Explanation-aware army builder for Warhammer 40k

Nenad Zikic

TDT4501 - Fordypningsprosjekt i Datateknologi
Høst/Fall Semester 2015

Artificial Intelligence Group
Department of Computer and Information Science
Faculty of Information Technology, Mathematics and Electrical Engineering
Norwegian University of Science and Technology

Supervised by Anders-Kofod Petersen

Abstract

The goal of this project is to design an explanation-aware, case base reasoning system that can create a winning army in the Warhammer 40k strategy board game when presented with the opposing army.

This system will use object oriented case based reasoning with general domain knowledge to simulate a Warhammer 40k domain. It will be able to maintain itself through any changes in the game versions, as well as proactively think and learn about the game, without any outside stimulus, by using maintenance policies. Explanations are used to teach new users, help expert users, and build trust towards new and expert users alike. State of the art and a structured literature review protocol are used to approach this project methodically.

The paper succeeds in fulfilling the goals and presents a designed system, presenting the background knowledge necessary to understand it, the state of the art, the architecture of the system and the actual design. The paper evaluates and discusses the limitations of the design, its contributions, and future work.

Preface

This is a project report written for the Fordypnings Emne, or Specialization Course, for the Master in Computer Science (Datateknikk) at the Norwegian University of Science and Technology (NTNU). It has been conducted at NTNU, at the Department of Computer and Information Science (IDI), in the Artificial Intelligence (AI) department, in the Fall semester of 2015.

This project was supervised by Anders Kofod-Petersen. I would like to extend my thanks to him first and foremost for helping me with the project and providing me with the opportunity to work on the project.

I would also like to thank Agnar Aamodt for his help in case-based reasoning and machine learning, as well as the faculty at IDI for their motivation and assistance with writing a good project.

Finally i would like to thank my friend, Drikus Kuiper, for his assistance with Warhammer 40k resources.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This project is an exploratory work into explanation-aware case-based reasoning. I will attempt to design a system that will build upon the basics of Case-Based Reasoning (CBR) and the fundamentals of explanations, and create a solution for an instance of a set of problems. The overarching class of problems is creating an artificial intelligence system with complex, ever-changing environments.

CBR has first emerged from the study of human memorization and understanding (Schank, 1982). At its core, a CBR system uses previous cases, often called solutions, and applies them to new problems, when presented with a problem. Since then, it has been worked and approached in many different ways (de Mantaras et al. 2006).

Explanation-aware systems can reason about their actions. Reasoning of ones actions is a sign of intelligence (Sormo et al. 2005). If we can reason about our own actions, then we do not do actions out of instinct, but rather we do them with a purpose. In the same sense, if the AI system can reason about its actions, it has a purpose, and thus has to have some form of intelligence. As we seek to build artificial intelligence, explanation aware systems should be able to as closely as possible mimic intelligence. Furthermore, an explanation aware system can teach both novice and expert users about the domain and how the system reached this conclusion (Roth-Berghofer 2004).

The rest of this chapter will present my personal motivation for doing this project in Section 1.1. The goals and research questions of the project are presented in Section 1.2, while the research method is presented in Section 1.3. Finally, Section 1.4 briefly describes the rest of the project structure.

1.1 Background and Motivation

I have been playing video games since I was five years old. My interests mostly as a child were in video games, and as such I wanted to enter an education where developing games would be my job. Around high school I also found myself drawn to board games, and especially role playing games. From then on out, I had played many tabletop role playing games starting from Dungeons and Dragons[1], to home made systems, to Warhammer 40k[2] in recent years. All the while, I had also maintained my love and interest for video games.

In the past few years, I had been dissatisfied with the progression of AI, or the computer player, in video games. Graphically, video games have progressed significantly in the last two decades, but the computer player has in most cases largely stayed the same. A good example of this is the Total War series, especially Rome Total War, and Rome Total War 2 [3], two games that I really enjoy. However, in the span of a decade, the computer player is still the same tool, just wrapped in a nicer package. In most strategy games, as well as in general any video game, the AI is balanced out by simple cheating. The AI can not win against better players, so they get more resources, whatever the resources may be. This cheating AI has sparked my interest towards AI and AI development. This has changed my goal from simple video game programming, to AI programming in video games, and the hopes to create an excellent video game AI. A long term goal evolving from this is the creation of an AI that can teach itself while playing against other players.

[1] http://www.wizards.com/default.asp?x=d20/article/srd35
[2] http://www.games-workshop.com/en-GB/Warhammer-40-000
[3] http://www.totalwar.com/

Turn based strategy games are very much like board games, except with animation and graphical effects. Warhammer 40k is a turn based strategy board game, and a very complex game as well. It features seventeen different playable races and factions, a myriad of units in each race, and a great choice of equipment for each unit. Furthermore, games are often done in different *scenarios* or *missions*, and the choice of units and equipment may depend heavily on the mission. Finally, before starting players agree on a resource pool of points, usually between 1000 to 1500, and then they purchase units and equipment to create an army.

This task of creating an army is, in my opinion, extremely well suited for CBR. It is a very complex environment, with almost countless possibilities. Eager generalization simply may not be possible in such a complex environment, but lazy generalization, which CBR is, may be. CBR largely mimics the thought patterns of humans (Schank, 1982) and as such I believe is able to function in this environment. Given cases, or previous armies, a CBR system could deduce what army would be a good choice, when presented with the opposing army. Not only that, but it could also explain how it acquired the answer (Roth-Berghofer 2004, Sormo et al. 2005) and be able to cope with the changes of rules, units and more, without having to rebuild an entire knowledge base. Finally, the AI created here, if successful, could have application in not only games, and potentially video games, but also in any problem set that requires a division of resources in a complex, ever changing environment. This is precisely the motivation for this project.

1.2 Goals

In this section I will present the main goal of the project, and the sub-goals associated with the goal.

Goal 1 Design an explanation-aware CBR army builder for Warhammer 40k

The first goal of the project is the same as the title. I will aim to design an explanation aware CBR system, that will be able to create armies in Warhammer 40k. There are three important overarching aspects to this system, and each one will be presented as a sub goal below.

Sub-Goal 1 Design a CBR army builder for Warhammer 40k

The first goal is to create the underlying system. This system will be based on knowledge-intensive CBR methods, and given information about the opposing army, should be able to come up with an army structure that has a good chance of beating it. The aim is to build a system that can reliably win at least 50% of the games.

Sub-Goal 2 Design an explanation aware CBR system

The second goal goes hand in hand with the first. In order to create a system that will be able to reason about it's choices as well as an understanding system (Schank, 1986) we need to have explanations. This will raise confidence and trust towards the system from the user (Moore and Swartout 1988). Therefore, I will aim to create an explanation system aimed towards novices and experts alike, with additionally, having options and being able to have a compromise-driven case-based method (McSherry 2003). The value of the explanations will be critiqued by both novice and expert users.

Sub-Goal 3 Design a system that can evolve within the environment that is Warhammer 40k

The third and final goal is to create a system that will be able to evolve. Warhammer 40k was created in 1987, and has so far undergone seven different editions, the last one being released in 2014. With small update packages being released regularly, it is important that our system is able to analyze new data, by reading it and integrating it into it's knowledge base. Not only that, but the metagame can constantly change, and therefore our system needs to be able to constantly change to adapt to the new metagame. If the system can be given new data, and it can subsequently integrate that into both old and new cases, while at the same time adapting the old cases to match new statistics and information, the system is able to constantly evolve within the environment.

Goal 2 Present the state of the art

The second goal of the project is to present the state of the art of the system. This project aims to be a methodical, scientific project, and the presentation of the state of the art will greatly aid with this. Furthermore, the reuse of knowledge is very important. By providing a methodical approach to the project, we build a strong foundation for any other projects in the future.

1.3 Research Method

On the high-level of methodology, we will approach our goals using the scientific method. In this project, the focus is on researching the goals, formulating the hypothesis and designing the system. The master thesis to follow this work will focus primarily on the implementation, experimentation, evaluation and expansion of the hypothesis and formulating a general solution to the problem.

On a more low-level approach, we will be using the set out steps by Paul R. Cohen and Adele E. Howe, in their paper *How Evaluation Guides AI Research* (1988). We will try to approach and evaluate each step of the research, as often AI research evaluates empirical data, but not much more. The five stages of evaluation are: Refining a topic to a task, Designing the Method, Building the Program, Designing the Experiments, and Analyze their results. The evaluation of the work done in this project will be presented in Chapter 5.

The main reason to follow this methodology is to evaluate all stages of research. The secondary reason is to have a well documented solution to a unique problem, that can hopefully be applied to further research in different research and application fields.

1.4 Project Structure

Chapter 2 introduces background theory and motivation for the project in detail. Chapter 3 focuses on the Architecture of the system and is closely linked to Chapter 4, which focuses on design. Chapter 5 outlines the evaluation of the project, contributions and future work to be done in the field, as well as well as including any limitations of the project.

Chapter 2

Background Theory and Motivation

This chapter will introduce the important theory topics for this project. The project will build on the theory presented in this chapter. In Section 2.1 the basic rules and important notions in Warhammer 40k will be presented. In Section 2.2 we will discuss the basics of CBR and knowledge-intensive CBR. Section 2.3 will cover explanation-aware computing. In Section 2.4 we will discuss the ever-changing environment of systems in general, maintenance in CBR systems, as well as the concept of metagaming. Section 2.5 contains the state of the art, in which we will discuss the structured literature review protocol and the motivation for this project.

2.1 Warhammer 40k

Warhammer 40k is a tabletop turn-based strategy game, played with two or more people. It is set in the 41st millennium and the genre of the game is strategy science-fiction. The game is intended for mature audiences, and while the game itself has nothing that could be harming for younger viewers, the core rulebook, as well as the additional books do deal with mature themes.

The game is played on a 6x4 foot table[1], for a standard 1000-1500 point match, and the distance is measured in inches. The game is played in turns, where each player gets a player turn in a game turn. Once all the players have had a turn, a new game turn starts. Each player turn is divided into the start of the turn, movement phase, psychic phase, shooting phase, assault phase, and the end of the turn. Within each phase of the turn, a player does what they want and can, and when that phase is done they move onto the next. The game is played using one type of die, the six sided die, though two other die types are sometimes required. For a three sided die roll, the six sided die result is halved and rounded up. For the special sixty-six sided die, two six sided dice are rolled, one for the *tens* and the other one for the *ones*.

Before starting the game players must first prepare a few things. First, they must decide on a mission. Missions are the game goals, or objectives. They have victory conditions, which when a player fulfills win him or her the game. Missions can be pre-made by Games Workshop, or created by one of the players. The missions are important, as they influence heavily what kind of army composition the players will choose. Furthermore, most missions are not fought in the open and usually have some sort of terrain features. These terrain features also influence what kind of army the players will create. A lot of open space might make the bring more tanks and other armored units, where as a lot of trees or cover might impede these units and make the players pick infantry over them. There are possibilities for playing without a mission, but nonetheless the players have to agree on all the parameters before the game starts.

Finally, the largest impact on the army is the race. There are nine races in the 7th edition of Warhammer 40k, and each of them has unique equipment and units, and therefore a unique army and play style that differs from one another. Furthermore the human race, or the so called imperium armies, are subdivided into an additional nine factions, which makes a total of seventeen different armies. The imperium armies do not differ as much as the other races do in between themselves. They still have different units units and equipment, but there may be overlap between the armies.

[1]Games Workshop is based in the United States of America, and the game uses the Imperial System for measurements. It is not necessary for the theory to convert these units to metres and centimetres, and it would only confuse the players and the readers. Nonetheless, the official conversion is 1 foot = 0.3048 metres, or 30.48 centimetres, and 1 inch = 2.54 centimetres. Furthermore, 1 foot contains 12 inches.

Once the players have prepared everything, they can begin playing the game. The system that we are trying to create will mostly deal with this preparedness phase and the army-building. Therefore, the next two subsections will note the unit statistics, or attributes, and the unit types, as well as equipment, and the actual process of creating an army.

2.1.1 Equipment, Units and Unit Types

The units in Warhammer 40k are the building block of any army. Units may be grouped together as squads, either because of the history of the units, or because they are weak individually, or because of the story and lore of the units. Finally, most units and squads are grouped in a specific Unit[2], and rarely do singular units and squads go on their own. Each unit has a characteristics block, in which the player can find all the characteristics of a unit. The characteristics block is shown below in Table 2.1.

Unit Name	WS	BS	S	T	W	I	A	Ld	Sv
Space Marine	4	4	4	4	1	4	1	8	3+
Ork Boy	4	2	3	4	1	2	2	7	6+

Table 2.1: Unit Characteristics

Table 2.1 shows the entire profile of a unit. Each unit has 9 characteristics, but some units may have a 0 or a dash through a characteristic. This means that that particular unit does not have that characteristic. These characteristics are very important to the game, as they are the attributes of each unit.

- WS stands for *Weapon Skill* and represents the ability of the unit to hit another unit in melee combat. The higher the skill is, the higher the chance to hit.
- BS stands for *Ballistics Skill* and represents the ability of the unit to hit another unit in ranged combat. Similarly to Weapon Skill, the higher the Ballistics Skill, the higher the chance to hit.

[2]In this case a Unit is a collection of squads with a purpose. An example is an anti-tank Unit, which can contain many anti-tank squads.

- S stands for *Strength* and represents the physical strength of a unit.
- T stands for *Toughness* and represents how resistant a creature is to damage and pain.
- W stands for *Wounds* and represents how many hits a target can take before they fall down, either dead or wounded and are out of battle. The larger the creature, the more wounds it usually has.
- I stands for *Initiative* and represents how fast the creature reacts. In close combat, a higher initiative creature hits first.
- A stands for *Attacks* and represents how many attacks a creature gets in close combat.
- Ld stands for *Leadership* and determines the morale of the unit. The higher the characteristic, the more brave the unit is in combat.
- Sv stands for *Armour Save* and determines the chance to avoid damage when getting hit. Unlike with other characteristics *a lower armour save is better*. A creature may never have armour save that is lower than 2+.

Units can, and usually do, have other special abilities not indicated in their basic statistics, but indicated on the specific unit page. These abilities are in addition to the creatures characteristics.

Units also fall into unit types. Unit types have their own special abilities and rules. There are in total twenty four unit types, ten of these being non-vehicle, and the other fourteen vehicles. Some unit types, like infantry, have no special rules, while others, like bikes and jetbikes have special rules. Some unit types even have special rules dependant on their race, and some races lack certain types of units all together.

The unit types are listed below, in Table 2.2 and Table 2.3 but will not be explained in detail, as they take over forty pages in the rule book. Rather, it is noteworthy to state that unit types can influence and even win battles, and are therefore an important factor to consider in making an army.

Non-Vehicle Unit Type
Infantry
Bikes and Jetbikes
Artillery
Jump Units
Jet Pack Units
Beasts
Cavalry
Monsterous Creatures
Flying Monsterous Creatures
Gargantuan and Flying Gargantuan Creatures

Table 2.2: Non-Vehicle Unit Types

Vehicle Unit Type
Vehicle Squadrons
Transports
Flyers
Chariots
Open-Topped Vehicles
Heavy Vehicles
Fast Vehicles
Skimmers
Walkers
Tanks
Super-heavy Vehicles
Super-heavy Walkers
Super-heavy Flyers
Vehicle Upgrades

Table 2.3: Vehicle Unit Types

There is a last classification of unit types, which are Characters. These special characters are named, and are usually very powerful and expensive units. They usually represent army commanders, or warlords, and are discussed more in detail in Subsection 2.1.2.

Finally, units have equipment, which is the last major part of a unit. Each unit comes with specific equipment, which in itself is dependant on the race and the specific unit. Sometimes the units have the option of upgrading their equipment at an extra cost as well. Equipment determines the range of engagement, the number of attacks, the armour penetration power of these attacks, their total damage and more. Equipment plays a vital role in the game, and there is countless pieces of equipment. As an example, the Space Marines, one of the factions of the imperium armies, have over fifty different pieces of equipment, usable by the various units in their armies.

2.1.2 Creation of an Army

As mentioned before, the first thing the players need to do before creating an army is to decide on how many points they will have to make the said army. Different units have different costs, and this can influence how a game progresses. The standard army creation points are 1000-1500, and the game takes a few hours. Larger armies, of 2000-3000 points are possible but they take longer to finish.

When the points are decided, the players decide on the second factor: Unbound or Battle-Forged armies. Unbound armies means that players can take any units they wish from their selection, as long as the total combined cost is not over the set point limit. Battle-Forged armies have strict organizational requirements; all units must be organized in detachments, and these detachments are presented in the rulebook.

A warlord unit is needed to lead the battle. Usually this is a special character-type unit, and usually a unique unit, of which a player may only have one of in an army. A non character-type unit can be selected to become the warlord, but does not gain any benefits other than the warlord traits. Warlord traits further add complexity to the game by giving the players the ability to change, or seize victory points and move closer towards winning the game.

Finally, when creating an army races matter substantially. As we discussed before in Section 2.1 there are nine races, and nine sub-divisions inside the imperium armies, or the human race, making a total of seventeen playable factions. Each race/faction has predetermined alliance rules, and multiple races can feature on the same board. There are four alliance rules; *Battle-Brothers* are races or factions that work extremely well together, and have no penalties. *Allies of convenience* work fine together, but lose bonuses from the warlord and are unable to come within one inch of one another on the playing table. Furthermore, they usually can not interact with the other race with their special abilities. Next is the *desperate allies*, which functions the same as allies of convenience, but if they are within six inches of their allies they must roll a six sided die. On a roll of one, the unit must either move away or can not do almost anything this round. Finally, the last state of alliance is *come the apocalypse*, where units function the same as desperate allies, but can not deploy to battle within twelve inches of one another. These alliances are important to consider when making an army; sometimes taking a penalty might provide an edge against the enemy player, other times it might be a double-edged sword and work unfavourably for the player.[3]

[3]All of the rules in this section were taken from the rulebook of Warhammer 40k 7th edition. However, for a reader that would like to see the core rules, a shortened version of the rules is available for free, courtesy of arbitorian@gmail.com. They can be found here: `https://dl.dropboxusercontent.com/u/4104995/Games/7edRef_V6.pdf` and were last updated in August 2015, as of the time of writing.

2.2 Case-Based Reasoning

The basics of CBR are most easily explained through the original figure presented in Aamodt and Plazas *Case-Based Reasoning Foundational Issues* (1994). This cycle is sometimes also referred to as the 4 R's of CBR, and presented in Figure 2.1.

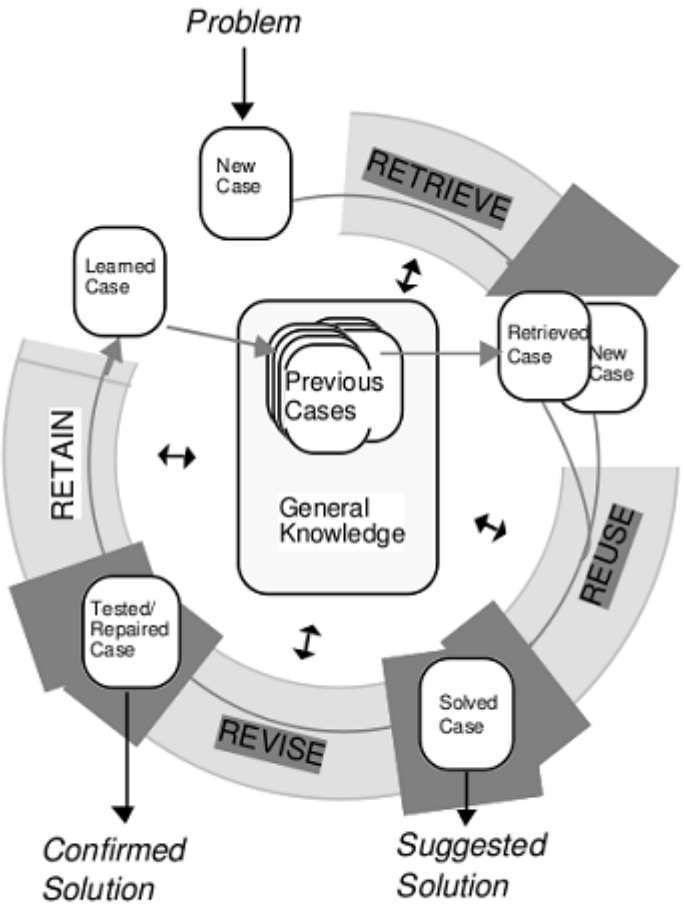

Figure 1. The CBR Cycle

Figure 2.1: The CBR Cycle (Adapted from Aamodt and Plaza, 1994)

The CBR cycle that is presented in Figure 2.1 has four main steps: Retrieve, Reuse, Revise and Retain. Furthermore, the cases are an important aspect of any CBR System, and can be represented in many ways, depending on the system and the complexity of the domain.

2.2.1 Case Representation

Before we discuss the four steps of CBR, we should first take a look at the cases. There are six important points for a good case representation, and in general for good representation in AI, as mentioned by Richter and Weber (2013):

- Representational adequacy, which asks "Is it possible to represent everything of interest"?
- Inferential adequacy, which asks "Can new information or knowledge be inferred"?
- Inferential efficiency, which asks "How easy (computationally) is it to infer new knowledge"?
- Acquisitional efficiency, which asks "How easy is it to formalize new information/knowledge"?
- Clear syntax and semantics, which asks "How easy is it to clarify what is allowed or not"?
- Naturalness, which asks "Are the representations easy to use and understandable"?

In order to have a good case representation, we will try to answer every single one of these questions with our cases.

As mentioned in Section 2.1, each unit has a set of nine attributes. Furthermore, each unit has a unit-type and equipment. Each unit, can therefore be presented as an object with several attributes: the basic nine attributes of the unit, unit-type and the equipment it is wearing. However, while the basic nine attributes and the unit-type are consistent, and fairly simple, the equipment itself possesses various attributes. Therefore, the equipment should also be presented as an object.

A squad inherits a unit object, and the army class inherits a squad object. Squads consist of one or more units, and an army is a collection of multiple squads. This is presented in Figure 2.2.

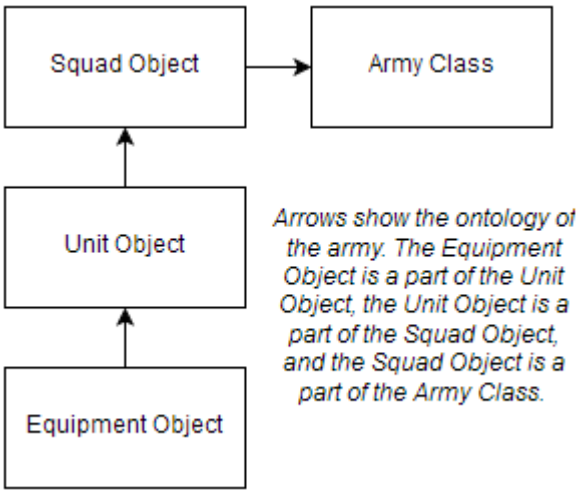

Figure 2.2: Classification of the cases

In Figure 2.2 the ontology is shown. An equipment object is a part of a unit object, which is a part of a squad object, which is ultimately a part of the army class. We can say that any changes at the bottom of the ontology affect the objects above it. Therefore, a change in an Equipment Object changes the Unit Object, Squad Object and Army class, while a change in the Squad Object does not change the Unit Object and Equipment Object, but changes the Army Class.

However, this is not enough for our system. We also need general domain knowledge, which describes the mission and conditions on the mission, as well as the alliance rules. The notion of general domain knowledge will be further discussed in Subsection 2.2.5. With the general domain knowledge and the object-oriented case representations, the question of representational adequacy is fulfilled.

The next three questions have to do with inferring and acquiring knowledge for the system. New information in the system can be inferred, either through similarity measure or through analyzing the cases. Computationally, inferring new knowledge is cheap. It comes naturally with the similarity, adaption and inference from the problem presented. Finally, formalizing that knowledge is creating a new object in the case base, which is a standard part of the CBR cycle, the retention.

Object-oriented approach is a very natural way of representing the units, as after all, this is how they are originally represented in the book, as blocks of information. Furthermore, alongside with general domain knowledge, the syntax for the cases should be very easy to clarify. This answers the two last questions of Richter and Webers list, and complete the case representation.

2.2.2 Retrieval

The first step is to retrieve a previous case from the knowledge base. The actual retrieval mechanism depends on what kind of system we want, the performance of the said system, and our desired accuracy of retrieval. Retrieval has been the subject of much research (de Mantaras et al. 2006) and many different retrieval mechanisms exist. They are in general, split into surface similar and structurally similar retrievals. Typically a surface similar retrieval is represented as a number, usually from 0 to 1, based on the similarity measure. An example is the k Nearest Neighbour approach (k-NN), which returns k nearest (most similar) cases. On the other hand structural similarity goes deeper into domain knowledge, and are likewise more computationally expensive, but also more accurate and relevant.

The case representation is very similar to Creek (Aamodt 1994). There too, does the system have both cases and general domain knowledge. In Creek, a two-step process is used to retrieve the cases: first a surface similarity assessment, to retrieve a set of potentially similar cases, and then a structurally deeper search involving the general domain knowledge on those cases. The retrieval system in this project will attempt to use a similar notion when retrieving the cases.

A final part to the discussion of retrieval is the notion of compromise. This is a point raised in the initial stages of discussion with and by my supervisor, and I wholeheartedly agree on it. If we assume that our retrieval system works flawlessly, we are still hindered by one notion: player choice. A player may not have all the miniatures to play with the retrieved case, in fact he may not even possess any of them. Furthermore, a player may simply express their dislike towards playing a certain race or faction. Therefore, we should have a method that can provide an alternative answer, or modify our answer. This small notion of freedom can raise the users satisfaction of the system greatly, as they have control over the retrieved cases.

McSherry in *Similarity and Compromise* (2003) proposes a compromise-driven approach to retrieval. Though our system is not a recommender system, the main idea stays the same; the system should be able to weigh the different armies according to the player requirements and be able to retrieve at least a set of cases using those requirements. Again, explanations will play a key role in this retrieval; if we are able to retrieve an army, but that army performs substantially worse than the retrieved army, then the user needs to be informed of that.

2.2.3 Reuse and Revision

Reuse and Revision are the second and third steps of the CBR cycle. Reuse in its simplest form is just applying the retrieved problem and solving the problem presented. However, more commonly we need some form of adaption of the knowledge, as after all, if we have the solution to every problem in the system the system is already solved, and there is no notion of learning. Revision is the actual testing of the (often) adapted solution and its performance. As the name suggests, if the solution has not performed well it will be revised, with possible changes to it.

Reusing a previous case in Warhammer 40k will almost always entail some kind of adaption. Let us look at a simple example. Let us suppose that we have an army of Space Marines and we are going to fight an army of Orks and also suppose the army of Space Marines won the previous three battles against Orks. The top suggestion would be to use the same army again. However, this time the Orks brought vehicles with them and the mission takes place in an open area. Our Space Marines do not have equipment to handle armor, and if we do not adapt a solution to this new problem, by giving our Space Marines equipment to handle heavy armor, we will not win. Therefore, the solution is adapted, and the Space Marines are given a Missile launcher with Krak Missiles (Krak missiles have higher armor penetration), to combat the longer range and the armor that the vehicles have. Since this also costs points, we will need to subtract some other equipment from other units, or possibly even subtract some units in the army.

While this is a simple example, it illustrates that a small change can have a big impact in army compositions. Even an addition of different kind of equipment can already make a difference from one battle to another.

Therefore, we will almost always adapt solutions, but we will also need to create some sort of threshold, so that the performance of the system does not suffer, as this is a very expensive step of the CBR cycle.

The adaption method will in most cases be simple substitution. This means that some parts of the solution will be exchanged for other parts. In our example, some units equipment will be exchanged for a missile launcher. Similarly units and squads will be exchanged for others based on the need and performance.

Transformation adaption will be the second method to adaption, which will be used closer to the start of the systems life-cycle. Once the system has obtained sufficient data about a race, it will not need to use the transformation adaption as much, due to the knowledge it possesses. However, early on, the system may need to develop different kinds of command structures and detachments, as explained in the Subsection 2.1.2, which will change the structure of the army.

When the army composition is adapted to the new problem it needs to be tested. After playing, the data can be entered back. In our case, the performance of the army is entered. If the army has won, then the adaption is successful. If not, then we need to revise the adaption. Perhaps the vehicles were slow, but extremely well armored. In such a case, a multi-melta would be a better choice than the missile launcher, and a suggested revision of the system. The revision is then possibly retained in the next step of the CBR cycle.

Before we move on to the Retention step, we bring up an important point regarding revision. We assume that the players are playing to the best of their abilities, such that they will only be hindered by probability. While this may seem like a very strict assumption, we only really assume that the players are trying to win using all the knowledge on the board. If the players are unable to play to the fullest capabilities, it is possible and probable that the results of the learning process will suffer.

2.2.4 Retention

Retention is the fourth and final step in the CBR cycle. In this step the system retains, or stores, the revised solution. The solution, however, need not be the only thing stored in the retention step. Some systems store entire derivation processes as well (de Mantaras et al. 2006).

While on the surface Retention seems simple, it is quite the contrary. While the system is able to store every solution, after some time in the life-cycle of the system, the system would get flooded with cases. This would increase the retrieval step time, since there are more cases to search through, but the adaption time would decrease. Once the retrieval time is higher than the adaption time the performance of the system degrades. This is called the *utility problem.*

There are ways to combat this. The simplest way is to simply delete cases. This can, unfortunately, also delete pivotal cases - cases that are vital to the systems knowledge. Smyth and Keane (1995) propose a competence model to value cases before they are deleted. This helps identify the pivotal cases and helps delete cases that do not contribute much to the systems problem solving capabilities.

This kind of competence model is actually present in many games. The traditional example is chess, which was the first to use the Elo rating, named after its inventor Arpad Elo. The Elo rating is a ranking that is given to players based on their skill levels. If two players have the equivalent Elo rating, then the theoretical chance of winning for either player is 50%. The actual rankings differ from game to game, but they follow a derivation of the Elo rating.

Warhammer 40k, as explained in Subsection 2.4.2, also uses a ranking system. If every army class has an attached property of *rating*, which could be a derivation of the Elo rating, we have a competency system. With this competency system we can filter out bad armies from the good armies. Furthermore, we can then extend this system to squads, as an army is a squad of squads. Then, as we reuse armies and squads, we track their performance in battles.

The performance is a simple measure of the value of the squad versus the value of the damage they have done on the battlefield, measured in points used to purchase the units and equipment. In this way, as the squad is used over and over again, it's progress is tracked. If it proves that the rating of this squad is low, the squad is not considered pivotal, and is deleted from the system by the use of maintenance policies discussed in Section 2.4. The formula is presented below.

$$SquadRating = \frac{ValueOfDamageDone}{SquadValue}$$

2.2.5 Knowledge-Intensive CBR

As discussed in Subsection 2.2.1, our domain needs to be enriched with some general domain knowledge. In other words, to fully represent the Warhammer 40k domain, we need to have more knowledge than what the cases can provide alone. Missions, terrain, time of day and alliances all play a key part in the game, but they themselves can not be represented fully within the cases. A domain knowledge is necessary if we want to have anything but unbound armies and flat, empty terrain, with no mission, which is not how the majority of Warhammer 40k games are played.

Aamodt describes that knowledge in a CBR system, and specifically in the Creek system (2004), is difficult to separate from information. Knowledge is then, our interpretation of the information present in the system. In the same sense, our system will be enriched with information that will be used to present the full Warhammer 40k domain.

With a knowledge-intensive CBR system, we will be able to create a system that will completely, or at least as close as possible, match the Warhammer 40k domain, and thus possess little or no assumptions and limitations.

2.3 Explanation-Aware Computing

As mentioned before, a system that can reason about its actions is an intelligent system. As artificial intelligence deals with trying to mimic intelligence, this is a good starting point for designing artificial intelligence. In order to make a system that can reason about its actions, we need to make a system that can explain why it has done something.

2.3.1 Fundamentals of Explanation

Spieker (1991), presents the basic categories of useful kinds of explanations and what are the good qualities of an explanation.

There are five basic categories of explanations:

- Conceptual explanations, which describe the unknown concepts.
- Why explanations, which provide justifications.
- How explanations, which provide deeper explanations of how something works.
- Purpose explanations, which provide the explanation for a goal.
- Cognitive explanations, which explain or predict the behavior of intelligent systems.

Of these, three are important for us: Conceptual, Why and How explanations. The purpose of the system is generally clear: to provide an army that can win against the opposing army. Therefore, providing an explanation is not necessary. Cognitive explanations will not be helpful, as we do not need to predict the behaviour of our system.

Furthermore, there are five good qualities of explanation, as presented by Roth-Berghofer (2004):

- Fidelity - the explanation must be a representation of what the system does.
- Understandability - the explanation must be understandable.
- Sufficiency - the system must know what it is talking about.
- Low construction overhead - explanation must not impose a load on the system, or that load must be light.
- Efficiency - the system must not suffer in performance due to explanations

These five categories must be considered during the design and the implementation of the system in order to reach good quality explanations. However, the last two qualities no longer present a limitation or problem at the time of writing of this project.

The paper presented by Roth-Berghofer, *Explanations and Case-Based Reasoning: Foundational Issues*, was written in 2004. The first commercial dual-core system was released in April/May 2005, by the name of Intel Pentium Extreme Edition 840[4]. Since then we have passed through ten years of development, and quad-core processors are commonplace for commercial use. An explanation can then be executed on a separate core of a system, even on a home desktop, and thus efficiency and low construction overhead are no longer issues that we must be concerned about.

2.3.2 Explanation in CBR Systems

Explanations in CBR have the advantage that the case is always proposed as a solution. This means that we already have a part of the explanation prepare, namely the case itself. We then need to fill in the gap with justification, or the why explanation, and the how explanations. Richter (2003) defines four knowledge containers that can help us with acquiring these explanations. The knowledge containers are presented on Figure 2.3.

[4] `https://en.wikipedia.org/wiki/Pentium_D`

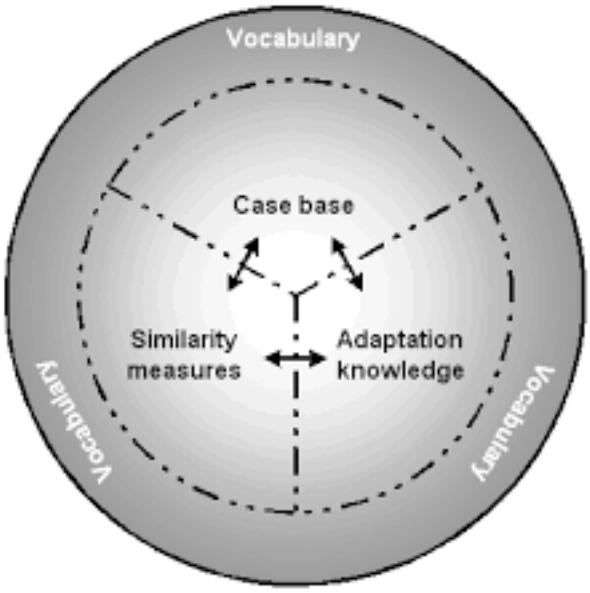

Figure 2.3: Knowledge Containers, adapted from (Richter, 2003)

The four knowledge containers do not only contain information or a definition, but rather a set of knowledge with a purpose. The vocabulary container contains the definition of the system. It contains the other four knowledge containers, as it defines them. This is the conceptual explanation of our domain, or the *what* explanation of our domain. Each of the remaining three knowledge containers interacts with one another, as it would be expected in the CBR system. The similarity measures container contains the similarity methods, and can help us justify why a case has been selected, as well as how it has been selected. Similarly, the rules in the adaption container can help us understand the system, and why it has adapted an old solution the way it has. Finally, the case base container provides context for the explanations. These knowledge containers provide us with a basis for the explanation in our CBR system.

Beyond all of this, the users of the system are the most determining factor when approaching explanation. For novice users, a justification of something is often enough. Saying that *a Space Marine has good shooting accuracy* is usually enough to justify that the Space Marine should be a unit of choice in the upcoming long range mission. However, for more expert users this will simply be an iteration of what they know. Instead, the system should present an option, similar to a *tell me more* option, where the system can present both a justification, and if a user is interested, go into detail about

how and why the system has selected this unit. The same principle is then applied to squads, armies and equipment as well. One of the purposes of explanations will be to teach newer users, and therefore, mixing low and high level explanations is a good way to approach this, and at the same time giving the user the ability to choose when to be presented with the low level explanation gives the user control and lets them learn at their own pace.

2.4 Ever-Changing Environment

There are two final aspects to discuss within the system itself: the possibility of a changing environment and the metagame.

The first edition of Warhammer 40k was released in 1987 under the name Warhammer 40k Rogue Trader. Since then, another six editions have been released every four to five years on average. With every new edition, new rules, units and scenarios were released. With every new edition the game has changed and the best armies may no longer be the best armies, and the worst armies may become a lot better. This is the ever changing environment. For our system, the consistent changes in the domain also mean, like for most humans, the necessity to relearn concepts.

To make our system capable of coping with these changes we need to be able to update the system when changes occur. This increases the flexibility and the longevity of the system. However, it is important to note that the updates have to be very flexible. In many scenarios, players prefer playing older variants of games, especially if the newer variant is not completely balanced. Furthermore, updates to the game can bring big changes in the rules and make for a completely different game. Therefore, a future topic of work for our system would be the addition of legacy support, which will differentiate newer editions and updates from the older ones.

Richter discusses these problems and provides solutions within maintenance policies (Richter and Weber 2013). There are three important aspects to maintaining a knowledge-based system: Corrections, Improvement of Performance and Adaption of changed environments and changed knowledge. All three aspects are equally important to our system.

2.4.1 Maintenance of the Knowledge-Base

At the high level of maintenance we have the changes that appear in the system and the techniques that show us when and how to cope with these changes.

There are two basic categories in maintenance, as described by Richter and Weber: Corrective actions, changes that require an immediate revision, and adaptive actions, changes that slowly accumulate. Corrective actions are to be applied to the system when we need an immediate change, while adaptive actions will consistently be applied to the system, to cope with the metagame, as described in Subsection 2.4.2.

An update to the game system is an immediate change. Therefore, corrective actions need to be applied. These changes are visible and they are on demand. Visible changes means that the changes are a direct result of observations and messages, while on demand changes means that they are a direct observation. This is in contrast to changes that are done to change the metagame, which are invisible and proactive. Proactive reaction is a reaction made before an observation or demand is made. Invisible changes are small changes to the context. Both of these systems are presented in the Figure 2.4. They are of vital importance for maintaining the longevity and the accuracy of the system. The utility problem was discussed in Subsection 2.2.4, and it entails the deletion of non-pivotal cases. Like the update, it is a corrective action, visible and on demand maintenance policy.

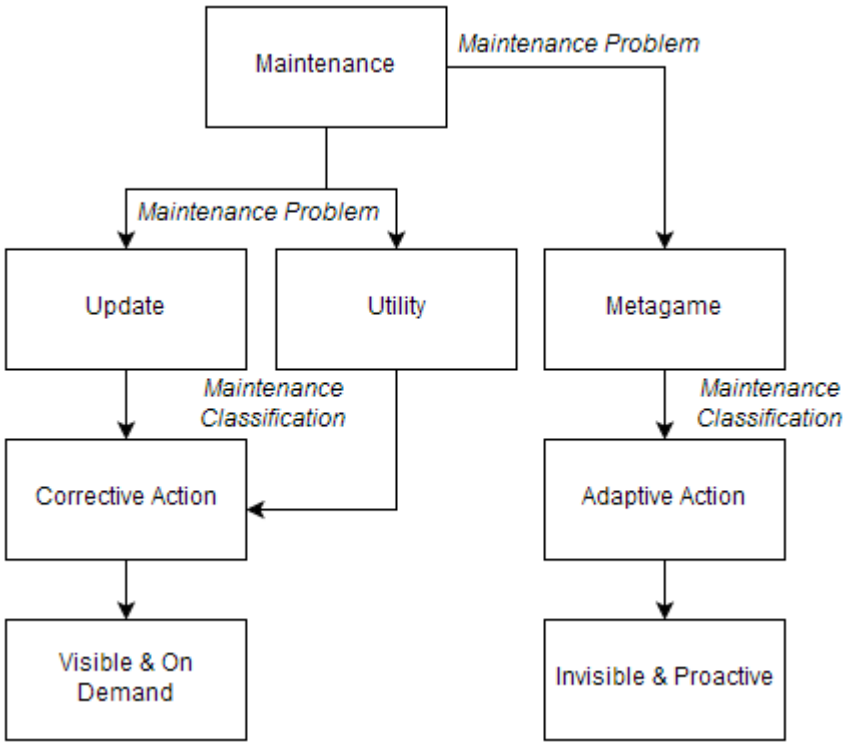

Figure 2.4: Maintenance Policies

The final part of the maintenance is the timing of the maintenance, or when to perform the maintenance. In the case of the update to the system, the maintenance is performed ad-hoc, meaning there is no systematic maintenance. We simply perform it whenever the update to the system happens. For the metagame, the maintenance can either be applied periodically or per event, such as a new addition to the case base. Periodic maintenance would put less of a strain on the system, and would also be able to maintenance when necessary. On the other hand, extensive experience and knowledge is required for setting the interval of maintenance. Event-based maintenance is more expensive, but does not require previous experience, and it may prove safer than periodic maintenance.

2.4.2 Metagaming

Even though the editions may change every four to five years, the game continues to change even between editions. Players keep creating and testing new army combinations, in the hopes of bettering themselves and their understanding of the game, but most of all their chances of winning. The notion of best strategy is called *meta*. Metagaming or the metagame in its essence is predicting what the opponent will do and then making a counter-play or counter-move towards the opponent. In a broad sense, the metagame is not a part of the game itself, but rather the tactics and strategies thought outside of the game, within the scope of the rules of course, and then applied to the game. Metagaming works differently depending on the game system or domain we apply it to.

To illustrate this, let us take a very simple example of tic-tac-toe. Tic-tac-toe[5] is a simple game that is played on a 3x3 square board, where the first player places an X in a square, the second places an O and then they alternate until one player has three in a row, be it diagonal, horizontal or vertical, or the game is tied. A typical game is shown in Figure 2.5.

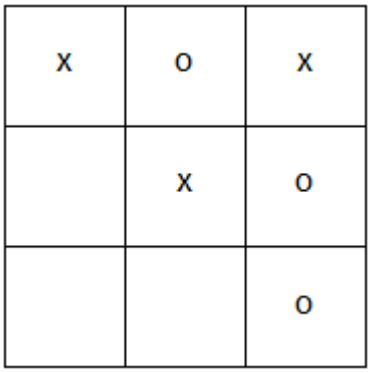

Figure 2.5: Tic-tac-toe, in this case the X player has won the game

Tic-tac-toe is very simple, however. In fact there is a perfect answer to tic-tac-toe and it is small enough to be placed on one page[6]. When a game is complete, it is no longer a problem set. Since we know the perfect outcome for every single move that can be made in the game we do not get any notion of metagame. Every strategy is already known, and we need only follow it.

[5]https://en.wikipedia.org/wiki/Tic-tac-toe
[6]https://xkcd.com/832/

We define a complete game as a game where if a player plays the best strategy, they will always win or draw, unless the game is such that a win or a draw is impossible. As we have said, the meta is a notion of best strategy, and in this case the meta is just to play the best move. However, there is no metagame, as all of the moves are already known. Therefore, we can state that any game that is complete has a meta, but loses the metagame.

Before we move on to the discussion of more complex games, we need to first define two important subtypes of games. Perfect-information games, or games where you can see your opponents moves, and partial-information games, where you can not see your opponents moves. In perfect-information games, like tic-tac-toe you can see what your opponent does, and thus you can react accordingly. Furthermore, you have the full knowledge of all of the opponents resources, for example chess pieces in chess. In partial-information games you can not see the opponents "hand", though you may be able to see what the opponent does. In these games you do not have the full knowledge of the opponents resources. This is fairly typical of card games, like poker or blackjack.

The difference is quite powerful, and leads to different strategies. In perfect-information games the metagame often focuses on making the best move based on the opponents move. Although by no means easy, these types of games can be solved given enough computing power and time, by brute force. Partial-information games can be more difficult to solve, and most of the times involve chance, and the metagame therefore revolves around increasing the statistical probability of winning. Players have incomplete knowledge of the game; either they can not see a players hand (like in poker), or they can not see the opponents moves (like in mahjong[7]), but can theorize about the strategy of the opponent.

Now we move onto the discussion of more complex games. As mentioned before, perfect-information games can be solved, eliminating the need for having a metagame. Once a game is solved, there is little point in playing the game, assuming the players know the solution. For complex games like chess a solution has still not been and may never be completed. Therefore, one can still metagame about strategies during the game. Predicting what the opponent will do next, either one or many turns ahead, is metagaming, as it is playing the game outside of the actual board. In most perfect-information games this is the only type of metagaming as the game is symmetrical and balanced, save for the first opening move.

[7]https://en.wikipedia.org/wiki/Mahjong

In partial-information games on the other hand, one can only theorize about the opponents moves. Since not all knowledge is present or visible, metagaming here involves predicting what the opponent has, either through experience or through what is present on the playing field. If the game is still absolutely balanced, as for example poker is, then one can draw statistical information about it. With this information, a player can calculate a probability and decide, based on experience, if the risk outweighs the cost of the play.

However, there are partial-information games that are not balanced, or perfect-information games that are not balanced either, at least not through the entire domain. In computer fighting games, like street fighter[8], a large part of the metagame revolves around characters that are good at beating other characters, or in a simplified variant, around rock-paper-scissors. At high levels of play the metagame, or predicting what the opponent picks, usually gives a player the competitive edge. However, in many of these games there are characters or units that are so strong, that they are the *meta*. In other words, if we were to take these characters as rock, they would still beat paper most of the time in their games. Many research hours from video gamers go into metagaming, by making lists of character strengths. In another popular fighting game, super smash bros (specifically for the Wii U console), an entire list is created ranking the characters by tiers[9]. One could say that the first character, or the first two-three characters on the list are the *meta* and until the game changes in some way, they will be picked more often.

Quite like in fighting games, in between the editions of Warhammer 40k there is a *meta*. There is an army that outperforms most other armies[10]. However, the more complex the game becomes, the harder it is to find the true *meta*, and that is most definitely the case in Warhammer 40k. Unlike in fighting games, there are a lot more variables in Warhammer 40k, and thus the meta can be broken by an army that can counter the best army, though the counter itself might not be a good army. Therefore, we can state that the metagame of Warhammer 40k is constantly moving and constantly shifting. The aim of our system will be, to no lesser degree, to try to make a system that can continuously adapt its solutions to keep up with the trend of the changing metagame.

[8]http://streetfighter.com/

[9]http://www.eventhubs.com/tiers/ssb4/

[10]http://www.torrentoffire.com/5612/7th-edition-three-months-in

The concept of the metagame is quite important in Warhammer 40k and in many other games. It should not be viewed as a concept limited to games, however. Even when there is no direct upgrade in a domain or a system, assuming the system is not perfectly solved, the system should continuously and proactively work to reach a better solution, and not be satisfied with the solution it has reached. Based on the knowledge-base of the system, the system should explore different avenues and compare the solutions with the top solution, or against it. If the system does this, then it continuously betters the domain that it works in. Of course, there are limitations to this, and the system would need to be diversified so that it does not reach the same solution multiple times, but diversification has proven to be a strong avenue of research in CBR (Smyth and McClave 2001), and with the help and research of users this is a possibility.

To achieve this proactive metagame capable system, maintenance policies can be applied. While maintenance policies normally deal with maintaining the case base itself, we can apply them to the system periodically as new cases. What is meant by this is that the maintenance policy will periodically create a random, legal squad, and then run the system with this squad as the problem against what the cases present in the system.

This can then be repeated over and over again, monitored by experts, but without intervention. The system is attempting to fight these squads against each other, using fair dice rolls, and see which one comes victorious multiple times. Furthermore, the system can use different units and thus from one active use of the system to the next, the system has passively already been *thinking and learning* about the domain, much like a regular player would. This would then fulfill the final sub goal of this project.

2.5 State of the Art

In this section we will present the Structured Literature Review (SLR) protocol and the non-personal motivation for the project. Combined, the SLR and the motivation represent the state of the art of the system.

2.5.1 Structured Literature Review Protocol

To acquire the literature necessary to study this project, we followed the Structured Literature Review (SLR) as presented in *How to do a Structured Literature Review in computer science* by Kofod-Petersen (2014).

The first two phases, the identification of the need for a review and the commissioning of a review were not necessary, as they were already in place at the start of writing the project. In the third step, we have specified four research questions for the SLR:

- Research Question 1 - What solutions exist to tackle our problem? That is, designing an explanation-aware CBR system for creating armies in Warhammer 40k.
- Research Question 2 - How do the solutions found in research question 1 compare to each other, in terms of our vision to create this project?
- Research Question 3 - What is the strength of the evidence in support of different solutions found in research question 1?
- Research Question 4 - What is the implication for designing our system based on these solutions?

Following these questions, we have set out to find papers on explanation-aware computing, case-base reasoning, especially knowledge intensive case-based reasoning, maintenance policies, and any relations to Warhammer 40k.

To find the solutions, and be able to answer the other research questions, we have searched two distinct sets of papers. The first set was papers obtained from the seminars attended with Anders Kofod-Petersen and Agnar Aamodt, who are experts in the field of AI. The second set was the papers obtained from the Internet directly, without the help of an expert.

The papers obtained from the experts were obtained during the course of the project. The experts were trusted to make sound judgements on their choices of papers. Furthermore, these papers were discussed by fellow peers and the experts. This meant that most of the drawbacks of the papers were presented, and it was a choice of whether or not to use them. The experts helped us obtain papers that were highly relevant to our fields of study, and these were taken as a primary source for both knowledge and the state of the art. The list of papers can be seen in Table 2.4, which also includes one book.

Paper	Author	Year
The Omnipresence of case-based reasoning in science and application	Aha	1998
Retrieval, reuse, revision, and retention in case-based reasoning	de Mantaras et al.	2006
Book: Case-Based Reasoning	Richter and Weber	2013
Explanations and Case-Based Reasoning: Foundational Issues	Roth-Berghofer	2004
Explanation in Case-Based Reasoning-Perspectives and Goals	Sormo et al.	2005
Evaluating the effectiveness of explanations for recommender systems	Tintarev and Masthoff	2012
Learning from explanations in recommender systems	Cleger et al.	2014
Tagsplanations: Explaining Recommendations Using Tags	Vig et al.	2009

Table 2.4: Papers obtained from Experts

Furthermore, we include seven additional papers acquired from the papers mentioned in Table 2.4. The seven papers are relevant to the project and are assumed to have already undergone a structured literature review, and hence will not be scrutinized again. They are presented in Table 2.5.

Paper	Author	Year
Dynamic Memory: A Theory of Reminding and Learning in Computers and People	Schank	1982
Explanation Patterns: Understanding Mechanically and Creatively	Schank	1986
Explanation in expert systems: A survey	Moore et al.	1988
Similarity and Compromise	McSherry	2003
Case-based reasoning, foundational issues, methodological variations, and system approaches	Aamodt and Plaza	1994
Remembering to forget: A competence-preserving case deletion policy for case-based reasoning systems	Smyth and Keane	1995
Similarity vs. diversity	Smyth and McClave	2001

Table 2.5: Papers obtained from Papers in Table 2.4

For papers obtained outside of these seminars, the key groups of terms that were looked for were: Case-Base Reasoning, Explanation Aware, Knowledge Intensive, Proactive and Warhammer 40k. The search engine used was Google Scholar. The search-terms are presented in groups in Table 2.6.

Term	Group 1	Group 2	Group 3	Group 4
Term 1	Case-based reasoning	Explanation aware computing	Proactive case based reasoning	Case based reasoning Warhammer 40k
Term 2	Knowledge Intensive Case Based Reasoning	Explanation aware case based reasoning systems	Maintenance in case based reasoning systems	
Term 3	Architecture in Case Based Reasoning		Self-Updating case based reasoning	

Table 2.6: Search terms sorted in Groups

After searching for these terms in Google Scholar (the first three pages were taken as the most relevant) and Google Search Engine (The first page taken as most relevant), 102 papers were obtained.

From these 102 papers duplicates were removed, both in relation between different search terms, and with the papers already presented in Tables 2.4 and 2.5. Furthermore, only the most recent publication of the same study was kept. With this, 49 papers were removed from the aforementioned 102, and 53 were left to scrutinize.

The 53 papers were then scrutinized, and if they passed a quality check they would be used. The papers were read abstract first, then if proved relevant the evaluation was read. Finally, if the paper was deemed plausible a full quality check was performed. Weight was placed on discussion, evaluation, methods, clear research goals and explanations of the papers, and hence most workshop papers were not considered, due to the lack of such concise information. From the 53 papers, 15 were kept, and are presented in Table 2.8 while the inclusion and quality criteria can be found below in Table 2.7. The inclusion and quality criteria were applied to all the papers, including those in Tables 2.4 and 2.5.

Criteria Identification	Criteria
IC 1	The study in the paper or book is concerned with either CBR, explanation-aware computing, maintenance, or a combination of thereof
IC 2	The study has either a good theoretical background for IC1 terms, or good empirical results for IC1 terms, or both
QC 1	The aim and results are clearly presented in the abstract and paper
QC 2	The results are discussed in the evaluation with respects to the aim of the paper
QC 3	At the time of writing of the paper the paper has presented previous work relevant to the aim of the paper

Table 2.7: Inclusion and Quality Criteria

Paper	Author	Year
Towards lifetime maintenance of case base indexes for continual case based reasoning	Zhang et al.	1998
Categorizing case-base maintenance: Dimensions and directions	Leake et al.	1998
Six Steps in Case-Based Reasoning: Towards a maintenance methodology for case-based reasoning systems	Berghofer et al.	2001
Mapping Goals and Kinds of Explanations to the Knowledge Containers of Case-Based Reasoning Systems	Roth-Berghofer	2005
An Evaluation of the Usefulness of Case-Based Explanations	Cunningham et al.	2003
An Architecture for Knowledge Intensive CBR Systems	Diaz-Agudo et al.	2000
Knowledge-Intensive Case-Based Reasoning and Intelligent Tutoring	Aamodt	1994
Different roles and mutual dependencies of data, information, and knowledge — An AI perspective on their integration	Aamodt et al.	1995
A case-based reasoning with the feature weights derived by analytic hierarchy process for bankruptcy prediction	Park et al.	2002
Case-based reasoning is a methodology not a technology	Watson	1999
Knowledge-Intensive Case-Based Reasoning in CREEK	Aamodt	2004
Explanation-driven case-based reasoning	Aamodt	1994
Knowledge-Intensive Case-Based Reasoning and Sustained Learning	Aamodt	1990
Case-Based Reasoning in a System Architecture for Intelligent Fish Farming	Tidemann et al.	2012
Myrmidia The Warhammer Fantasy Battle Army Builder	Strandbraten	2011

Table 2.8: Papers obtained from online Search

As a final note, websites were considered as helping knowledge and are included in the footnotes throughout the report. where relevant. These websites are not used as sources or are related to the state of the art as such; these websites are additional information that the reader may pursue should he or she wish to do so. Unfortunately, the information presented in these websites is unavailable offline, though it, as stated before, only plays an informative role in the project.

2.5.2 Motivation

The creation of the system is in itself motivation. As of now, no system has been created to tackle the Warhammer 40k domain. There has been previous work in a similar domain, the Warhammer Fantasy domain, but the domains are significantly different for the creation of the Warhammer 40k system to be novel. Especially with the year 2015, where Warhammer Fantasy Battle has been replaced with a much more simplified Warhammer: Age of Sigmar. To illustrate this point, the Warhammer Fantasy Battle had a rulebook of over 200 pages, whereas the Age of Sigmar has a rulebook of 4 pages.

Nonetheless, the work done in Warhammer Fantasy Battle was done by a previous master student at NTNU, and the system relating to that is called Myrmidia (Strandbråten 2011). No papers were found relating Warhammer 40k and case based reasoning or explanation-aware computing.

The creation of this system will enable novice users to learn Warhammer 40k easier, veteran users to get tips and metagame analysis, and in general, it will create a learning AI. Therefore, in just creating the system the sub goals and the main goal are fulfilled, and if anything, the system has potential of teaching and assisting within the Warhammer 40k domain.

However, the practical application of the problem itself far surpasses the Warhammer 40k domain. There is a large potential of application in video games, as previously mentioned, to assist computer players in making decisions. Decisions are typically created with if/else actions, and eventually with some basic weights and sliders, or simply hard-coded into the computer player, so that it always plays in a similar fashion (or a few similar fashions) without much diversity. With the application of this system to strategy video games, we can see an emergence of strong computer players that can make complex decisions. It may be possible to reduce or even eliminate the (sometimes massive) bonuses that computer players often get.

Furthermore, there are practical applications in non-entertainment related sectors as well. Industry, for one, can benefit from this kind of decision system. A system that can, given information, produce the best way to divide resources, can potentially save costs on material or simply assist constructors and architects in early planning phases. In medicine as well, the system can aid in emergencies, where doctors have to respond to a numerous amount of patients, by classifying the importance of emergency and the capabilities of doctors on hand.

In general any situation where a limited set of resources has to be divided in order to tackle a complex problem, this solution can be applied to. Furthermore, the system is designed, so that those problems can change over time at no expense of the system itself. Finally, due to explanations the system can be used as both a helping tool and as a stand alone system, thus not necessarily taking over jobs while still being capable of aiding and improving work.

According to Schank (1986) explanations need to be both *inclusive* and *instructive*. Many explanation aware systems focus on building trust and confidence for the user, as they are often used in recommender systems. These explanations are either used to help the user build trust in the system, through various means (Tintarev and Masthoff 2012, Vig et al. 2009) or used in attempting to improve such systems (Cleger et al. 2014).

Instructive explanations can also be interpreted as teaching. The explanations are fairly high level in these recommender systems. In the Warhammer 40k domain, I will attempt to create an explanation-aware system that can not only build confidence in the user, but also instruct them and teach them about the domain itself. This is a core concept of explanations, and should not be overlooked, especially in a domain and system that can facilitate such explanations (Sormo et al. 2005).

Maintenance is another point of motivation for this project. Ramon Lopez de Mantaras et al. (2006) points to a number of different uses for maintenance that have been developed over time. All of them have to do with either case deletion/case base maintenance, case inconsistencies or case acquisition. While some of these approaches will be used to maintain the case base in this system as well (Section 2.4) an important use of motivation is simulation.

Unlike the case acquisition solutions, this maintenance process actively creates new smaller subsets of problems following legal rules for creating squads. This in turn consistently queries the system and the system is able to learn and *think* for itself. Naturally, as with all exploratory work, this process needs to be supervised by an expert or a team of experts.

If successful, this system could be further expanded to create full simulations, and we will potentially have a system that can teach itself, following a set of rules and doing so without interference, or in other words, teaching itself proactively. The focus in this project would be to evolve the metagame with this particular technique.

Finally, when it comes to the core knowledge-intensive CBR system, the system takes inspiration from Creek (Aamodt 2004), with further implications that the techniques and methods mentioned above will be integrated into the system. Therefore, while the basic CBR system is not re-invented, the additions will significantly change how it operates.

Chapter 3

Architecture

Software architecture is the highest level representation of a specific software or system, that describes how the software functions without going into any design details. Furthermore, the relations in between the different system substructures and structures are captured, as well as the functional and non-functional requirements of the system.

There are two main goals with the architecture of a system. The first is to give an overview of the system as a whole, so that the reader may read this chapter without going into the design details, or read specific design details that interest them. The second goal is to create a template that can then be reused in future work.

Section 3.1 will present the stakeholders of the system and their roles. Section 3.2 will present the functional and non-functional requirements of the system. Section 3.3 will present the architectural template and the instance of the class of problems.

3.1 Stakeholders

The stakeholders of the system are people that have some sort of interaction with the system, be it direct interaction with the system (as is the case with a user), or a complete interaction and creation of the system (as is the case with the developer of the system). Table 3.1 shows the most important stakeholders for this project and their roles.

Stakeholder	Role
End User	The end user is any user who uses the front end of the system, in other words the user that uses the system to help him or her create an army
Developer	The developer is the author of this project and the creator of the system
Supervisor	The supervisor oversees the creation of the project and system and provides guidance
Games Workshop	Is the owner of the Warhammer 40k domain

Table 3.1: Stakeholders

Stakeholders are an important part of the system, as they can directly influence how a system is going to be designed. Therefore, it is important to know the stakeholders before beginning the design phase of the system.

The end user is a person who uses the system to either learn or get help with building an army. They determine largely how the explanations of the system are handled, and which audience to target as described in Section 2.3.

The developer is the author of the project. The success of the project in its entirety is important to the developer, both as their performance on the project (the grade) as well as the validity of the system for possible future research.

The supervisor oversees the construction of the project in all forms and offers guidance when necessary. As a scientist in computer science, artificial intelligence and especially explanation-aware computing, it is important to the supervisor that this project provides results.

Games Workshop is the owner of the Warhammer 40k domain. All of the domain belongs to them, and without their approval the project can only be published with regards to the improvement of artificial intelligence and sciences, and in no way used for any commercial or other distributive purposes.

3.2 Requirements

Functional requirements of a system indicate the tasks that the system needs to perform. In other words, they are the requirements for our system in order for it to function according to our needs. Non-functional requirements are qualities that do not directly influence the behaviour of the system, but are qualities that the system should possess in order for it to perform well. Non-functional requirements can differ largely from system to system, and are quite dependant on the end user.

Both the functional and non-functional requirements have an important secondary purpose. Once the design and the implementation stage of a system is finished, they are used to create tests and the test plan. If the tests are created with the purpose of testing each functional and non-functional requirement, then we know that once the testing is complete the system will work the way it was intended to work as detailed in the requirements.

3.2.1 Functional Requirements

A goal-driven approach was used to gather functional requirements of the system. This means that the goals and sub-goals presented in Section 1.2 were used to create the functional requirements. This was then further expanded on by the theory as presented in Chapter 2, and the requirements that were finally created can be seen in the Table 3.2.

Number	Name	Description
1.1	Retrieve	The system needs to be able to retrieve the best-match case from the case base
1.2	Reuse	The system needs to be able to present this solution to the user, and adapt it to the new problem
1.3	Revise	The system needs to be able to revise the solution if it did not prove useful
1.4	Retain	The system needs to only retain a solution that passes a certain quality metric
1.5	General Knowledge	The system needs to be able to store and use general knowledge in conjunction with the cases to match the complexity of the domain
2.1	Explanation: Basics	The system needs to be able to explain and justify its choices
2.2	Explanation: Presentation	The system should present the explanation based on the user knowledge of the domain and the user need of the explanation
3.1	Maintenance: Utility	The system needs to have maintenance policies to avoid the utility problem.
3.2	Maintenance: Consistency	The system needs to be able to maintain itself when given updates to the Warhammer 40k domain
3.3	Maintenance: Metagame	The system needs to be able to utilize maintenance policies to consistently improve its case base.

Table 3.2: Functional Requirements

Table 3.2 shows the number, name and description of each functional requirement. The number is used as reference for test creation and test planning. The name is the reference to the functional requirement, and it is a short way of describing it. The description of the functional requirement is the actual requirement itself. Note that the functional requirements are split into three distinct categories, as indicated by the numbering (1.x, 2.x and 3.x). This represents the relation of the functional requirements to each of the sub goals, presented in Section 1.2.

3.2.2 Non-Functional Requirements

As mentioned, non-functional requirements, sometimes called quality attributes, can differ largely from system to system. However, representing non-functional requirements is still a difficult task, and their classification is a subject of research, but it is agreed that they are important (Glinz 2007). The task of classifying non-functional requirements is beyond the scope of this project, but as they are necessary the *main* six requirements that are featured in almost any software-related project will be presented here as well.

Name	Description
Reliability	The system will function within a prescribed interval of time
Usability	The ease of use of the system by the user
Security	The security of the system, or the unavailability to change the system from any unauthorized source
Availability	The readiness to use a system at any given random time
Portability	The possibility of the system working in several different environments, or platforms
Maintainability	The ease of which the system can be maintained

Table 3.3: Non-Functional Requirements

Table 3.3 shows the six most important non-functional requirements. From it, we can discuss the non-functional requirements for this particular system.

Reliability is an important factor in our system. It needs to continue working through a Warhammer 40k game at the very least, which can take a few hours. Furthermore, if we are to follow our own maintenance policies, as presented in Section 2.4, we need the system to run for at least a few days, and we need it not to crash at any point of time, especially while doing deletion or addition to the case base. From this, we can say that the system should have a mean average error time of one week or longer.

Usability is largely covered by the explanations functional requirements. In essence, the explanations have to be good enough to enable both novice and expert users to use and learn from the system. This is a fuzzy requirement to consider, and usability must be a subject of scrutinization from the users and the developer in order to fulfill it. Furthermore, we need to make sure that the user can use the system easily, and that the user does not need to learn pages of instructions in order to use the system.

Security is very important to the system. Under no circumstance should the case base, the system itself, or any of its parts, be prone to editing or changing from any source but the developer. This means that the actual system end should be protected, and that the source code and the functionality should reflect this with password protection and good practice.

The system needs to be available at the time the user desires to use it. In other words, the system needs to function when a user wants it to function, so the availability has to be as high as possible (close to 100%).

The system need not be portable to other environments or platforms, and there is no need for such actions in this stage of the project.

While maintenance is largely covered by functional requirements, the ease of maintenance is represented in maintainability. As it is a large part of the project, maintainability is very important to the system, and the system needs to be near-automatic when it comes to maintenance, mostly with observations and small changes from the developing team.

3.3 Architectural Template

Architectural templates are often used in software engineering so that they may be reused and that it may be easy to adopt the system to other problems. However, in complex or newer systems, like this project, it is very difficult to find a completely matching architectural pattern. Rather than force the system into a pattern, we will create a pattern that other systems can follow and base themselves on. This is represented in Figure 3.1. The arrows in the figure show which parts of the system interact with each other.

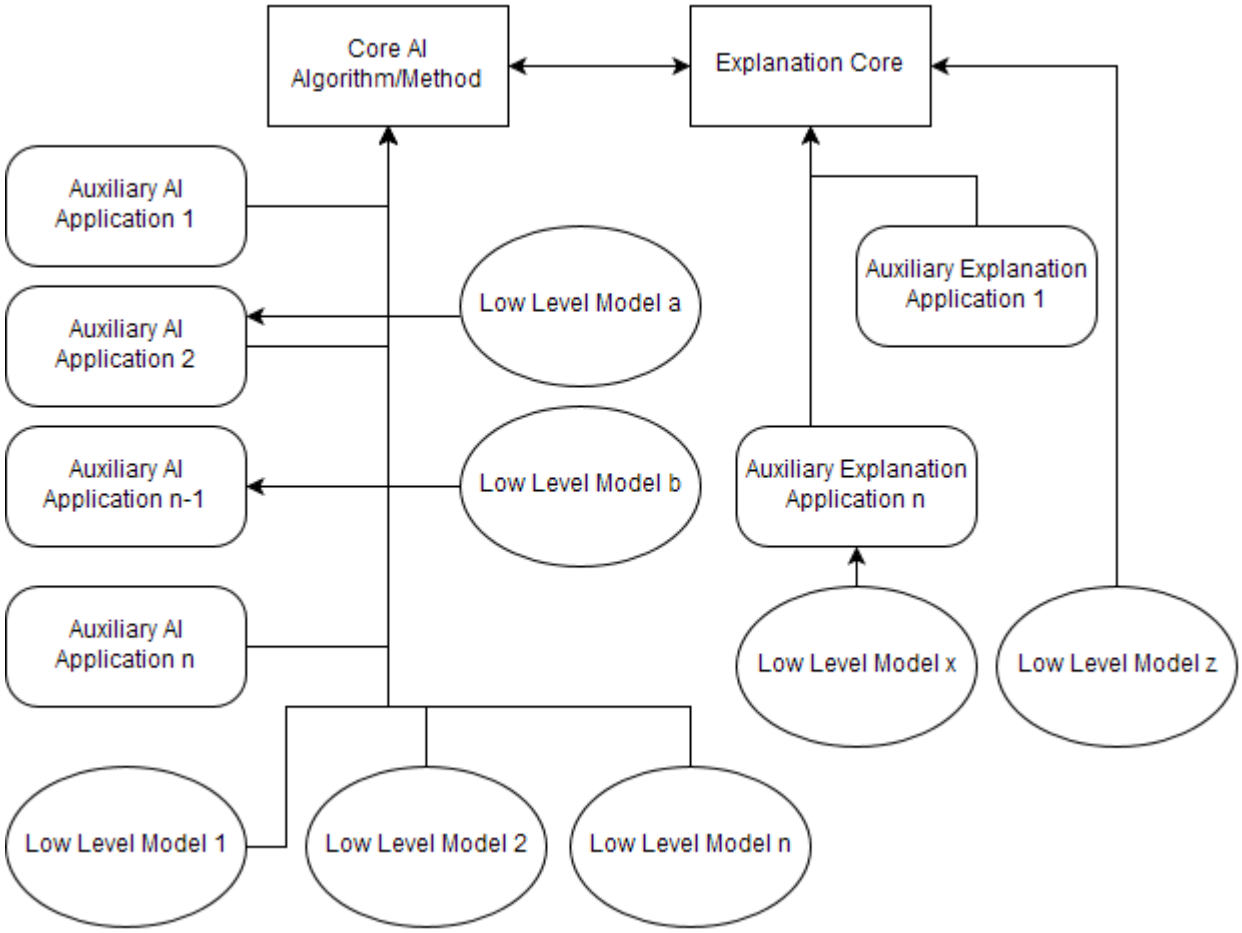

Figure 3.1: Architectural Representation of the Class of Systems

In the figure we can see the most basic layout of the system, loosely adapted from the techniques discussed *Case-Based Reasoning in a System Architecture for Intelligent Fish Farming* (Tidemann et al. 2011).

At the very top we have the Core AI Algorithm/Method and the Explanation Core. These two are mandatory for any explanation-aware system. The Core AI Algorithm/Method defines any method or algorithm that the engineer would like to implement. It exchanges information with the Explanation Core, which is the part of the system responsible for explanations. Note that the explanation core could potentially be integrated in the Core AI algorithm, but since that is not the current practice, it is separated. The explanation core has access to, in general, what the AI method or algorithm provides.

Next we have Auxiliary applications. These applications do not necessarily have to be full applications or programs. They can be simple or complex tools that aid the Core AI and Explanation systems to perform better. Again, some of these systems can be integrated with the core, but that will depend on the cores themselves. It is up to the developer to choose how many (or if any) auxiliary applications he or she may want to add to the cores.

Finally we have the low-level models, which can both interact with the applications or the cores. These represent models and perhaps databases for auxiliary applications, or even some domain knowledge. Much like the auxiliery applications, the developer has the choice to add the models to the core.

Below, in Figure 3.2, the instance of our particular system is shown, based on the architectural representation shown in Figure 3.1.

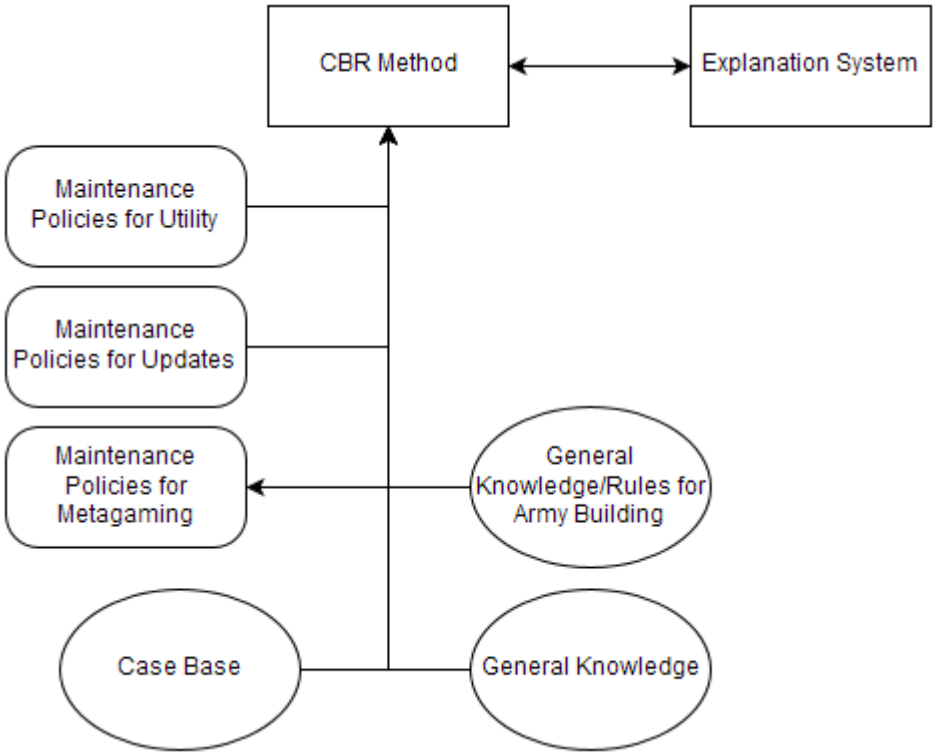

Figure 3.2: Architectural Representation of the Project

Figure 3.2 represents the architecture of this project. The CBR Method is the Core AI Method, and the Explanation System draws information from the knowledge containers, as described in Section 2.3.

The three auxiliary applications are the maintenance policies for utility, updates and metagaming and metagame improvement. The CBR System has the general knowledge and case base as its own models, and the maintenance policies for metagaming has its own general knowledge and rules for army building.

Chapter 4

Design

In this chapter we will discuss the system design in detail. Each component that was introduced in Section 3.3 will be discussed individually, as well as how it combines with the rest of the system. At the end of this section a simple prototype of the system in the form of a discussion will be presented, to give the reader a general idea of how the system should look like and behave when the system is finished.

4.1 Case Based Reasoning

The CBR part of the system is divided into case representation, the CBR cycle and the general knowledge, similarly to how it is presented in chapter 2. The CBR part is designed to be stand alone. This means that it does not rely on any other parts of the system to function, and it should be able to perform its function as indicated in the sub-goal 1, in Section 1.2.

4.1.1 Case Representation and Case Base

The Case Representation needs to be clearly defined and designed before we move onto the four steps of CBR. In Subsection 2.2.1 we have identified four representations that are crucial for the system: the unit object, the equipment object, the squad object and the army class.

The unit object is any one unit in Warhammer 40k. A unit will be represented as an object, with various attributes attached to it.

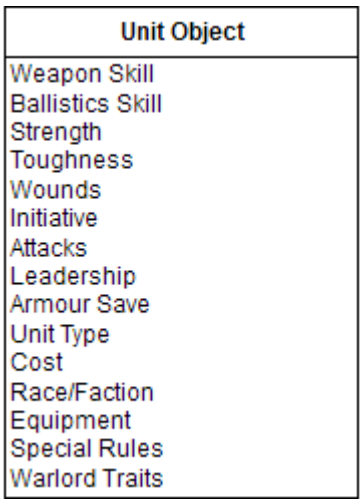

Figure 4.1: The Unit Object

Figure 4.1 represents the unit object. The first nine attributes, weapon skill to armour save, were discussed in Section 2.1 and will not be repeated. They are all integer values. The tenth attribute, unit type, was also discussed. However, it is important to mention that some unit types get special rules that modify some attributes of a unit. In the case of unit types the modifications are automatically applied to the attributes of the unit. The unit type attribute is a string.

Furthermore, we have the cost of a unit, which represents the cost of the unit in points, which is an integer value. Then the Race/Faction attribute, which represents the race or faction that the unit belongs to. This is represented as a string value. Then the Equipment, which is another object entirely. A unit can possess several pieces of equipment, so it is vital that this attribute of a unit is represented as an array of equipment objects.

Finally the two last attributes of a unit object are the Special Rules and Warlord Traits. Both of these vary substantially from unit to unit, and the majority of the units do not possess any Warlord Traits[1], while possessing one or more Special Rules. As such these attributes are presented with an array of strings to designate the Special Rules and Warlord Traits, and these are contained within the general domain knowledge of the CBR system.

The Equipment object is represented below, in Figure 4.2

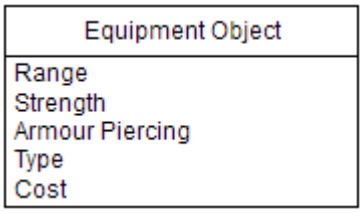

Figure 4.2: The Equipment Object

The range of a piece of equipment is represented in inches, and is an integer number. It can be represented as a - in the rulebooks, which just means that the equipment has no range, and therefore can be represented with a 0 in the object representation. The strength of the weapon is the strength with which it attacks. The strength can range from 1 to 10, however, a weapon can also use the users strength, or do some other modifications, such as multiply the strength by 2. Therefore, strength must be represented as a string or as a combination of integers and strings.

The armour piercing attribute is an indication of how good the weapon is at penetrating through armor. Like armour save in the unit object, the lower the number the better it is. The type of the weapon is a similar attribute to the special rules of the unit, combined with the unit type. It is represented as an array of strings, since a weapon can have multiple types. The weapon type rules are contained within the general domain knowledge. Finally the cost is represented as a integer, much like the cost is for the unit object.

The squad object is a collection of unit objects. As previously mentioned, some units may be grouped into squads, either because they are too weak individually, because of lore reasons, or because they simply perform better as a squad. A squad object therefore, just inherits an array of unit objects.

[1]Typically only the units with the unit type that equals character possess Warlord Traits.

A squad object also has a cost, which is inclusive of all the units. In other words, the squad costs as much as the unit objects in it cost together, and is represented as an integer. Some units do not have an individual cost, but must come in a squad. Due to this the squad cost superceedes the individual unit costs when doing any calculations. A squad also has special rules, much like a unit object does.

Squads can also have options. These options range from adding more units, to adding or changing equipment of those units. Knowing these options is crucial for the system as they are used in case adaption in the reuse stage of the CBR cycle. This is represented as a string of arrays, of which the rules are again contained in the general domain knowledge.

Finally squads have a rating. This is something that is not normally present in the Warhammer 40k domain, but is included in this system. The rating is the notion of how well a squad performs. The ratings were discussed in Section 2.4 and will be used to denote the performance of a squad[2]. The squad object can be seen in Figure 4.3. A squad can only be one unit, in which case the cost and special rules are inherited from the unit object, and the options are none, since the squad inherently does not exist in Warhammer 40k, but it is an easier representation of a single unit in the system.

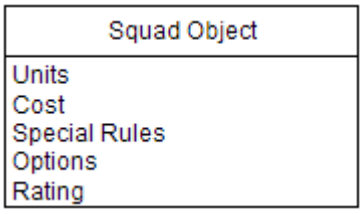

Figure 4.3: The Squad Object

The army class is a collection of squads with a rating and a cost. Much like the cost of the squad, the army cost attribute is inclusive of all the squads inside the army. The rating is again, a representation of how well the squad does in the game. The rating is only present in squads and armies, and this is primarily due to the fact that units and equipment will not be retrieved per object basis, but rather as a part of squads and armies. In the case of one unit squads, the unit is usually so powerful that it can match squads in power, and therefore does not deteriorate the system. The army class is represented in Figure 4.4.

[2]More information on the rating can be found in Apendix C

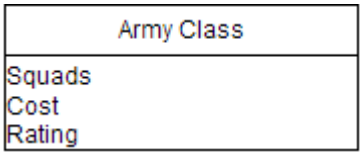

Figure 4.4: The Army Class

The case base will contain all of these objects, but the retrieval part will only focus on retrieving armies. The reuse and revision parts will focus on using the squad, unit and equipment objects to adapt and revise cases accordingly.

The case base will initially be filled with two or three armies, at least twelve or more squads, and all of the units and equipment for the Space Marines faction. This should provide a good learning simulation, as most of the players that start playing Warhammer 40k pick one race and usually a previously constructed army described in the rule books. Then, this case base will be expanded on within the life cycle of the CBR system, by the addition of new units and equipment from other races/factions, new squads and armies, and their respective ratings.

4.1.2 Retrieval

Retrieving the cases from the case base will be a two step process, as mentioned in Subsection 2.2.2.

The first step to retrieval will use a very simple k-NN algorithm. The cases retrieved will be the armies with the highest rating that match the required cost. Nothing else is retrieved from the army class. The rating will have a maximum value, represented by 1 for similarity, and a minimum value, represented by 0 for similarity. For the cost, same or lower costs are acceptable, with similarity declining linearly, but rapidly, for lower costs, and similarity nullified for higher costs, as it would be an illegal army.

This means that the first step of retrieval will retrieve the k number of armies with the highest rating and the required cost as indicated by the user.

This can, and will, be changed by the compromise driven retrieval. As mentioned in McSherrys *Similarity and Compromise* (2003), a most similar case need not be the best case. In our system this also holds true. While the top k rated armies may be good armies, if a player wishes to play with another race or army, the option should be available to them.

Therefore, the retrieval is modified to include compromise based on the race or faction the player wishes to play. If the player has no particular wishes, the k armies are returned without any change. If the player, however, desires a specific race or faction, the system retrieves the armies and then checks the squads attribute, and subsequently the unit object, to see which race the army is consistent of. It is important to note that the armies may consist of multiple races or factions, so each squad needs to be checked, and the compromise system has to ask the player if they are in agreement with playing with those multiple races or factions. The armies that have the same race or faction have a similarity index of 1, where as the others have 0. Then a k set of those armies is retrieved, with same parameters as before, regarding the costs and ratings.

Once these armies are retrieved we will have a much smaller subset of armies, or cases, to work with. Based on performance the subset number, or k, will be adjusted. The bigger the k the better the likelihood of finding more similar armies to our query, as the best armies do not necessarily have to be able to beat, and usually are not able to beat, every army. The subset of armies will then undergo a structural similarity search by the use of a structure-mapping engine.

The structure-mapping engine is a robust algorithm that uses both the general domain knowledge and the army classes to produce the best answer. It compares the ratings of the individual squads on both teams and their main purpose, then looks through the armies to see which army would be best suited to combat the problem army. The rating of an army is an influential factor in this, but not the only deciding factor. The statistics and capabilities of the units in either army, and the composition of an army is also vital to finding an army that has the best chance of defeating the enemy army. In the case that the retrieval can not find an army that has an increased chance against the problem army, it will simply retrieve the best rated army.

4.1.3 Reuse and Revise

We have mentioned in Subsection 2.2.3 that reusing a case will almost always lead to adaption in Warhammer 40k. This is usually true for humans, as the task of finding comparisons is a lot easier for us, than it is for the AI (de Mantaras et al. 2006). From a design perspective this kind of adaption would definitively be difficult to create, as we would never have an army that is the same, and thus retention would be useless. Alternatively retention can store everything we adapt, but then we reach the utility problem.

Another point to consider when designing the adaption is that Warhammer 40k is still a chance based game. While there is a large amount of strategy involved in the game, ultimately everything is decided by the dice rolls. The strategy creates the best statistical advantage for the player.

With these two points in mind, we can say that adaption is not necessary all the time. To represent this, the reuse step will only adapt cases that pass a certain threshold. The threshold is acquired from the retrieval step. If the retrieved army is a good match for the problem army, we will reuse the army as it is without adaption. If the retrieved army is not a good match, then we will perform adaption. If we assume that retrieval is simplified to a similarity index between 0 and 1 a good match is any army that has a similarity of over 0.6. The value is derived from analyzing the metagame statistics. The worst army in the current edition of Warhammer 40k has a 40% chance to win. That means that if we determine our army has a 60% chance or more of winning by the nature of similarity, then we will not adapt this solution. This is a variable that is subject to change, both from the metagaming perspective, but also from a design point.

Once we decide adaption should take place, the adaption method is in most cases going to be a simple substitution, with one squad in the army taking place of another, or a unit exchanging equipment. The adaption is done by targeting the weakest part of the army, by analyzing squads that matched the worst against the problem army. Then these squads are exchanged with others that are known to perform better, by comparing the special rules, options and the rating attributes of a squad, as well as general knowledge.

Transformation of squads is also a possibility, where the options attribute of a squad and the general domain is used to adapt the squad to the problem. This is likely to get done closer to the life cycle of the system, and as we create more and more squads, we will create more and more knowledge in the system, and thus require less transformation adaptions, which is also the more expensive adaption in the system.

It is important to note that at every step of the adaption process the system checks if the army is legal and if it is within the rules of the general domain knowledge, based on what kind of battle is played. Unbound battles need to be checked for points, where as battle forged armies need to be checked for the legality of the divisions. If the adaption is not correct, the adaption proposes another solution. This is done until a legal army is created.

After playing a game of Warhammer 40k with the recommended army from the previous two steps of the CBR cycle, the data about the performance is entered back into the system. Each squads performance is entered in the system, and thus we need to keep track of each squads performance during the game. The victory or defeat of the armies is also entered into the system. The chance, or dice rolls, are assumed to be fair and thus we ignore that factor. This is done for both the problem and the solution army, as we want our system to be able to learn about different armies from the games.

Once each squad has their performance entered, the system calculates the ratings of each squad. The ratings are calculated by using the formula:

$$SquadRating = \frac{ValueOfDamageDone}{SquadValue}$$

Once all of the performances are calculated, the revision looks at the lower half of the ratings, that is at the SquadRating that is less than 1. If the squad rating is less than 0.5, the squad is replaced with another, similar purpose squad. Squads that have not participated in battle for any reason are considered to have a rating of 1. This calculation is performed for all of the squads in both of the armies.

4.1.4 Retain

Once all of the performances are entered and the army revised, the system adjusts the ratings for all the squads that have existed in the system. Ratings are increased for the squads that have a SquadRating greater than 1, while ratings are reduced for squads with SquadRating less than 1. SquadRating of 1 does not change the ratings of squads.

The armies and squads, alongside their rating attributes, are then retained in the system. For the armies, both armies are retained in the system, and if the army has been directly reused its rating attribute is changed. This helps with the retrieval steps, as well as with further reuse and revision steps. If the squads did not previously exist in the system, the squads are retained in the system with the initial value rating attributes. Squads of the solution army that did not exist in the system are only retained if their performance is greater than or equal to 1. New squads in the problem army are always retained, even if they had performed poorly, as they are a part of the retained army.

4.1.5 General Knowledge

General knowledge has been mentioned several times through the design phase already, and it is established that the general knowledge is necessary to simulate a complex environment like Warhammer 40k.

Here is a short example to illustrate this point. Many Space Marine units have an ability called *And They Shall Know No Fear*. This ability falls under the special rules of the unit object. The ability makes it so that this unit automatically passes fear and regroup checks in the game. This means that, if we are facing another army that can create fear, the particular units have the advantage, as they do not need to take a chance on this roll, but automatically succeed it. There are around sixty of these rules just for units. If we take into consideration that there are also rules about equipment, options for squad creation, missions and terrain, it should be clear that some sort of general domain knowledge is necessary. The general domain knowledge is possible to implement as it is still finite, and it does not change often, only when the system is updated. Table 4.1 shows the uses of general domain knowledge in our system.

Knowledge of	Description
Rules	The rules of Warhammer 40k, including the rules for constructing armies
Special Abilities	Special abilities and rules of all units and equipment
Options	Extra options provided to squads for changing them
Missions	Mission objectives and victory conditions for missions
Terrain	Terrain uses, types and heuristics for terrain advantage
Strategy	Heuristics gathered from experts which advise on general strategy for Warhammer 40k army creation

Table 4.1: General Domain Knowledge Uses

The rules are all of the rules for Warhammer 40k that are important to us. This includes most of the rules for army creation, such as alliance rules for factions/races, battle forged and unbound armies, and the detachment rules, amongst others. This domain knowledge is mostly used in the retrieve and reuse steps, but it is also sometimes used in the revise step.

The special abilities rules are used in the retrieval and reuse steps, and they contain rules for the abilities that the units possess. If used properly, abilities can present a great advantage for an army. The options rules are used in a similar fashion, and in the similar stages of the CBR cycle. A unit with options is more diverse, and can be more adapted to different situations. It is necessary to know what options a squad provides in order to utilize it fully.

The missions knowledge contains knowledge of victory points and objectives. This is used, again, mostly in retrieve and reuse steps. Sometimes a weaker army can be victorious by focusing on the objectives, rather than the combat and elimination of the enemy army. The terrain knowledge holds knowledge of what terrain can do, and how the terrain will benefit or hinder a specific army. It is also used in the first two steps, but as most others has some uses in revision.

Finally, the strategy knowledge is used in the retrieval, reuse and revise steps, and it is one of the most important factors in determining the retrieved army. The strategy knowledge is a set of heuristics retrieved and aggregated from experts in the Warhammer 40k domain. It contains information on the benefits and drawbacks of specific army compositions and how to use these benefits to gain an advantage and limit the drawbacks. It is next to impossible to capture all of the strategy, but a good deal can be acquired by a few simple heuristics, and can greatly aid the system in choosing the correct army during retrieval.

A final use of general domain knowledge is to provide context and term explanations for the explanations in the system. The explanations are discussed in Section 4.3.

4.2 Maintenance Policies

The three maintenance policies that we will consider are the utility, consistency and metagame maintenance. The maintenance policies are an addition to the CBR system, and in terms of the architecture are the Auxiliary AI applications. This means that the maintenance policies do not operate on their own, and require the CBR system to function. The maintenance policies target both sub-goal 1, by helping the system operate, and sub-goal 3, by proactively improving the system.

4.2.1 Utility Maintenance

The utility problem was discussed in Subsection 2.2.4. The maintenance for the utility problem will essentially solve the problem of utility for the system.

The policy will run two timers, most likely on different cores or threads as to not interfere with the CBR system, that will measure the retrieval and adaption step. As the utility problem is defined as the retrieval step becoming greater than the adaption step, we need to stop the problem before it happens. The two timers are compared to each other after each run of the system. Should the retrieval step be within 10% of the adaption step, or in other words should $\frac{RetrievalTimer}{AdaptionTimer}$ become greater than or equal to 0.9, the maintenance policy will be started.

The maintenance policy is performed after the retention step of the cycle when the timers reach the threshold. The maintenance policy will then delete army and squad cases from the case base. The cases that are deleted are the cases with the lowest rating attribute. Furthermore, each faction or race, will have a minimum set of armies and squads in the case base. This means that if we have hundreds of Space Marine armies, but only a handful of Ork armies, the low rated Space Marine armies will get deleted, even if the Ork armies are lower rated. This is done so that the system would not forget about armies and squads that are lower rated, but are still quite different from one another. Armies that boast multiple races will have a count of which race is larger by cost, and that will be considered the primary race of the army, as far as the maintenance policy is concerned. The same principle is applied to squads, only that many more squads are kept than armies, due to the fact that many squads make up armies. On average, we expect to have at least five to ten times the amount of squads than armies in the system, at all times.

Units and equipment, being finite, are never deleted from the case base. As they are not collections and they are much simpler objects, their deletion would have to come from a system update, and they generally have little to no impact on the utility problem.

4.2.2 Consistency Maintenance

In Section 2.4 we discussed the consistency, or update maintenance. Updating our system involves updating the case base and the general knowledge of the CBR system.

One of the things that we did not discuss is the process of the update to the Warhammer 40k system. A Warhammer 40k edition update does not happen in one day. Rather, over the course of some time, usually a year to two years[3], each faction receives an update. This means that in the period between the releases the game is in a mixed state of the previous, and next edition.

[3]For the 7th edition, the Orks codex was released in June 2014 (ISBN 9781782533290), whereas the Dark Angels codex was released in July 2015 (ISBN 9781782537526) with more codex releases pending in the future.

This unfortunately, makes it hard to time a system update. We can not wait one or two years to update the system if the new edition is being played, yet we have to take special care of the rules and units, to make sure they are all legal in the new systems.

Fortunately, since third edition of Warhammer 40k this has not been a problem. Considering that the third edition has been released in 1998, this trend has been going on for over 17 years, at the time of writing. Thus it is safe to assume that all new updates to the system can be made as they come, and they will not interfere with the basic rules of the system. This means that as each update is released, we can subsequently update the system to that version.

An update to the system is performed ad-hoc and is not at all tied to the CBR cycle. The update is created by the developer and then applied to the system, and thus the system will have some downtime while the update is performing.

The first part of the update is the update to the general domain knowledge. All of the parts of the general domain knowledge are updated whenever a new edition is released, assuming there is change. Furthermore, the strategy in the general domain knowledge is updated every two months, regardless of the update to the system itself, so that if there has been any change in strategy the system will be aware of that as well.

The second part of the update is the update to the case base of the CBR system. The case base is updated with new units and equipment whenever the system itself is updated. As the squads and armies inherit units and squads respectively, we do not need to update them. Since the armies and squads fill the majority of the case base, the update itself should perform quite quickly. However, while the squads and armies do not need to be updated, the rating attributes tied to the specific faction or race, are reset. This essentially means that the system does not forget about these armies and squads, but it does forget about their ratings. This is done to prevent the system from using high rated armies that are no longer good, or not using lower rated armies since they were so bad. The system will then, during its life cycle, rate these armies accordingly to their new, updated version.

As the explanation system is tied directly to the CBR system, it needs no updates. Any updates to the CBR system automatically update the explanation system as well.

4.2.3 Metagame Maintenance

Unlike the two other maintenance policies, the metagame maintenance does not actually change the core system in any way. The maintenance policy is not a traditional maintenance policy. Rather, it is more of a simulation that serves as the last auxiliary application in the architecture.

The metagame maintenance is completely separated from the CBR system, much like the update maintenance. It will only start when the user is finished with the system, and indicates that they are finished by allowing the maintenance to run. This will ensure that the maintenance does not run when the system needs to be available to the user. If the user wishes to interact with the system the maintenance shuts down, nullifying any recent changes.

The maintenance policy includes the unit, equipment and squad objects, much like the CBR system does. It also includes rules, special ability and options knowledge from the general domain knowledge in its knowledge base. Furthermore, it includes a combat simulation that actually simulates fighting in between units, and the necessary knowledge to simulate this fighting. It is not another CBR system, nor an AI system, but an application.

When allowed to run, the maintenance policy creates a legal army with a cost of 100 to 150 in points. This is the approximate cost of, at most, three or four squads, or about a tenth of a normal sized army. Two criteria are very important in the creation of this small army. The first criteria, is that the army is completely legal, which is why the system has the background knowledge that it does. The second is that the army created must be created randomly and it must not follow any specific pattern. This means that if the maintenance policy has the knowledge to create a hundred small armies, each army should have a one in a hundred chance of being created. This will ensure that the system is always learning, and not iterating the same solution over and over again.

When the army is created, the maintenance policy automatically presents it as the new problem army in the CBR system. It connects to the CBR system, and feeds it data, in a similar manner that the user would, without the user interface. The CBR system responds by retrieving an army that best matches the problem army, as described in Subsection 4.1.2. The CBR system then uses the Reuse step, as described in Section 4.1.3 to adapt the army, if need be, to the problem army. This adapted army is then taken as data by the maintenance policy.

After the CBR system gives its response, it waits on the maintenance policy, that is acting much like a user, to enter the data back after testing. The maintenance policy now starts the combat simulation, following the phases of combat and player turns described in its knowledge base. The simulation has fixed parameters, no terrain and no missions. It is a simple test of how units perform in combat. The simulation lasts until one army has won.

After the simulation is finished, the maintenance policy enters the data back into the system in the Revise step. The maintenance policy is then finished, as the CBR system performs the Revision and Retention of the case, adjusting the ratings accordingly. This entire process can be seen in 4.5. The maintenance policy steps are represented with squares, and the CBR system steps are represented with ellipses. The arrows in the figure represent the work flow, which loops at the end of a cycle.

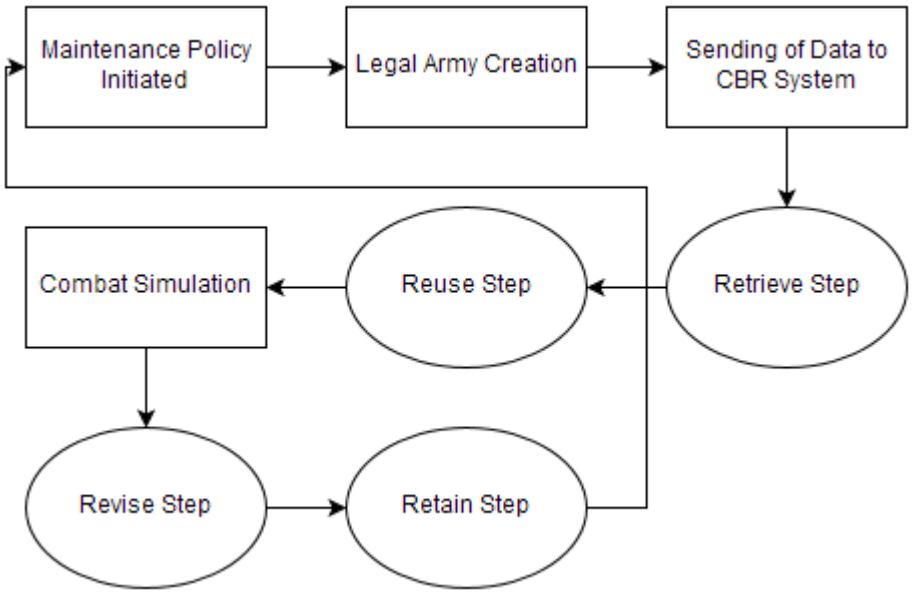

Figure 4.5: Metagame Maintenance Policy Sequence

To simplify the combat simulation the smaller armies are used instead of correct point armies. Furthermore, this kind of policy is highly experimental, and it is safer to use smaller armies and increment the system slower, rather than teach the system poorly with larger armies.

This maintenance policy does require some explanation to the user. In this case, we must amend our architecture to state that the explanation system will also generate explanations to what has happened in this maintenance policy. This should be considered a special case of our architecture, and not something that should be normally done.

4.3 Explanation

The explanations in the system will focus on explaining how the system has derived a solution, and why it has chosen that particular solution. The conceptual part is automatically derived from the content of the retrieved solution, and does not need to be made explicit. Each of the four steps of the CBR cycle will be explained by the system to the user. Each of the steps will use the context from the knowledge containers, and also use the general domain knowledge is included in the vocabulary container, to explain its actions. Furthermore, the maintenance policies will all be explained to the user as well.

The explanations in the system work as a two-stage method. The first explanation that is shown is a simple explanation generated as the system functions. This explanation is the why explanation, or the justification of a systems actions. The explanation is usually not longer than a sentence. For example, after retrieving the army the system will say: *This army has a rating of 2083 and has a 67% chance to win against the enemy army.* This kind of explanation fulfills all of the criteria of what a good explanation is supposed to be, as presented in Section 2.3.1. It is a representation of what the system does; it is understandable to a normal user; since the system drew the explanation from the knowledge containers it knows what it is talking about, therefore it is sufficient; the explanation is generated as the system runs, probably on another core, and it does not deter the performance of the system, so it is efficient and has low construction overhead.

While the explanation is a good explanation for newer users, it is not sufficient for experts. It is also not very instructive, other than allowing the player to learn exactly why the solution is good by playing. Therefore, the second stage of the explanation is created. This stage is done in conjunction with the first, and again, fulfills the criteria, though it is somewhat more demanding performance-wise than the first explanation due to the verbalisation of the longer explanation.

This second stage explanation is presented to the user in the form of a button captioned *tell me more*. If the user clicks on the button, a longer explanation is provided, using more terms and features. In our example of retrieval, the system might say something like: *This army is good against the enemy army, because units have And They Shall Know No Fear, and the enemy army has Fear as special abilities. This army is good against the enemy army, because units have longer range and blast equipment, and the enemy army has numbers advantage* and so on. This button will be present on every stage of the system, and the user will have to initiate the explanation, since the length and terms may be overwhelming to a new user.

4.3.1 Explanations - CBR

The first step of the CBR cycle that is explained is the retrieval step. The justification part of the explanation is drawn entirely from the rating. Therefore, the similarity measures of the k-NN algorithm provide context for the justification. For the how explanation, the system uses context from the similarity mapping engine, determines the most valuable features of the retrieved army, and explains them by simply presenting them, alongside the vocabulary container which provides context for the special abilities, rules, missions and so on. An example of the retrieval explanation was presented already. In the case of compromise-driven retrieval, the system will explain the difference between the compromise army and the best army, if the difference would be worse for the user.

Adaption is explained in a similar manner, using the adaption knowledge container and the vocabulary container. A justification of the adaption merely states why, or why not, an army was adapted. An example of the system dialogue could be: *The army had no blast equipment to fight the enemy army with larger numbers, so blast equipment was added instead of single target equipment.* If the adaption step does not change the army, then a message is shown explaining this. For a more in depth explanation, the entire adaption process is shown, showing exactly what has been changed. The same example would then be: *The army had no blast equipment to fight the enemy army with larger numbers. Strategy states that blast weapons are very effective against enemy army. Equipment with (Strength greater than 8, Single) has been removed, and replaced by (Strength 5, Blast 2).* These are simple examples and the length of the in depth explanation can be expected to increase in the actual implementation of the system.

Justification in revision is presented in a simple fashion. Any squads that are replaced are simply noted as bad performing, and they have been replaced by better performing squads. The system would say: *Some squads that have performed badly have been replaced with other, better performing squads.* If none of the squads have been replaced, the explanation is fairly static. The in depth explanation, however, goes into the detail of rating attributes, and which squad has taken the other squads place, their ratings and why that particular squad was chosen. For example: *Tank Squad has been replaced with Artillery Squad. Tank squad has had a performance rating of 0.4 in the game and is rated at 1403 points. Artillery squad performs the same role according to Strategy and is rated at 1836 points.*

Finally, when retaining squads and armies, the system will explain, in the form of a message, that the system has adjusted squad ratings, and will state that squads with a poor rating were not saved. A more in depth explanation will show an explanation that includes a list of squads and their previous ratings, and their new adjusted ratings, as well as how they were adjusted.

4.3.2 Explanations - Maintenance

The maintenance policies themselves will need some kind of explanation as well, as we are performing changes on the system directly.

The first maintenance policy, which addresses the utility problem, has to do with case deletion. Subsection 4.2.1 outlines the details of this implementation. As this is done outside of the CBR cycle, the explanation is a pop up window, which indicates what has been done. The explanation in this case is a middle ground between justification and transparency. The system will indicate which armies it has deleted and provide those deleted armies as cases for the user to view if they wish. This is the transparency part of the explanation. The system will then justify that the deletion was done because of its low rating attribute. In this way both the novice and the expert users can look over the maintenance policy, and possibly try to reintroduce the armies deleted back to the system at a later date.

The second maintenance policy, to address consistency, does not need an explanation. The developer is aware of exactly what is updated as he is the one doing the update to the system. An update manifesto will be presented to the user, that indicates what changes have been done in the system as well as the version of the current system. The only control over the update that the user has is whether or not they wish to apply the update.

The last maintenance policy, which addresses the metagame simulation, will create explanations in a log form. As the maintenance policy is intended to mostly run when the user is not using the system, it is assumed that the user will not be next to the system to read the explanations. Instead, the system will write two log files in the form of simple text files. One log file will indicate basic explanations followed by a tag number that will be incremented for each run of the maintenance policy. The other log file will be a longer list of the entire process of the maintenance policy. This process is derived from the other explanations in the CBR system, and from the log of actions and dice rolls in the army creation and combat simulation. In this fashion a user can look at the simple log file, and if there is an interest for any particular run of the maintenance policy, the user can look that up in the longer log file by finding the tag number.

4.4 Prototype

The implementation of the system is outside the scope of this project, and so is the creation of a prototype. However, having some kind of prototype is useful for understanding how the system works first hand. This section will showcase a dialogue between a user and the system. Italics represent actions, while non formatted text represents the actual on-screen prompts.

In this scenario, the enemy army is an Ork army, featuring large numbers of melee and short ranged units, with high strength and high armour saves (remember that the higher armour saves the worse it is for the unit). The terrain is an open field with trenches and rocks for cover. The mission is to defeat the other army. The battle is unbound, so no divisions are needed.

User: *Enters the enemy army into the system.*
System: *Performs the retrieval step calculation.*
System: This army is a high rated Space Marine army. It has a 79% chance to win this match.

User: *Tell me more.*
System: This Space Marine army has seven Tactical Squads of Space Marines, one Captain Squad, one Captain Sicarius squad, one Assault Squad. Strategy suggests that mid ranged units with high strength and low armor save and high initiative defeat short ranged units. Strategy suggests that mid ranged units with high strength and low armor save and high initiative defeat melee ranged units. Strategy suggests using warleader. Strategy suggests using equipment that has Blast 1, Blast 2, Blast 3 against short ranged and melee units in groups of more than five. Terrain indicates open field where units with mid or long range perform well. Each squad is 61% or more better than the enemy squad when comparing ratings per squad. Enemy squad contains unknown squads, they are rated at neutral rating (1500). Normal rating comparison indicates win rating chance of 74%. This has been modified with: +5% because of advantages. There are 7 unused points out of 1000.
User:*Continue.*
System: This army has a high chance of winning. adaption is not necessary.
User:*Continue.*
User:*Enters post combat data. The army performed better than the enemy army. Each squads rating is entered. The Tactical Squads average rating is 1.3, The Captain Squad is 1.1, Captain Sicarius 1.8, and the Assault Squad 0.7. The enemy squads performances are also entered*
System: The army will not be improved. The ratings have been registered.
User: *Tell me more.*
System: No squads have performed below the threshold rating of 0.5. The squad that has performed the worst is the Assault Squad with a rating of 0.7.
User:*Continue.*
System:*Performs the Retention Step*
System: The army ratings have been stored. The Space Marine rating has increased by 6.4. The enemy army is retained.
User: *Tell me more.*
System: The army rating has increased by 6.4. Space Marine army was rated at 1780 rating points. Enemy Army was rated at 1500 rating points. Assault Squad rating has been reduced from 1680 to 1651 with a rating change of -29. Tactical Squad rating has been increased from 1710 to 1719 with a rating change of +9. Captain Squad rating has been increased from 1578 to 1592 with a rating change of +15. Captain Sicarius Squad rating has been increased from 1790 to 1796 with a rating change of 6. Enemy army was rated at 1500 since it was not present in the system. Enemy army was retained in the system because of loss. Enemy army is now rated at 1493.

Chapter 5

Evaluation and Conclusion

In this chapter we will evaluate and conclude the project. Section 5.1 will evaluate and discuss the project. Section 5.2 will discuss the contributions of this project to the state of the art, and Section 5.3 will conclude the project and discuss the future work to be done based on the study done in this project.

5.1 Evaluation and Discussion

The goal of this project is to design an explanation-aware CBR army builder for Warhammer 40k. As such, the scope of this project ends where implementation begins, and no formal or informal tests have been performed on the system, and thus there are no direct results to evaluate.

The project has succeeded in the goals it has set out to do. We have designed an explanation-aware CBR system that constructs armies in the Warhammer 40k domain, and we have presented the state of the art in this domain and the domains of the sub-goals. The project has done this by following the formula of Cohen and Howe (1988). We have refined a topic to a task and designed the method. We have constructed a design and an architecture for the system, but we have not implemented a program, and therefore do not have experiments or a discussion of such, as that is outside of the scope of this project.

As we have shown with the state of the art, there is no AI system currently in place that concerns itself with the Warhammer 40k domain. This is both good and bad at the same time. It is good because the research is new and it explores a domain that was not explored before. However, without a present state of the art, we can only guess and make assumptions that the domain will fit the system, and that it will work in the fashion we have designed it. The system will most likely change as it progresses through the development, and while we can anticipate the change, we can not anticipate exactly what will change. Any numerical values, such as squad rating, are also assumed and will most likely be subject to change. The project will require future work placed into it to prove the hypothesis, and we have only reached the halfway point of the scientific method. This project should be therefore looked at as a foundation for future research.

The first limitation of the system is its implementation complexity. The CBR part of the system is quite complex. While many k-NN algorithms exist, creating a structure-mapping engine will be difficult. It requires expert knowledge in the form of the general knowledge, and the creation of this general knowledge will take time. Assuming we could potentially insert all of the rules, special abilities, options, missions and terrain easily, the general strategy and heuristics will take some time to collect from the experts in the field. Furthermore, to create the structure-mapping engine we will need to discuss with the experts and create a list of most important concepts, so that we may as accurately as possible, retrieve a good army. In comparison, once this up front work is done, the Reuse and Revise steps will use the same knowledge, thus no extra work needs to be done for these two steps.

The second limitation lies in between the Reuse and the Revision step. In Section 2.2.3 we have stated that the players are playing to the best of their abilities. As we are targeting both novice and expert users, it should be clear that the skill levels of these players will differ, and sometimes differ greatly. Unfortunately, there is no feasible way for us to predict or gauge the skill of the users, so we must take this limitation as is. This limitation can, and most likely will hinder the learning process. A possible solution to this is to allow players to enter their own expertise into the system, and based on that adjust the ratings of squads and armies by a smaller or larger percentage. However, we risk that the users will under or overstate their skill levels, and create an even worse learning situation. Therefore, we must accept that this is a limitation to the system.

The third limitation of the system is the complex rules of Warhammer 40k. Even with the general domain knowledge, it may be difficult to address every single aspect of the domain. The primary example to this is the warlord unit, discussed in Section 2.1.2. The warlord unit can bring more benefits than just the strength of the unit. The warlord special abilities can enhance all the friendly units on a battlefield in some way, or simply move the army closer to victory. However, our retention and revision steps only count the damage done by this unit, and not the other benefits it brings to the army. These small nuances are hard to capture, which is why we must use a general formula, and why this is a limitation. A possible solution would be to include all the benefits in the calculation of squad rating, but this would make the calculation step very complicated and would in most cases result to the formula we already have. Furthermore, it would make the entry of the revision step even more taxing on the user than it already is, which is yet another limitation of the system.

Two of the three maintenance policies are fairly well known. The utility problem and the consistency updates have both been created and used before, and it should be no problem to adopt them to this system, since the system is created with these policies in mind. The third maintenance policy, the metagame policy, is different. It will require the creation of another program, one with general knowledge, rules, as well as combat simulation. This will require more initial work to produce a tangible result.

Finally, if the maintenance policies and the CBR system is implemented, the explanation part of the system can draw information and context from these systems, and does not need any additional work or sources besides implementation. However, it does depend on a good implementation of both the CBR system and the maintenance policies, and any lack of information there will reflect itself in the explanation system.

5.2 Contributions

The design of the system as a whole is the main contribution of this project. The project analyses theoretical knowledge, the state of the art and designs the architecture of the system. The design of the Warhammer 40k domain becomes an instance of the class of problems, and this instance is further then designed and discussed.

As previously mentioned, the state of the art is created and presented. Often, the state of the art is overlooked and AI projects lack documentation about the state of the art (Cohen and Howe, 1988) or the method at which they have achieved their result. The presentation of the state of the art means that not only this project, but future projects as well, have firm legs to stand on when designing and implementing an instance of this system. This alongside the creation of the architecture and the class of problems is an important contribution to explanation aware CBR systems.

The use of maintenance policies to update the system and make it think is an idea that can be applied to multiple systems, creating proactive systems that can think and work when not interacted with. This is a significant step forward from machines that simply react when a button is pressed by the user.

Finally, work is done on showing that explanation aware computing is vitally important for these kinds of systems. As users without AI knowledge are the primary users of this system, it is important that they are able to understand and trust the system, in order for the system to be taken seriously. This extends to not only this system, but to other AI systems as well.

5.3 Conclusion and Future Work

This project has set out to design an explanation aware case based reasoning system, and has succeeded in doing so. It has also presented the state of the art concerning the system. It has done this methodically, and it has approached the problem from a general view first, and domain view second, so that this project may further be used for designing similar instances of this kind of system.

In Section 1.2 we have defined the goals for this project. To conclude this project, we will look at and discuss these goals again.

Goal 1 Design an explanation-aware CBR army builder for Warhammer 40k

The first goal was subdivided into three goals, Designing a CBR army builder for Warhammer 40k, Designing an explanation aware CBR system and designing a system that can evolve within the environment that is Warhammer 40k.

Sub-Goal 1 Design a CBR army builder for Warhammer 40k

We have designed the underlying architecture for the class of problems and we have taken an instance of this architecture and designed a CBR system that can create armies in Warhammer 40k. The limitations of the design are presented in Section 5.1, and the fulfillment of the goal should be considered with those limitations in mind. Furthermore, this part of the project has designed the hypothesis, and for it to have any scientific value the system will need to be implemented and tested.

Sub-Goal 2 Design an explanation aware CBR system

We have designed an explanation aware part of the system, and discussed how it is implemented within the architecture and the design itself. We have shown how we will extrapolate the knowledge necessary for the explanations and how these explanations will be presented, in order to build trust and confidence towards any kind of user. Again, much like with the first sub-goal, we will need to implement and test the system in order to be assured that it is correct.

Sub-Goal 3 Design a system that can evolve within the environment that is Warhammer 40k

We have designed a system with three maintenance policies. Two of these, the update and the metagame maintenance policies, are a crucial part in fulfilling this goal. The update maintenance policy handles the frequent updates to the domain, while the metagame policy simulates active usage of the system, and thus proactively betters the system. We have also shown how and when these policies will run, and their respective properties. Similarly to the first two sub-goals, we must implement and test this design before we can be assured that it is correct.

It should be noted that even though implementation and testing is necessary to prove the hypothesis and complete the first goal of our project, all of these sub goals have been designed with the state of the art in mind, and we have no reason to believe that they will not function from a theoretical point of view.

Goal 2 Present the state of the art

The state of the art has been presented in Section 2.5. The process of acquiring the state of the art is also present within the same section.

This project is the first step towards creating and fully implementing a army-building system for Warhammer 40k. The next step is the implementation of the system, the collection of test data, and the refining and testing of the hypothesis. Past this, the system should also be capable of legacy support, to allow users to switch between different versions of the game.

Bibliography

[1] Schank, R.C. 1982 *Dynamic Memory: A Theory of Reminding and Learning in Computers and People*, New York, NY: Cambridge University Press.

[2] Ramon Lopez de Mantaras et al, May/June 2006 *Retrieval, reuse, revision, and retention in case-based reasoning*, IEEE Intelligent Systems, pp. 39-49.

[3] Thomas R. Roth-Berghofer, 2004 *Explanations and Case-Based Reasoning: Foundational Issues*, P. Funk and P.A. Gonzales Calero (Eds.): ECCBR 2004, LNAI 3155, pp. 389-403. Copyrighted by Springer-Verlag Berlin Heidelberg 2004

[4] Richter and Weber, 2013 *Case-Based Reasoning*, eBook, Springer Heidelberg New York Dordrecht London Copyrighted by Springer-Verlag Berlin Heidelberg 2013

[5] Cohen and Howe, 1988 *How Evaluation Guides AI Research.* AI Magazine Volume 9 Number 4 (1988), Copyrighted by AAAI.

[6] Schank, R.C 1986 *Explanation Patterns: Understanding Mechanically and Creatively*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1986.

[7] Johanna D. Moore and William R. Swartout 1988 *Explanation in expert systems: A survey*, Research Report RR-88-228, University of Southern California, Marina Del Rey, CA.

[8] McSherry, D. 2003 ***Similarity and Compromise***. *In proceedings of the Fifth International Conference on Case-Based Reasoning*, Berlin: Springer, pp. 291-305.

[9] Wizards of the Coast 2014, *Warhammer 40k, 7th Edition*, Games Workshop Ltd. Willow Road, Lenton, Nottingham, NG7 2WS, Copyrighted by Wizards of the coast and associates.

[10] Aamodt A. and Plaza E. 1994 *Case-based reasoning, foundational issues, methodological variations, and system approaches*, AI Communications 7(1), 39-59

[11] Aamodt A. 2004 ***Knowledge-intensive case-based reasoning in Creek***. *In Proceedings of the Seventh European Conference on Case-Based Reasoning*, Berlin: Springer, pp. 1-15

[12] Peter Spieker. 1991 *Naturlichsprachliche Erklarungen in technischen Expertensystemen*, Dissertation, University of Kaiserslautern.

[13] Anders Kofod-Petersen, October 8 2004, *How to do a Structured Literature Review in computer science.*

[14] Tintarev N. and Masthoff J. 2012, *Evaluating the effectiveness of explanations for recommender systems*, copyrighted by Springer Science+Business Media B.V.

[15] Vig J. and Sen S. and Riedl J. 2009, *Tagsplanations: Explaining Recommendations Using Tags*, IUI'09, Sanibel Island, Florida, USA.

[16] Cleger S. and Fernandez-Luna J.M and Huete J. F. 2014, *Learning from explanations in recommender systems*, Elsevier Inc. All rights reserved.

[17] Smyth B. and Keane M. T. 1995, ***Remembering to forget: A comptence-preserving case deletion policy for case-based reasoning systems.*** *In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* San Mateo, CA: Morgan Kaufmann, pp. 377-383.

[18] Smyth B. and McClave P. 2001 ***Similarity vs. diversity.*** *In Proceedings of the Fourth International Computational Intelligence 17(2), 196-213.*

[19] Glinz M. 2007 *On Non-Functional Requirements*, 15th IEEE International Requirements Engineering Conference, New Delhi, India.

[20] Tidemann A. and Bjornson O. and Aamodt A. 2011 *Case-Based Reasoning in a System Architecture for Intelligent Fish Farming*, a joint work of SINTEF Fisheries and Aquaculture AS Trondheim, Norway and the Department of Computer and Information Science, Norwegian University of Science and Technology, Trondheim, Norway

[21] Sormo, F. and Cassens, J. and Aamodt, A. 2005 *Explanation in Case-Based Reasoning–Perspectives and Goals*, Artificial Intelligence Review (2005) 24: 109–143.

[22] Bass, L. and Clements. P, and Kazman, R. 2013 *Software Architecture in Practice: Third Edition*, Copyrighted by Pearson Education, Inc.

[23] Zhang, Z. and Yang, Q, 1998 *Towards lifetime maintenance of case base indexes for continual case based reasoning*, Volume 1480 of the series Lecture Notes in Computer Science pp 489-500.

[24] Leake, D. B. and Wilson, D. C, 1998 *Categorizing case-base maintenance: Dimensions and Directions*, Volume 1488 of the series Lecture Notes in Computer Science pp 196-207.

[25] Roth-Berghofer, T. and Iglezakis, I. 2001 *Six Steps in Case-Based Reasoning: Towards a maintenance methodology for case-based reasoning systems*, Includes the Proceedings of the 9th German Workshop on Case-Based Reasoning.

[26] Roth-Berghofer, T. and Cassens, J. 2005 *Mapping Goals and Kinds of Explanations to the Knowledge Containers of Case-Based Reasoning Systems*, Volume 3620 of the series Lecture Notes in Computer Science pp 451-464.

[27] Cunningham, P. and Doyle, D. and Loughrey, J. 2003 *An Evaluation of the Usefulness of Case-Based Explanation*, K.D. Ashley and D.G. Bridge (Eds.): ICCBR 2003, LNAI 2689, pp. 122–130, 2003. © Springer-Verlag Berlin Heidelberg.

[28] Diaz-Agudo, B. and Gonzales-Calero, P.A. 2000 *An Architecture for Knowledge Intensive CBR Systems*, E. Blanzieri and L. Portinale (Eds.): EWCBR 2000, LNAI 1898, pp. 37–48, 2000. © Springer-Verlag Berlin Heidelberg.

[29] Aamodt, A. 1994 *Knowledge-Intensive Case-Based Reasoning and Intelligent Tutoring.*

[30] Aamodt, A and Nygard, M. 1995 *Different roles and mutual dependencies of data, information, and knowledge — An AI perspective on their integration*, © 1995 Elsevier Science B.V. All rights reserved

[31] Park, C.S and Han, I. 2002 *A case-based reasoning with the feature weights derived by analytic hierarchy process for bankruptcy prediction*, © 2002 Elsevier Science Ltd. All rights reserved.

[32] Watson, I. 1999 *Case-based reasoning is a methodology not a technology*, © 1999 Elsevier Science B.V. All rights reserved

[33] Aamodt, A. 1994 *Explanation-driven case-based reasoning.*

[34] Aamodt, A. 1990 *Knowledge-Intensive Case-Based Reasoning and Sustained Learning*, Published in ECAI-90, Proceedings of the 9th European Conference on Artificial Intelligence, edited by Luigia Aiello, Stockholm, August, 6-10,1990. Pitman Publishing, London, 1990. Pages 1-6.

Appendix

Appendix A - Glossary

AI - Artificial Intelligence
CBR - Case Based Reasoning
ISBN - International Standard Book Number
k-NN - k nearest neighbour algorithm. This algorithm retrieves the k nearest most similar neighbours
NTNU - Norwegian University of Science and Technology/Norges teknisk-naturvitenskapelige universitet
SLR - Structured Literature Review
Warhammer 40k - Warhammer 40000, often referred to in text as Warhammer 40k, as it is easier to read

Appendix B - Software Used

Latex, a word processor and document markup language used to write this document. www.sharelatex.com was used to write the document. First accessed on September 10, 2015.

draw.io to draw flowcharts, from https://www.draw.io/. First accessed on October 22, 2015.

Fide Chess Rating calculator to help calculate ratings and understand the rating system for chess, from https://ratings.fide.com/calculators.phtml. First accessed on October 27, 2015.

Appendix C - The Rating System

The rating attribute of armies and squads is calculated using the same method that is used for chess players and the calculation of elo ratings. The K number in this case is 40.

Armies and Squads are introduced into the system at an even rating. This rating is an integer number that equals 1500. The ratings are adjusted identically for the squads and the armies.

The rating goes up if a win is recorded, or the performance of a squad is greater than 1. A performance less than 1 is considered a loss, and a performance of more than 1 is considered a win. For every 1 performance over 1 the squad gains an additional win, to represent that it did exceptionally well. Squads that have a rating of 1 remain even.

Due to the fact that a squad is very likely to engage other, varied rated squads, the squads rating is compared to that of an army in order to calculate the points. This assumes that the base average value of the rating of an army is typically an average value of its squads, over time. While this may be a limited assumption, especially since we can create a new 1500 rating with very good squads, due to complexity reasons we simply have to accept it.

The rating system does not increase linearly. The bigger the difference is in between the armies, the less points are gained for the winning army if that army was the higher rated army. At equal rating, a win is considered 20 rating points, a loss is considered -20 points, and a draw is considered 0 points. A rating that differs by 400 points is the maximum difference that the system calculates. Any larger rating differences are set back to 400. At this rating, a win for the higher rated army is only 3.2 points, while a loss is 36.8 points. This is done to prevent any army from reaching incredible ratings by fighting only weak rated armies.