being executed to achieve each of the goals. Each time there is an open goal the RTEE queries the BG module to generate a behavior to solve it.

They use CBR to maintain a current partial plan tree. Initially, the plan consists of a single goal: “win the game”. Then, the RTEE asks the BG module to generate a behavior for that goal. That behavior might have several subgoals, for which the RTEE will again ask the BG module to generate behaviors, and so on.

The main difference with our work is that they use CBR to construct a plan tree based on goals and behaviors while we use CBR to execute strategic and tactical reasoning without building a general plan. Moreover, this system needs to annotate the traces of the game with the goals the player was pursuing while our system automatically extracts cases from replays.

### 4.1.3 CBR for Build Order

Additionally, using *Wargus* RTS game, Weber and Mateas [2009] present a case-base reasoning system for selecting build orders. The system communicates with the integrated agent framework of McCoy and Mateas [2008] and plays complete games of *Wargus*.

The case retrieval process generalizes features of the game state and selects cases using domain-specific recall methods, which perform exact matching on a subset of the case features. This retrieve process is called conceptual neighborhoods approach. An overview of the process is as follow. First, the transform step selects 0 to n generalization methods and applies them to the current game state, where n is the maximum number of generalizations allowed. Next, the recall step performs exact matching using a set of
recall methods. Then the system evaluates the recalled cases by computing a distance metric based on the applied generalization methods. Next, a case is selected from the set of recalled cases using a weighted random selection. Finally, the agent performs the behavior contained in the selected case.

Cases are defined as a game state and behavior pairs. The states encode six features of the game. The behavior specifies a build order action to execute. The system presents three retrieval processes. The conceptual neighborhood selector (CNS) retrieves cases using the conceptual neighborhood approach. The nearest neighbor selector (NNS) performs retrieval using Manhattan distance based on the unit and building counts of all unit types, for both of the players. The random case selector (Random) selects build actions randomly from the set of currently valid cases.

Consequently, this system is similar to our work in the sense that adds a CBR approach to deal with strategy selection and executes build orders in an RTS environment. Nevertheless, we used nearest neighbor approach in the retrieval process, while this method aim conceptual neighborhoods. Furthermore, our agent includes a fuzzy CBR approach to deal with tactics actions in the game. Similar to this system, we use add-on libraries to handle other aspects of the game, like resource management, in order to play an entire match.

### 4.1.4 Building a Player Strategy Model

Case-based reasoning (CBR) approach for learning and predicting individual player strategies has been apply by mining series of actions from replays [Hsieh and Sun, 2008]. The proposed system is capable of learning and predicting individual player strategies, and shows that players provide evidence of their personal characteristics through their building construction order.

This technique uses replays of StarCraft games to analyze player strategies in terms of building construction order and unit creating strategies. Also, case-based reasoning (CBR) approach is used to construct the system for learning and predicting individual player strategies by mining series of actions from replays.

Therefore, the model treats buildings status as game state as well as the action of constructing building as strategy between two states. The game state is based in six features. A ranking mechanism is used to normalize the six feature variables, with shorter distances indicating a higher rank. The system also defines a strategy performance by tracing the results of several replays.

Collected data on building construction sequence is used to analyze and categorize player strategies and playing styles.
4.2 CBR for micromanagement in RTS games

In computer RTS games, micromanagement involves small-scale management of individual units. In other words, it involves focusing on individual units in battle, preserving and making the most use of the units. Following, we present some research work in real time strategy games based in CBR that focuses in micromanagement.

4.2.1 Improving Micromanagement

Szczepánski and Aamodt [2008] present a case-based reasoning system implemented in the Warcraft 3 (RTS game) that focus in the micromanagement of units during battle. The system runs on a custom map made as an arena.

Every second, the current game state is abstracted into a case and compared to previously stored and solved cases. The most similar case is retrieved and the solution provided by the case is executed. The case structure consists of three parts: the condition part, the description part and the solution part. The solution part consists of a strategy that contains actions and behaviors. Actions are simply individual unit orders; behaviors are a set of actions.

The system uses a weighted nearest neighbor algorithm in the retrieval process and implements four different case matching methods that consist in the presorting of units.

Hence, the system use CBR approach to manage tactics at micromanagement level and is tested in an arena field where the battlefield contains two armies fighting each other in a mirror battle.

The similarity with our work relies in the use of CBR to manage tactics that orders military units to execute commands. Differences are that we also use CBR to strategic management. Moreover, we use fuzzy variables to abstract the game state at tactics level. Finally, we integrate our approach with BWSAL library in order to play a complete match of the game.

4.2.2 Learning Continuous Actions Models

Molineaux et al. [2008] combine case-based reasoning and reinforcement learning for selecting actions at the tactical level of gameplay. They introduced CASSL (Continuous Action and State Space Learner), an algorithm that integrates case-based reasoning (CBR) and reinforcement learning (RL) methods that do not discretize action spaces. CASSL was tested in MadRTS engine.

The state space consists of eight features, which are defined relative to the position of the player’s units. At each time point, the agent receives a state vector with the values of these features and selects an action to execute. An action in this space corresponds
to an order given by the agent to a group of units. CASSL applies a nearest neighbor model in the retrieve process of the CBR cycle.

CASSL applies CBR/RL focuses on unit combat. In contrast, our work covers strategic and tactical management. Moreover, CASSL uses a hybrid CBR/RL technique and our approach relies only in a CBR system with fuzzy attributes.

4.2.3 Transfer Learning

Also, using MadRTS as test bed, Sharma et al. [2007] present a multilayered architecture named CAse-Based Reinforcement Learner (CARL). It uses a novel combination of Case-Based Reasoning (CBR) and Reinforcement Learning (RL) to achieve transfer learning while playing against the game AI.

The system allows learning at both strategic and tactical levels of abstraction. In particular, they use CBR as an instance-based state function approximator for RL, and RL as a temporal-difference revision algorithm for CBR.

Upper layers reason about strategy while lower levels are concerned with ever finer distinctions of tactics. The top-most strategic level is based on a hand-coded strategy. The middle tactical layer uses a hybrid case-based reasoning and reinforcement learning approach. Specifically, this layer makes tactical decisions over an action space. The lowest layer incorporates a reactive planner, scripted to perform the tasks.

We use a similar architecture to reason about tactics. In our case, we use a fuzzy CBR approach to reason about tactics and also we add a lower layer that uses a scripted agent to improve micromanagement. Nevertheless, our system uses fuzzy CBR to reason about tactics (not CBR/RL hybrid) and it adds a CBR approach to reason about strategy.

4.3 Other Approaches in RTS games

The SORTS agent is capable of playing an entire standard RTS game, including the use of high level strategy [Wintermute et al., 2007]. SORTS includes algorithms based on human perception to form unit groups and to focus attention. Unit micromanagement is handled in the middleware with the use of finite state machines (FSMs). To enable a larger degree of tactical coordination between units, the FSMs that handles military and resource gathering units are managed by global coordinators. These coordinators employ simple learning to enhance the efficiency of the agent.

Dynamic scripting [Spronck et al., 2003] is a technique for achieving online adaptation of computer game opponents. In dynamic scripting, scripts are created online, during gameplay, based on rules extracted from a rulebase. The technique is based on reinforcement learning and adaptation proceeds by rewarding or punishing certain rules according to their influence on the outcome. Ponsen and Spronck [2004] applied
a modified dynamic scripting algorithm to an RTS game (Wargus). They employ different rulebases for the different states of the game and their implementation executes weight updates based on both an evaluation of the performance of the game AI during the whole game and on an evaluation of the performance of the game AI between state changes. Moreover, they propose the use of an offline evolutionary algorithm to enhance the performance of dynamic scripting, by evolving new domain knowledge.

Olesen et al. [2009] proposed the use of the NEAT and rtNEAT neuro-evolution methodologies [Stanley and Miikkulainen, 2002, Stanley et al., 2005] to generate intelligent opponents in real-time strategy (RTS) games, with the objective of adapt the challenge generated by the game opponents to match the skills of a player in real-time. NEAT is a method used to construct artificial neural networks automatically which is based on Topology and Weight Evolving Artificial Neural Networks (TWEANNs). Evolution adjusts both the connection weights and the topology of the network. Real-time NEAT (rtNEAT) is able to complexify neural networks as the game is played, making it possible for agents to evolve increasingly sophisticated behaviors in real time.
Chapter 5

Conclusions

In this thesis we have presented the use of fuzzy sets and case-based reasoning approach for dealing with strategical and tactical reasoning in the real-time strategy game of StarCraft.

Case-based reasoning is the process of solving new problems based on the solutions of similar past problems. Four processes compose a general CBR cycle: retrieve, reuse, revise and retain. The description of the CBR approach and its processes were shown in Chapter 2. A description of fuzzy systems and its convergence with case-based reasoning was presented in the same chapter. Furthermore, an introduction to the real-time strategy game of StarCraft, the Brood War Application Programming Interface that interacts with StarCraft, the BWAPI Standard Add-on Library (BWSAL) that develops several add-ons for BWAPI that can be useful for a wide variety of AIs and the Brood War Terrain Analyzer (BWTA), an add-on for BWAPI, which analyzes the current StarCraft map were presented in the same chapter.

Chapter 3 described the implementation of our approach in StarCraft. First, to deal with the vast space of actions that a match of StarCraft proposes, we showed the division of the game in four categories: resource management, strategical management, tactical management and micro management. Then, the selected strategy to battle against built-in AI and the parameters of matches were described. Subsequently, we showed the implementation of case-based reasoning to deal with strategical management; we described the format of the cases used to keep the human knowledge. Then, we presented the implementation of the retrieve, reuse and revise processes of CBR cycle adapted to StarCraft. Moreover, we showed the matching treat based in Euclidean Distance.

After that, also in Chapter 3, we described the implementation of fuzzy case-based reasoning to deal with tactical management. To accomplish tactical reasoning and to deal with the large amount of actions and information that an RTS game like StarCraft proposes we presented the division of the state of the game into regions. Then we explained the features of the cases for tactics using a fuzzy representation of the world. Subsequently, we showed the membership functions that enable the abstraction of the game. Later, we presented the retrieve, reuse and revise process of the fuzzy CBR cycle
adapted to StarCraft. Moreover, we showed the fuzzy matching technique implemented to compute the similarities between cases in the retrieve process. Finally a summary of the implementation of fuzzy case-based reasoning in StarCraft was presented.

5.1 Research Questions Revisited

How can CBR approach incorporate human knowledge for decision-making? Case-based reasoning can incorporates human knowledge in the decision-making process using a knowledge base that represents the experience of the system. A feature of the CBR algorithm is that it is able to use past experiences to deal with actual problems. In the environment of StarCraft, the knowledge acquisition was performed using *replays* of the game where we extracted the human knowledge. In strategical management, the cases are composed by numeric values that represent the state of the game and by a list of actions that describe a plan to build units. In tactical management, the cases are composed by fuzzy values that represent the state of the game and by a set of actions that leads the tactical movements of the units. Using these representations, CBR can incorporates human knowledge in the decision-making process in StarCraft.

Is it feasible to treat the attributes of an RTS game like StarCraft as fuzzy variables? In StarCraft, it is possible to treat the attributes or features of the game like fuzzy variables. In the implementation of our FCBR agent, we use human knowledge to define the fuzzy sets that represent the state of the game. The attributes of an RTS game like StarCraft can be represented using linguistic variables. The linguistic variables enable an abstraction of the state of the game and enhance reasoning with a large space of actions. Therefore, using appropriate membership function it is feasible to represent the state of the game as fuzzy variables. Moreover, the use of fuzzy variables to represent the cases enables an abstract reasoning process in CBR.

How can CBR approach deal with incomplete information? The structure of the features that describes the cases allows CBR to propose solutions in environments that enforces incomplete information. In StarCraft, incomplete information is presented in the fog of war. Players can see only regions where they have units. The case structured used by our algorithms and the addition of fuzzy values in the representation of cases enables CBR to deal with incomplete information. Results showed that our system is able to defeat the built-in AI of StarCraft under incomplete information conditions represented in the fog of war.

How can we combine strategic and tactical reasoning for an RTS game in CBR technique? Using a fuzzy CBR approach for tactics and a CBR approach for strategies, we incorporated strategical and tactical reasoning in an RTS environment. Using human knowledge extracted from *replays* enables our system to reason with strategies. The addition of fuzzy logic to CBR allows representing an abstract state of the game.
Moreover, the introduction of the division of the map into regions with a fuzzy representation of the state of the regions enables our system to reason about tactics in StarCraft.

Is it the combination of fuzzy theory and CBR a good approach in the development of RTS game AI? Fuzzy case-based reasoning is a good approach in the development of AI modules for RTS environments. The results presented in subsection 3.5.4 show that a fuzzy case-based reasoning approach that enables tactical reasoning in addition with a case-based reasoning approach for strategical management can successfully defeat built-in AI opponents in StarCraft independently of the start locations of both players. Therefore, FCBR approach can deals with abstract information and a large space of actions and can effectively adapts to varying conditions of the map presented in terrain differences of every region. Our system shows its potential defeating the built-in AI of StarCraft in approximately 60% of the matches.

The main objective of this thesis was successfully reached. We built an agent-based system that incorporates case-based reasoning with fuzzy attributes to accomplish strategical and tactical reasoning and wins the game of StarCraft. Our system wins against the built-in AI of StarCraft in approximately 60% of the matches. Moreover, it can deal with incomplete information and is able to abstract the state of the game using fuzzy sets in case representation. Furthermore, the division of the map into regions and its fuzzy state representation enable our system to adapt to different configurations of the map, increasing its performance.

5.2 Contributions

The main contributions of this thesis are: a case-based reasoning framework to deal with strategical management in StarCraft and a fuzzy case-based reasoning framework to deal with tactical management in StarCraft. We implemented the fuzzy case-based reasoning approach in C++ code. We linked up our system to BWSAL, BWTA and BWAPI libraries to interact with StarCraft game. Moreover, we introduced automatic extraction of human knowledge from saved replays played by StarCraft players. We built two case bases that are used by our systems, one for strategies and one for tactics. Both case bases keep the human knowledge in form of cases and are saved in text file format. In addition, we introduced the idea of map division into regions to enhance tactical management. We also introduced an abstract representation of the state of the game using fuzzy game features with its corresponding membership functions. Finally, we presented a micro management framework that incorporates some methods to improve micro level organization of units in StarCraft.
5.3 Future Work

There are several research directions for future work in this thesis. The first direction is to evaluate the potential of our approach in transfer learning tasks, such as playing all three races in StarCraft. Actually, our system plays Protoss versus Protoss games. One possibility to enhance our system to play all races is using CBR approach. It is possible to use CBR to extract human knowledge from *replays* where different combination of races engages in battle. However, it is necessary to investigate how to index different race cases in the case base and if it is reliable to merge the information of different races or use a system with multiple case bases. The introduction of information about different races might change the performance of the CBR algorithm. An important aspect to take into account is the development of a better micro management. Different type of units use different attacks depending of the type of enemy. Micro management is crucial while playing RTS games. Hence it is necessary to develop a micro organization that can be applied with different races.

Additionally, like second direction, we would like to systematically explore how the knowledge learnt in a set of maps can be applied to a different set of maps. The enhancement of our system to reason in different type of maps can also be implemented using CBR. For instance, the information of different maps can be compiled in single databases that can be loaded at the beginning of the game. The system must verify the map and selects the database or the system can combine multiple databases. It is necessary to explore the performance of CBR using multiple databases for different maps.

The third direction of research involves to experiment adding a case retention module in our system that retains automatically all the behaviors that had successful results while playing, and also annotating all the cases in the case base with their rate of success and failure allowing the system to learn from experience. Case retention can improve the behavior of our system, learning new cases from experience. It is necessary to develop a module that executes new actions when the system has no information about the state of the game. It is necessary to adapt the successful behaviors to case format and to index the cases in the case base.

The last direction of future work is to merge the directions mentioned before and implement a system to contend in the AIIDE StarCraft AI Competition [AIIDE, 2011]. AI And Interactive Digital Entertainment Conference (AIIDE) is the definitive point of interaction between entertainment software developers interested in AI and academic and industrial AI researchers. The purpose of AIIDE StarCraft AI Competition is to foster and evaluate progress of AI research applied to real-time strategy (RTS) games.


B. Heuser. The evolution of strategy: thinking war from antiquity to the present. Cambridge Univ Pr, 2010. ISBN 052115524X.


