NTNU – Trondheim
Norwegian University of
Science and Technology

CBR-based Explanation-aware Army Builder for Warhammer Fantasy Battle

Eivind Hærum

Master of Science in Computer Science
Submission date: June 2015
Supervisor: Anders Kofod-Petersen, IDI

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

Sammendrag

Warhammer Fantasy Battle er et brettspill med to distinkte faser, hærbygging og selve kampene med disse hærene. Hærbyggingsfasen kan sees på som et resurssoptimeringsproblem, hvor målet er å utnytte en poengsum best mulig for å bygge en god hær. Godheten til en hær er ofte definert av hvor godt enheter oppfyller hverandre og har synergi seg i mellom. Derfor er det gunstig å kunne basere seg på tidligere erfaringer i form av ferdig utprøvde hærer.

For å være i stand til å foreslå hærkomposisjoner til en bruker, og gi han muligheten til å spesifisere hvilke enheter han vil ha med og poengrensen til denne hæren, har vi bygget et case-based reasoning (CBR)-system. Systemet baserer seg på gjennbruk av en tidligere hærliste og adaptering av denne inntil den oppfyller brukerens krav og befinner seg innenfor spillets regelverk. Systemet er realisert ved hjelp av rammeverket myCBR.

Resultatene tilsier at systemet er fleksibelt nok til å løse testkonfigurasjonene, og kan gi en rekke forskjellige lovlige løsninger. Det er dog uklart om disse løsningene er gode nok for å bruke i en ekte kamp.

Problem Description

Warhammer fantasy battle, 40k, warmachine and hordes are strategic games using miniature figures, where each of two players control one army. One of the challenges is constructing a "good" army, based on the rules of the games. Choices made as to the nature of the troops, their equipment, and so forth is typically a function of opponents and player style.

This project aims to develop a decision support system for army building. The system must be able to explain, among others, army composition based on the rules applied and the choices made by the player. Case based.

Assignment given: 18. January 2015
Supervisor: Anders Kofod-Petersen, IDI

Abstract

Warhammer Fantasy Battle is a board game with two distinct phases, building army lists and using these armies in combat. The army list creation can be viewed as complex resource optimization problem, where the goal is to fully utilize the points available to build a good army. The goodnes of an army is often defined by how well the units of that army synergize with each other, and make up for drawbacks of other units. Thus some insight may be aquired from previous experiences in the form of well tested complete armies.

To be able to propose complete armies to a user, and provide the opportunity to specify both units to be included and the number of points of this army, we have built a case-based reasoning (CBR) system. The system bases itself on reusing a previous army list and adapting this army until it meets the users requirements and stays within the game rules. This system was achieved by using the framework myCBR, its powerful modeling capabilities and retrieval engine.

The results verify that the system is able to produce a varied set of valid solutions to the problem. Yet the goodness of these solutions have yet to be determined.

Preface

Conducted at NTNU for the AI group. I would very much like to thank my supervisior Anders Kofod-Petersen for his invaluable input and our discussions. I would also like to thank the WFB community here in Norway for answering any questions I have had regarding the design and implementation, and also for their contribution with data sets. Furthermore I wish to thank my fellow students at ITV262 Sule for our discussions, and keeping up spirits during this semester. Finally I wish to thank my friends and family for supporting me through this endeavour.

Eivind Hærum
Trondheim, June 21, 2015

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 BackGround and Motivation

I have always been interested in games, specifically those which comprise of some role playing or strategy elements. Both of these categories are a large part of the game of Warhammer Fantasy Battle (WFB). I myself was not well acquainted with this game however, but I was familiar with the basic concept of how to play. Therefore my work not only included researching how to adapt the elements of WFB into a CBR setting, but also to learn the intricacies of the game.

Warhammer army building is inherently a constraint satisfaction problem as there are only a given amount of resources you are allowed to use, and these resources cannot all be spent on more than a certain amount of units from each group in the unit hierarchy. WFB is a board game played in the physical world where all the units are naturally unique and may also have access to a wide variety of distinctive weaponry. Thus a secondary set of constraints that needs to be taken into consideration is the fact that not all players will have access to all the units nor all the weapons each unit can use. Another aspect that must be considered is that players often choose to specifically focus on a certain playstyle, which limits the pool of desirable units.

WFB does have some characteristics similar to optimization problems, but there are certain elements of chance and uncertainty during the gameplay that differentiates this game from an ordinary optimization problem. What it does share with optimization problems are a given finite pool of resources that should be allocated in the most efficient manner possible. However, as mentioned the gameplay is not completely deterministic, thus the solution space is possibly infinite. Firstly due to the fact that you most likely do not have complete knowledge of your opponents army when creating your own, secondly the number of different

ways to compose your army are innumerable, and finally the outcome of actions during the gameplay are affected by the roll of dice. It is therefore quite difficult to determine beforehand whether an army will succeed or fail in the heated battle. However, given enough data you could possibly determine certain combination of units that work better together than others. A CBR system might very well accomodate this notion, as it would be able to review previous successful armies and try to adapt these to the given criteria. As the solution space is so vast, the adaptation phase of the CBR cycle could be handled by a CSP solver to speed up the process. This is also suggested as a possible improvement by Strandbråten and Kofod-Petersen [2011]. The end users of this designed system are going to be novice/intermediate WFB players so it is important that the user will be able to understand why the suggested army is good, thus the system is also required to have a emphasis on being able to explain itself to the user.

Following are some facts to illustrate how complex army building is in WFB: There are currently 15 playable races per Warhammer [2010], each of which have access to a number of different figures. A simple count in the beastiary provided in the book showed that there may be as few as 17 figures to as high as 44, and this is before any sub-figures are taken into account. Additionally, there are a large number of different weapons and pieces of armor available, some just for a few figures, and some that can be used by many. Furthermore all the figures cost a specified amount of points to use in the army, and there are several different thresholds for how many points the army may consist of. WFB army building share some similar traits with AutoClave (Hinkle and Toomey [1994]) in this regard. In both domains there are a finite set of resources, one piece of metal versus one upper limit for points, and in both domains the aim is to use as much of these resources as well as possible. A CBR approach is used in that work to store previous solutions that may be reused, or new configurations may be required. The CBR system was deemed very helpful, both in finding viable solutions and in reducing the time needed to find these solutions.

Some work already exists for this problem, Strandbråten and Kofod-Petersen [2011] worked with a very similar problem description and made an inital pass that seems promising. The result of their work was a working CBR system in this domain that was able to explain its suggestions to the user. It did however operate on a slightly limited model, only 3 out of 15 races were included. The adaptation phase attempted to match the query from the user with that of the retrieved cases and populate the solution with as much information as possible. It then used a constraint method in deciding whether the solution was within the boundaries of the rules, before finally making additions where necessary to match the query using a similarity measure for the units. This similarity measure worked by comparing the attributes of the units and looking at its type classification. The revise phase was entirely manual, the users could change parts of the suggested

solution until they were satisfied and could then test the solution. The solution was then stored in a temporary memory until the user could return with the result of the battle, thus enabling the system to either retain or discard the solution. The work done on the domain knowledge was also very specific, and time consuming.

The design in this thesis will take a slightly different approach. The intent with this design is to allow the users to make queries that may be incomplete. The user might only specify the race of the army list and a target number of points used. He may also add information about which units that should be included from the final result. To accomodate this feature, the cases have to be indexed on several fields. This indexing should also help making the solver find a viable solution. The intent of the solver is to have it create a complete solution to the problem description by altering and collecting information from the retrieved cases. The explanation system would look at the reasoning trace of the solver and display the information to the user to justify the solution. The system will also be based upon generality in domain knowledge, thus working with already complete data sets as opposed to model the whole domain from scratch.

1.2 Goals and reseach question

Goal *Use CBR-method to create complete Warhammer Fantasy Battle army list configurations (rosters)*

Sub Goal Determine what kinds of explanations are useful in this setting

Sub Goal Adapt the proposed design to specific implementation plans

Sub Goal Create a system capable of the proposed functionality

Research question *How well can a CBR system function within the boundaries of WFB army creation?*

By answering this question we ultimately discover how well CBR methodolgy can function for a resource optimization problem.

1.3 Research Method

To be able to create a suggestion system it is vital to know what kind of systems already exist, both in the domain of WFB, but also in the larger scope of resource

optimization and planning. Thus the search for previous work included looking for such systems, and finding papers exploring similar work.

As extensive domain knowledge is important when creating a suggestion system, interaction with domain experts become increasinly important. Thus a set of interviews were conducted to broaden the domain knowledge and get a better grasp of what the WFB players find important both when playing the game, but also in their expectations of a suggestion system.

1.4 Thesis Structure

Here details regarding the following chapters will be explained briefly.

Chapter 2 will thoroughly walk through the most vital background knowledge this work is built upon. Following in Chapter 3 the architecture will be explained and implementation specifics will be accounted for. The details and results of various experiments are accounted for in Chapter 4. The significance of these findings are then evaluated and discussed in Chapter 5. Finally the thesis will be concluded with Chapter 6, and possible suggestions for future work are explored.

Chapter 2

Background Theory and Motivation

The work done with workflow management and CBR by Weber et al. [2004] shares some interesting properties with this problem. The approach CBRFlow could potentially be helpful with the work done here, yet the heavy reliance on specific rules for each occuring event do not mix too well with WFB. A perhaps more interesting approach is reviewed in López [2002], where the combined use of CBR and CSP methods form a scheduling service for holiday planning. The notion of maximizing the time available with activites bears strong resemblance to maximizing the points spent for WFB most effectivly, and a CSP gives enough room for randomness to explore multiple different scenarios. This is further explored in Chaper 2.6.

2.1 Warhammer Fantasy Battle

As briefly mentioned in Chapter 1, Warhammer Fantasy Battle (WFB) is a turn-based miniature figure board game set in the physical world. Each player controls an army of units with various abilites and attributes. WFB is not a traditional board game as the board is a constructed battlefield with various obstacles, terrain changes and points of interest. The battlefield is likely to be different from game to game as the battlefield is constructed by modular parts and the players take turns in placing the pieces together. In fact this construction is an important part of the gameplay. The armies of each player initially start on the opposite sides of the battlefield, and as the battle commences the players take turns in moving their units, use spells and attack. The outcome of many actions are

partially determined by dice. The dice may be rolled when attacking, moving, casting spells and so forth, and the outcome of these actions are then determined both by the units internal attributes for the action and the result of the roll of dice.

A unit is defined as either a singular miniature figure or multiple figures grouped together. Each figure in the unit performs the same actions and thus the unit acts as a singular entity. This enables the players to easily command big armies of units and thereby choose how much micro management of the figures they wish to have. Given two identically composed armies, the structured groups of units may vary greatly.

Each figure has a set of attributes that characterizes their capabilites as seen in Table 2.1. They may also have certain special abilites that could alter the capabilites of the figure beyond mere numerical values.

Attribute	What does it determine
(M) Movement	How far the unit can move in inches
(WS) Weapon skill	How well the unit can wield a melee weapon
(BS) Ballistic skill	How well the unit can wield a ranged weapon
(S) Strength	How much damage the unit will do when striking
(T) Toughness	How well the unit is able to resist ranged or melee hits
(W) Wounds	How many hits the unit is able to endure
(I) Initiative	How fast the unit is when attacking
(A) Attacks	How many times the unit can attack this turn
(Ld) Leadership	How brave the unit is, how unlikely it is that the unit will flee

Table 2.1: The standard attributes for a miniature figure

On a grand level the object of the game is to win through conquest, thus the importance of building a good army is a crucial aspect of the game. To ensure an even playing field it is customary to set a restriction on what units are allowed both through a point system, but also by enforcing that there may only be a certain amount of special units for each player as seen in Table 2.2. Each figure and accessory weapon have a specified price of a certain amount of points, and the agreed upon limit of points thus specifices what sort of army you may be able to build. The point limit also has the functionality of specifying an approximate duration of the battle, where a higher point limit usually will enable a longer battle. Standard point limits are 1000, 2000, 2400 and 2500, but as the players themselves determine the limit there is virtually no upper limit on how high this point limit may be set.

The game is divided into two very distinct phases. Building the army list, and playing the game with this army list against an opponent. The focus of the work done in this thesis is on the army list creation phase.

Type of unit	Point threshold	Duplicate units limit
Special	<=50%	up to 3 (6 if over 3000 pts)
Rare	<=25%	up to 2 (4 if over 3000 pts)
Lords	<=50%	no limit
Heroes	<=50%	no limit
Core	>=25%	no limit

Table 2.2: Unit distribution in an army

2.2 Case based reasoning

Case-Based Reasoning (CBR) is a field of Artificial Intelligence that is well explored, papers such as Aamodt and Plaza [1994] specify the possibilites and limits of this method. The main concept of CBR is that learning from previous experiences will allow for better decision making in future events. Similar to how a human might store an important event or piece of knowledge and later recall this when faced with a similar occurrence, a CBR system will store new interesting facts as *cases* which can be reused later. These cases form the case base which encapsulates all the knowledge the CBR system has available when faced with new challenges. Typically though, the CBR system needs to be able to adjust to events that are only partially similar to what it has stored in the case base, thus the CBR system is usually divided into 4 phases of a cycle.

- **Retrieve** Find the most similar cases.
- **Reuse** Attempt to solve the problem by looking at the information and knowledge stored in these cases.
- **Revise** Review and test the proposed solution and make alterations where necessary.
- **Retain** Store the solution if it is a noteworthy addition

Typically the retrieve phase is achieved through some similarity measure, where the k nearest neighbours (kNN) are then returned as the closest matching cases. The reuse or adaption phase is where the system will attempt to build

a solution to this new problem by reviewing the retrieved cases and extract the useful information from these cases. In the revision phase the system is able to learn how well it fared by testing the proposed solution. This may take time, depending on the which domain the CBR system is used in, so the system might keep the solution in a different case base for untested solutions. If the solution is deemed unsuccesful the system may either discard it, or attempt to repair it to better match the problem. Finally if the solution passes the tests and then is marked as a good solution, the system may retain it in the permanent case base.

The similarity measures are varied, yet the main paradigm in similarity functions is to compare feature vectors to determine similarity. Cunningham [2009] presents a overview of different approaches.

2.3 myCBR

myCBR is an open source tool designed to assist in the creation on a CBR application. The main strength of this tool is the ability to create complex models of a domain rather quickly, and create similarity functions for this model. The tool enables this through two seperate components, the workbench GUI and SDK. The workbench is the prime tool for creating the model, the similarity function and finally running simple test queries on the model, whilst the SDK is what connects the myCBR tool to a standalone CBR application. The SDK provides similar capabilities as the workbench, but due to some extra complexity in working with the SDK compared to the workbench, the SDK is primarily used for adding and retrieving data instances from the model and not altering the model itself. Bach et al. [2014] and myCBR [2014] provides a thorough overview of myCBR.

The model is created as a *Project*, and this model is populated by different *Concepts*, which finally can have several different *Attributes*. These attributes may be one of several different types depending on what sort of information they should store. Some of these types are integer, float, string, boolean, symbol and concept. With the exception of symbol and concept the types are quite simple, storing only rudimentary data, whilst the symbol type can only store a subset of defined values similar to an ENUM. The most interesting type though is Concept, as this enables the creation of a tree-like structure of concepts acting as sub-concepts for other concepts, similar to a tree.

To populate the model with actual data the tool has the built-in ability to create *Case Bases*. These stores *Cases* in the form of *Instances*, where an *Instance* is an instantiated *Concept* object. The case bases are able to store any type of instance, but it is most beneficial to only store instances of the same type together, and elect to create several case bases for different purposes. This is due to how the retrieval engine does not distinguish between instance type when running

queries, which in itself is not a problem as wrong instance types will return with zero similarity but it makes the retrieved cases unnecessarily cluttered. Creating a seperate case base for each type of used concept is a good starting point, and further case bases may be created for subsets of instances where applicable.

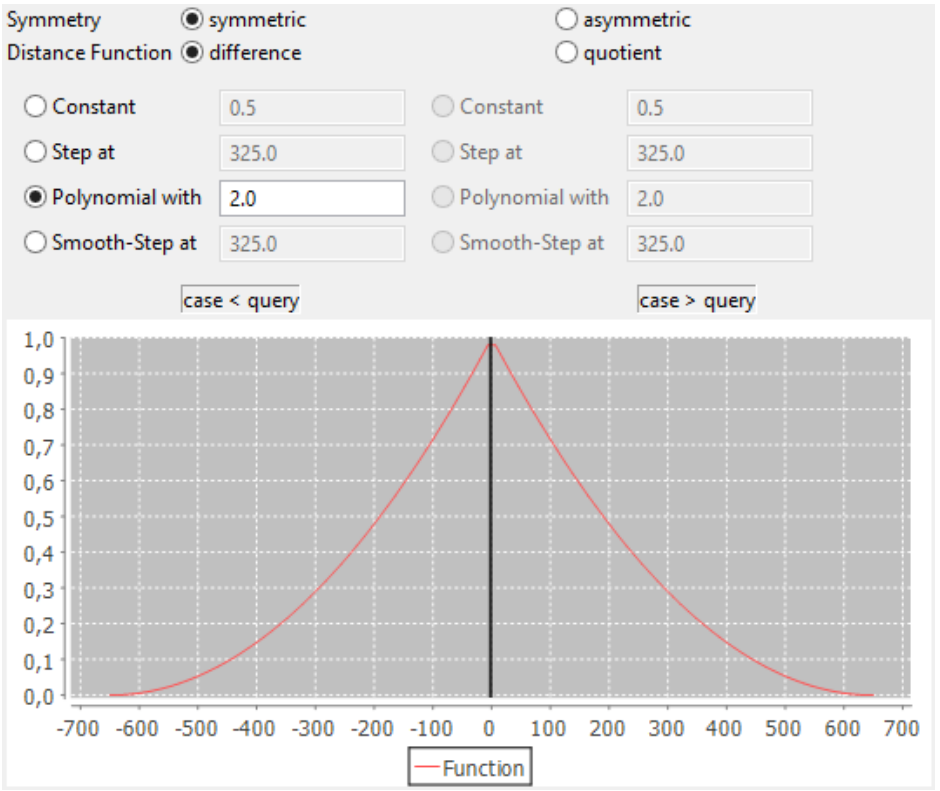

Figure 2.1: Float similarity

In order to enable the retrieval engine to find the most suitable case to a query, similarity measures must be in place. In myCBR each single attribute has a seperate simliarity function, depending on what type of attribute it is. The user may also create new similarity functions and create an approriate name for the function. The similarity function may be altered in a variety of ways, including different mathematical formulae and weighting of certain occurences. Figure 2.1 is an example of how the similiarity of a float attribute is measured. In this example the difference is calculated polynomically, where quite similar results are rewarded highly but this reward rapidly diminishes as the difference grows.

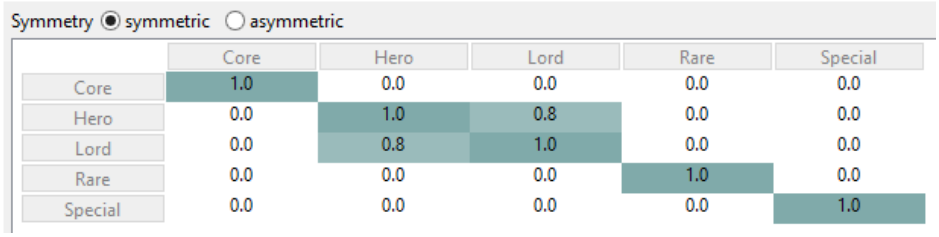

Figure 2.2: Symbol similarity

Figure 2.2 on the other hand is an example of how similarity between symbols are measured. "Hero" is deemed quite similar to "Lord" by 0.8 out of 1.0, whilst the rest are deemed completely dissimilar. These similarity functions are part of the concept similarity function, where the similarity function for each attribute can be weighted individually. Figure 2.3 shows how the similarity function defined in Figure 2.1 is weighted 0.5 (CostPerModel), and Figure 2.2 is weighted at 2.0 (ArmyType). Thus the similarity score will be weighted 4 times higher if the conditions of Figure 2.2 are met compared to that of Figure 2.1.

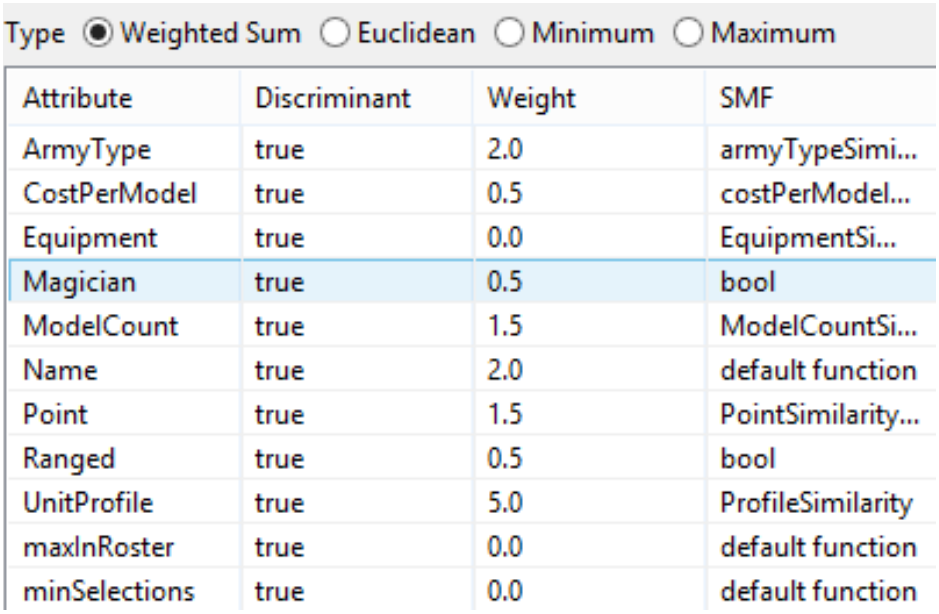

Figure 2.3: Concept similarity

To perform a query using the retrival engine, the user has to specify which case base this query should get the cases from and which concept to use. The similarity measures of the concept are used when comparing the query to that of the instances/cases in the case base.

2.4 Existing WFB army list systems

There are a few existing tools that provide a user with the capability of creating their own WFB roster, the most popular ones are BattleScribe and Army Builder. These two tools enable the user to rather quickly build complete rosters, save these rosters for later use, and print out these rosters to bring with them when playing the game. Both of these tools are somewhat complete in their functionality, but that functionality is limited to only providing the user with easy access in creating their armies, there is no automation in the process.

2.4.1 BattleScribe

Battlescribe[1] is a tool that was created to provide users with the ability to create a model for rules, units and the limitations of these in a fairly easy manner. It is a very general tool that provides a framework for other users to create more specialized rule sets. The tool gives the users the ability to create rules and data sets for whichever game they want, and then share this with other users. The tool itself does not provide the users with anything but the framework, and then they are able to use datasets created by other users, or create datasets themselves. When the tool is used in conjunction with a WFB dataset it will allow the user to create a roster from all the available units for the specified race within the rules of WFB. This means that the datasets have incorporated the number limit rules of Table 2.2, in addition to beeing bound by rules set by a "master" WFB file that instructs the tool in matters that are similar for all units accross the WFB domain. For instance the master file contains different identities for specifiying unit types akin to Table 2.2, which then enables the tool to categorize each unit and enforce the percentage targets for each unit type. Furthermore the master file specifies the identities of units, with their characteristics, and all the different characteristics needed for the different equipment types.

The tool is very open and flexible, and perhaps more importantly, free. Additionally, as the datasets used with the tool are "open source", the access to all the data within these files are readily available as xml files. This allowed me to create a parser for these datasets and easily extract the information they contain.

[1] BattleScribe - `http://www.battlescribe.net/`

A sample army list created using the BattleScribe roster editor is shown in Figure 2.4.

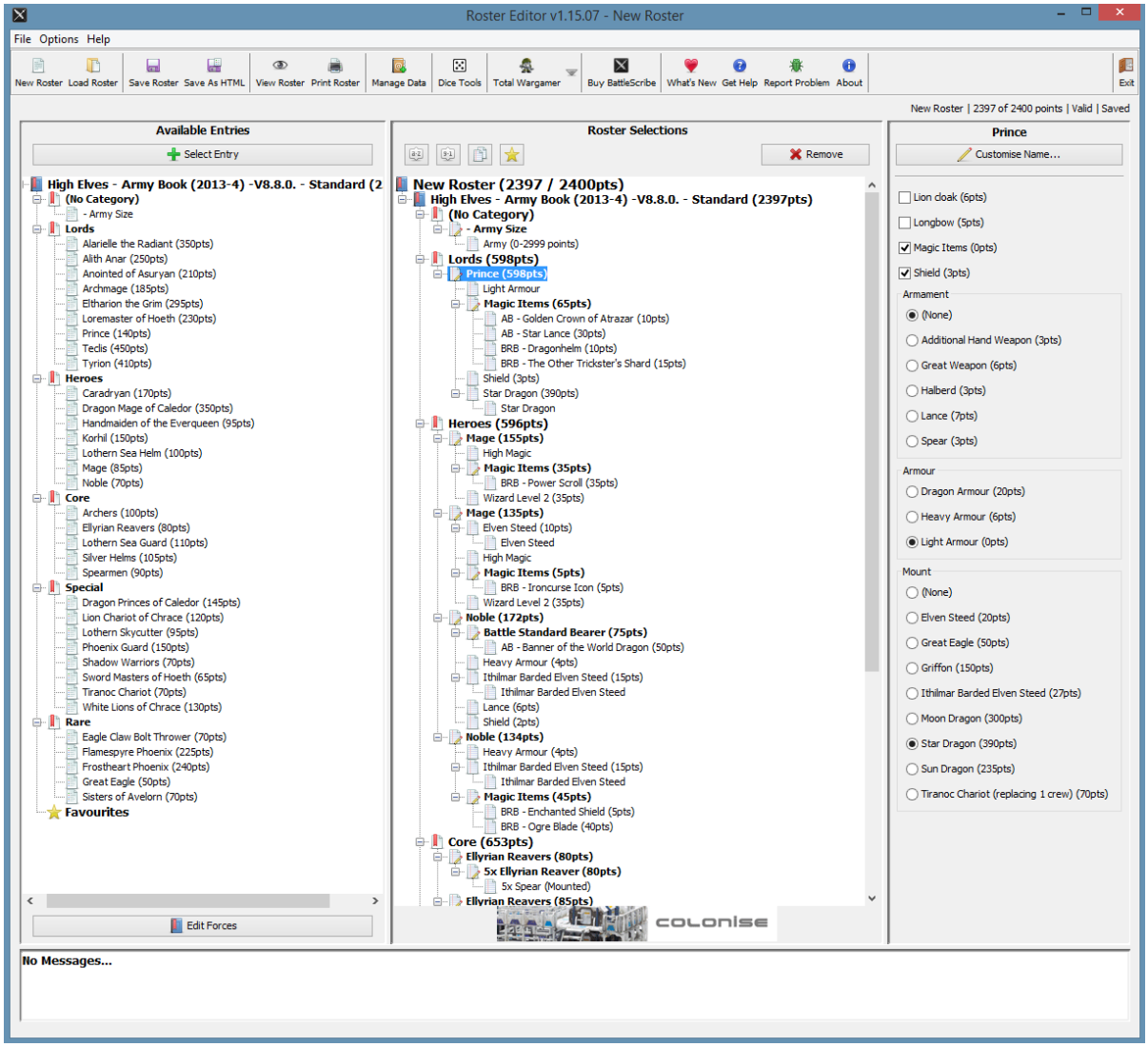

Figure 2.4: BattleScribe Roster Editor - Screenshot

This figure shows how the tool is able to devide the units into seperate categories depening on unit type. Furthermore it shows how a user can pick amongst any of these units to include in their roster, and equip this unit with an assortment of different equipment available to this unit. The tool will also warn the user through the text box at the bottom whenever some point limit is too high, there are extra units beyond what is allowed in some category or other similar boundry breaching rules.

One downside to this tool however is that due to the importance of specific rules in WFB, this tool may provide insufficent information. In most cases the creator of the datasets have provided information as to what the rule is called and where the rules are explained in the rulebooks, but they cannot go into detail about the rules due to copyright restrictions. As such the final roster will contain all the rules for each unit listed, but these rules are not explained.

2.4.2 Army Builder

Army Builder[2] is another tool that enables the user to create army rosters. But this tool is a more contained and restricted in its functionality. Similarily to BattleScribe it is a framework to build models for rules and units, but the framework itself is locked behind a paywall. This instantly makes this tool less desirable to look at compared to BattleScribe. You can get a sense for the tool in the demonstration mode where you are limited to rosters of maximum 3 units, and you can create new datasets with its editor similarily to BattleScribe. However the datasets are structured quite differently, making them harder to parse without using the tool itself.

The upside of Army Builder is that the rules are all accounted for, as the creators have made a partnership with Games Workshop, the creators of WFB. This means that the end user can rely solely on the rosters they create with this tool and not have to worry about rulebooks, except for off instances when the rules may need to be put into context with something else.

2.5 Constrain satisfaction problem

A constraint satisfaction problem (CSP) is a very flexible approach to problem solving. A user only has to specify a set of constraints and a solver then attempts to create a solution that matches all these. Usually this is done by modifying a solution until this solution fulfills the constraints. A CSP is domain independent, yet it works with complete models and finds a solution for these models. CSP

[2]Army Builder - `http://www.wolflair.com/index.php?context=army_builder`

solvers are usually algorithms that take advantage of certain properties or mathematical equations which enables them to work quite fast and explore a variety of solution. There is one noteworthy flaw, it is difficult to assess whether a solution created by a CSP is better or worse than a previous solution by the same CSP, as the solutions may be very different even in the same domain. This is partly why CSP solvers usually work with finding *a* solution, rather than an optimum one. There are three main categories for CSP, those that use search, those that use inference and those that utilize both search and inference. Marling et al. [2002], Russell and Norvig [2009]

- **Search** The solving methods is to search through different possibilites, and adapt the solution until a valid result is found. This can for example be done through back-tracking or local search algorithms.
- **Inference** There may be several sub problems that can be solved seperately, but together can create a solution to a problem. Arc consistency is one such example.
- **Search and Inference** A combination of both approaches may be benefitial in certain domains, but requires extensive knowledge.

In this work, search algorithms are the most interesting due to the very nature of what we intend to do.

2.6 CBR and CSP

There have been several successful implementations of systems utilizing both CSP and CBR in solving specific problems. The approaches are varied, following are two separate examples.

There are the systems that mainly use CBR as the core system, and CSP for adaption. Qin and Wei [2009] created a system to work with product configuration. The customers could have several needs, and thus want a product configuration to best serve those needs. As it is likely that another customer could want a similar product configuration, it would be beneficial to store solutions to reuse in later queries, which a CBR system is very well suited for. The core idea of this work was to create a system where each case would be a solution to a CSP, a valid complete solution. The system would find cases similar to the desired configuration, run a pass of the CSP algorithm "min-conflicts" to alter the solution into a suitable product configuration and store the solution for later use. The synergy between the CBR and CSP subsystems are seemless and they complement each other.

There are the systems that use CSP as the main solver, and secondarily use a CBR method to adjust for any shortcomings in this CSP iteration, Sqalli and Freuder [1998] proposed such a solution. This work attempts to look at a solution to interoperability testing of protocols. In this domain the knowledge may be incomplete or flawed, as the experts writing these protocol specifications may not include certain important aspects. With an incomplete protocol specification two systems interacting with each other may handle certain aspects of the protocol differently, which could produce problems. Interoperability testing is the act of attempting to discover these flaws. This is usually done manually, but in this paper the method proposed is to implement a CSP approach to allow for easier testing. Each subproblem is reduced to a CSP graph, which ultimately are combined into a set of subproblems that together test the entire system. The idea is that each CSP graph are implemented as needed as opposed to a regular test. With enough information the CSP would be able to handle these subproblems, but if the CSP cannot determine the outcome, a proposed CBR solution should attempt to look at previous cases of CSP solutions and account for missing information. The CBR part of the system was not yet completely implemented but the idea seemed promising.

The avid reader might notice that the approaches are quite similar, both systems store cases of previous solutions to the CSP, but one of the systems try to use the speed of the CSP solver as the main system and only include CBR when necessary. The approach the design in this thesis will look at is closer to the first example.

2.7 Explanation aware systems and explanations

Explanations are about conveying information regarding a certain event, object, situation or action. This information is an attempt to make another person more aware of the implications and reasoning behind this occuring event. When an explanation is done with the intent of letting the reciever of this explanation more knowledgeable, the explanation should enrich this persons experience in some way. A person might inquire as to why one outcome is prefered to another, which promptly would inspire the acting person to present his reasoning for this outcome. "I want Manchester United to win the game because I don't like the way Chelsea plays." More commonly though, an explanation will be regarding actions and events that have or are suggested to happen. "Would you like to buy this book? It is similar to your previous purchase.", "I want to order the hamburger, the last one I got here was delicious.". The domain of computer science use explanations in systems where the user should be made aware why a certain suggestion or action was taken, but the capability of a computer to explain itself is a bit more narrow.

In explanation aware systems there are five main categories of useful explanations, Roth-Berghofer [2004]:

- *Why explanations* will attempt to justify to a user why the system would propose this specific result. This can be further specified into cause and justification explanations. A cause explanation will explain the reasoning behind why this result occured, whilst a justification explanation will attempt to explain why the proposed result is reasonable in this current state.

- *How explanations* are more thorough variants of *why explanations* where the user is presented with the reasoning trace behind the outcome result. The user will learn *how* the system ended up with this specific result.

- *Purpose explanations* will attempt to explain the reason or purpose of an objects existence. They are answeres to questions such as "What is ... for?" or "Which purpose does ... serve?".

- *Cognitive explanations* are yet another special case of *why explanations* where the system will attempt to predict or explain the behaviour of another system. This is done by looking at the borders of the assumptions, limits and goals the system is specified to work within.

- *Conceptual explanations* are tasked with making the user associate unknown information with what is already known. They will formulate answers to questions such as "What is ...?" or "What is the meaning of ...?"

All of these except the Conceptual explanations describe various scientific answers to questions we might naturally enounter in the physical world. They utilize static knowledge about a certain domain that the system has been given, and they attempt to convey these facts to the user. The Cognitive explanations on the other hand are about explaining the behaviour of a system, they attempt to interpret social behaviour that may not be naturally defined.

The soundness of these explanations are categorized by Tintarev and Masthoff [2007], Roth-Berghofer [2004], Sørmo et al. [2005]:

- *Understandability.* The explanations are required to be understandable, both in its content and as to why it is significant to the current domain. The users of the system should be assured by the facts it presents, and thus be enriched with new information. The system should avoid telling the user what is already known but rather provide new information that could be put in context with what the user already knows. This could be done by letting the system infer what the user knowns through the initial questioning process. With relevant enough dialog between the system and the user,

the user should be assured that the system is able to provide information that is relevant in regards to the users own knowledge. Furthermore the language used should be linguisticly sound and feel natural. Ideally then, the system should behave as if it was a human with expert knowledge in a certain domain.

- *Sufficiency.* The system must be able to answer what the user asks it about, thus it is required that the system possesses enough knowledge to be able to do so.

- *Fidelity.* To best be able to accurately describe what the expert system actually does, the explanation system has to be based on the same knowledge base the expert system uses in its reasoning.

- *Transparency.* The user should be able to influence the system where appropriate in such a manner that the user better understand why a system acts the way it does. It should be clear to the user how the expert system is reaching its conclussions.

- *Scrutability.* If a system allows the user to customize what kinds of results he is interested in, the user is more likely to be satisfied with the results. When the explanation or suggestion feels relevant for the user, the users confidence in the system increases.

- *Trust.* The trust a user has in a system affects how likely he is to use it again. This is directly linked with how the user perceives the explanations he gets from the system.

- *Persuasiveness.* If the system is able to point to previous or similar results, and those results strengthen the explanation, the user is more likely to believe in the system.

- *Effectiveness.* This is a measurement that rates how relevant the explanation is to the user. This requires either the system or the user to rate the result beforehand so it is possible to compare the rating results before and after to see if the user agrees with the system. This would be most useful in a recommender system where the use of predictions could help the system learn.

- *Efficiency.* When conversing with a system to reach a certain set of results, the efficiency is measured by how fast the user finds what he percieves to be the best result. This endeavour is taken to improve visability when presenting multiple results. The research into this has not proved fruitful as of yet.

- *Satisfaction.* How satisfied a user is with the system depends on how well they perceive it. Thus this depends on the points above, if the explanation is deemed satisfactory the user would be more likely to use the system again. The presentation and the goodness of the explanation impact the users satisfaction.

There are also some points that not directly target the goodness of the explanation yet still affect how a user may perceive the system.

- *Low construction overhead.* Implementing a explanation system in a expert system should not affect the construction of the expert system. Ideally it could be integrated with or replace a certain part of the design of the expert system.
- *Efficiency.* The explanation system should not have a major impact on the performance of the expert system. This is partly linked to the construction, where if the expert and explanation systems are closely linked the act of explaining could have a very small impact on runtime performance.

In this design the intent is to use a set of justification explanations. The user will likely want to understand why a certain army composition was created to match the criteria set by the user, which is precisely the purpose of justification explanations.

2.8 Lessons learned from the interviews

Two seperate semi-structured interviews were conducted with experts in the domain of Warhammer Fantasy Battle. During these interviews the subjects were presented with the goal of this project, and subsequently questioned about several different aspects of this approach. They were asked how CBR methodology could function within the limits of army list creation and what results this system might be able to produce. They were also questioned about what aspects are most important for a user of this system for it to be worthwhile, and what sort of explanations this system should be able to produce for the user. Furthermore they were questioned regarding game specifics such as what considerations should be made when building an army and how developments in playstyle effects these choices.

2.8.1 Subject 1

Subject 1 is an avid WFB player with ties to the national ETC-team (European Team Championships). He frequents numerous forums where different army list

compositions are discussed, thus he is one amongst a few players helping design the best possible army lists for various tournaments where the ETC-team competes.

To illustrate how difficult it is to weigh a unit into some sort of measurement system, he explained in detail how the game is played with numerous different strategies. For example, positioning is a very important aspect of warhammer, thus Movement is an important stat. This is both because you ideally want to be in range of the opponent to inflict damage through either melee combat, or through ranged weapon usage or magic. Yet by positioning this unit close to the opponent you give him the ability to move close enough on his turn to inflict damage onto you. As each single unit has different movement stats depending on the sort of unit it is, you may be unable to avoid certain rush-to-the-enemy tactics and thus need to devise a strategy for this. One posibility is to create a ranged heavy army and a few strong "tanks" to soak up damage in the front, while the heavy hitters are standing in the back uninterupted. Yet this may again be countered by having very movement capable units that can flank the less durable ranged units. In essence, the weights of the stats depend very much upon what kind of strategy you want to primarily rely on. Though the overall optimum strategy would be something that can deal with most tactics, and not have any major downsides.

Furthermore, a unit by itself may be underwhelming, due to poor statline or capabilities, yet when paired with other units may look far more impressive. This element is called synergy, and is one of the most important considerations when building the army.

There are also a number of different special rules that may significantly alter both the capabilities of the unit, or the entire faction itself. One such example is for the faction Daemons of Chaos where the units are "unbreakable", which means that they have no fear of standing alone, they are immune to psychology. The impact of this is that when playing this faction you do not need to fear having units spread around the battlefield and having them flee from battle due to "panic".

A unit may also seem underwhelming at first, but as the metagame changes, new options for how to use this unit may emerge. The impact of units are not set in stone and may change due to how they are percieved. Yet again there are units that have a very good statline and are quite cheap, which makes them immediately categorized as good units. Yet at some point diminishing returns kick in and an obvious good unit may lose some of its value when included too much. This may be due to special capabilites that do not "stack", for example a unit with an aura increasing the Initiative will be valuable, but there is little need for more than one such unit in the army, unless you intend to battle on multiple fronts.

Core units are often regarded as cannon fodder, they are quite weak by themselves and are often sent in to die so that other stronger units can remain unharmed. As the rules specify that you need at least 25% core units, it is often ideal to spend just that amount of points on cores and try not to spend more than required.

Due to how complex the actual game is, creating an army is no easy feat. The suggested system, one with the capabilites to suggest armies to a user, may very well work. Yet the results it produces may not be higher than novice level, it will likely struggle with finding good compositions. This is due to the simplifications needed to devise such a system. You have to make sure to stay within the rules, and pick units to meet those requirements. A suggestion is to let the user specify what kind of army they would want, defensive, offensive, magic heavy, ranged heavy and so on. Yet as the main purpose is reusing previous cases, some of these features are already set, it may be wise to find varied cases for the data. Another approach would be to look at unit sizes and base the system on that, pick random units that stay within the rules, give it some equipment and so on. The end result may not be great but it would be a legal army. A unit may be drastically changed depending on what kind of gear it is given, as such gear combos are also quite important. Synergy is also an important part of equipment options.

When selecting the units, the unit size may pose a challenge, as some units don't have a upper cap on the unit size. Overall a good strategy would be to keep the unit sizes rather small and equip the unit with some gear. The difficult part would be to get everything to add up together and fit together.

As this is a CBR system, there will be a lot of data already set, the intent is to find the most similar unit in the case that matches what the user sets in the query and work from there.

The approach to this would be to categorize the units into classes, units have different types, infantry, cavalery, warmachine etc. and they have different stat lines, so there are a lot of categorizations you can set about the unit. Which seems similar to what you intend to do.

You need a general, some core units, and a few special units. From there you can just fill in the missing spots with whatever seems most suitable. Focus on keeping it simple at first, make sure you have all the requirements met and try to specialize from there.

As the idea behind CBR is to reuse when possible, I intend to have structure of the army set already by the original case, and make alterations where needed. Meaning swap units with the most similarity to the desired units and then focus on meeting the WFB rules.

Dooing this would take a lot of the random factors out of the equation, the focus on trying to optimize the stats by the statline and special rules would

pose too difficult. Yet when comparing similarity between units, even though the stat line may be similar, there may be a discrepancy in points spent and similar complications. It may be difficult to find the most similar unit. Trying to put a value on a unit is something that has been discussed heavily, and it depends very much on the environment and situation you use the unit in. This complexity is what makes the game fun, but also what makes evaluation of an army difficult. You have to test the army in practice a few times to see its strengths and weaknesses. Some factions also in general perform better versus some factions than others, and may require very different tactics depending on the opposing faction. Yet when creating the army you often do not know what the opponent will end up playing, thus you need to maximize your chances by creating the best overall army.

As stated earlier, even though a unit may be similar to another in stats and cost, some special rules may completely alter what this unit is capable of, which again determines the value of the unit, which thus will make it difficult to assess the unit in comparison to another.

This is why I intend to give suggestions to the user, and possibly reiterate if the suggestion is not a good one

Giving the user the possibility to refuse an army may be a good idea. It may be wise to let the system learn from its mistakes as well. It could be as simple as letting the user tell the system to not include a few particullar units and take it from there.

Taking a look at what already exist may provide some ideas as well, Army Builder and BattleScribe.

Yes, in fact I am currently extracting the data used in data sets from BattleScribe

The functionality of these programs is something you have to mimic in order for the system to be viable, it is very easy to create the army, and make sure the army stays within the rules.

Testing the system could also prove difficult as it is hard to distinguish a very good army from a mediocre one if you don't know what to look for. The game is not solved in any way, thus the outcome of similar matchups are not determined beforehand. In chess for example you can somewhat predict how the outcome will go depending on the moves you make, the same is hard to do for such a complex game as Warhammer, with its hundreds of rules. You cannot simulate the game, and thus it is hard to automate some sort of testing sequence to tell if the proposed army is decent or not. It may be viable to run the program a great number of times and finding results the repeat themselves, and thus maybe get a grasp of what is good.

Explanations in this system may be as simple as explaining why certain units are in the army, and how it was changed to reach the users criteria, but giving

good explanations may be hard. The user intuitively know a lot about Warhammer already, so the explanations either have to be very detailed, or just simple rules to explain what happend.

Overall the CBR assisted army suggestion system sounds promising, but it is doubtful that the results the system are able to suggest are good enough for experienced players. Experience is a very important factor when creating an army, and affects what choices you know to be decent and which are not.

2.8.2 Subject 2

Subject 2 is an experienced WFB player that previously was one of the driving forces in WarTrond, a WFB community in Trondheim. He still plays on occation, and was recently playing in tournament hosted by the WarTrond community.

The idea of creating a system capable of interchanging units of a previous army based upon user preference is something that sounds very interesting. Yet it may prove difficult to distinguish these units from eachother easily. There are a number of different factors involved here. Looking at just the stats in the beginning seems like a very viable approach. Ignoring the special rules is a necessity for this system to be achievable. As long as these rules are accounted for somewhere, the user should have sufficient information. Reiterating the solution until the user is satisifed with the solution and the unit it contains sounds like a good idea.

There should be a way to distinguish units of the same type based on their combat capabilites, such as how some infantry have ranged weapons and some do not. Giving the units an attribute specifying whether they have melee, ranged and magic properties should be sufficent.

Looking into the data files of BattleScribe and extracting that data sounds like a decent way to get all the required information, but you may need to extract away superfluous information at first and maybe incorporate this when the model can support it.

Focusing on only the army creation aspect of the game is the correct choice, it would be far too difficult to account for the gameplay itself. This is precicely the reason why no simulation tools exist. The roll of dice should be easy enough to account for, but units may completely change behaviour and capabilities by simply equiping a certain item. Synergy is a very important aspect.

You have to focus on optimizing the army based on what you want to do, you cannot do anything about the opponent or the variety they may pose. Thus you need to create the best army for your particular strategy. There are certain factions that will always have the upper hand against other factions and so on. Different scenarios may also completely change how the game is played and how

the result of a matchup could end. It would be wise to just focus on what is general about the game, and ignore all special cases regarding scenarios, otherwise this task will be far too immensive.

Synergy is such an important aspect of Warhammer, yet to achieve synergy you have to know so much about all the different possibilites certain compositions may have. This is something that would be far too difficult to incorporate properly, you would have to include information that may not intuitively give the best army, certain units only work in conjunction with others. Some lists can go from useless to good by changing just a few units or equipment. Thus mostly novice players are going to benefit from this system, they get a good starting point and can work on it from there.

Working with another layer with more specified information could be a good approach to ensure better solutions, but that would mean specifying a lot of rules and occurences that rely on the creators own experience. This would pose a lot of work. Start with the more general Warhammer.

There are tools that would allow checking if the units of the army are considered good by tournament players, you input all your units and get a score from 0-20, where lower score signify taking the units considered very good and have good synergy, and higher score define taking decent units and not having too much synergy between eachother. But this is based on a seperate ruleset than the standard Warhammer one, which makes this tool less interesting when in a non-tournament setting. This ruleset aim at equalizing the factions and giving them similar chances of working well. Some of the standard rulebooks are just too good or too weak. Some of the factions are also balanced around special rules, and without them they would be very unbalanced. The statlines of the units may be overall very good, but certain rules inflict random misfortunes for example.

Some factions also have a lot of variance amongst the unit line, whilst some overall have very similar units. Working with only a few select factions at first is a good idea, otherwise the amount of data may be overwhelming. Dwarfs and High Elves are a decent starting point. To broaden the scope a human and daemon faction could be included. There are also a few factions that are just so dissimlar to the rest of the factions that they are not worth looking at, such as Skaven, Lizardmen, Ogres, and Orcs & Goblins.

Leadership is a very important stat, the morale of the units affect how likely they are to flee and thus how effective they can be in battle when their numbers start to diminish.

End Times is another seperate ruleset that included a new type of magic, which allows for summoning of units. This completely changes the game, enforcing the position of magic in the game.

There are also a wide variety of spells that can completely change the outcome

of a fight, for instance a spell that kills a unit based upon roll of dice in correlation to their strength, meaning if they have low Strength they are very likely to die from this spell. Similar spells exist for Initiative and so on. Magic is a very imporant aspect of the game, you either have to use magic or use equipment and units to cancel out the effect of magic. Yet all this magic is hard to quantify, they are all very special instances. Making them difficult to account for in numbers.

Some units have identical stats but yet may work different due to what weapon they use. Elect to try and spend about the same amount of points on two seperate units with similar stat is a decent approach to ensure that there is enough variance in the army to account for the subtle differences.

Try to not change category of units when swapping them, for example replace cores with cores. Yet there is no massive distinction when exchanging some special with a rare or hero, although it may alter the balance of the groups. Just make sure to uphold the point limits of the different groups. There are also some limitations in duplicates in rares and specials, which should be very easy to incorporate.

To assess the similarity it would be wise to focus on unit type, and also make a distinction on if a unit is ranged etc. The special rules that defines other capabilites can be ignored to simplify. Let the user make choices that may be too hard to assess computionally.

Presenting the choices made by the program, and allowing the user to make alterations where he deem necessary is a good approach to make sure he understands why the presented army looks the way it does, and quickly allow for changes if he disagrees with some of these choices.

Letting the users define what sort of strategy they want to use and what sort of units they want is important to let the system make the right calls. This also allows the user to speficy what units the has available and which he does not, reducing the possible set of units somewhat. Furthermore, letting the system refine a previous case, or opt not to if the case matches would give the user the option to make alterations wherever he would like and let the system run again.

Explanations can be based around statistically ratify why a unit was chosen over another. Weighting the stats and explaining that certain units are more powerful due to having more of a particular stat than another etc. Very difficult to create good explanations without extensive knowledge.

2.8.3 Summary of interviews

Both subjects agreed that CBR might not be able to fully encapsulate the nuances and special properties of the game. The complexity of the game is massive in both number of rules, variations in units and synergies between units. Where rules and synergies are what make the game interesting, but is also the most difficult part

to incorporate in the proposed system. Thus it was suggested to solely focus on the variations in the units themselves, namely the differentiations in numeric values such as the attributes the units posess. Unit type, ranged capability and magic capability is also important measurements to distinguish the units. The end result would most likely be a system ill fit for the more experienced user, but it could prove useful for novice players. Both interviewees mentioned that it could be wise to focus on details such as users limitations to units and weapondry, and thus incorporating a way to specify what units to include and exclude. The explanation aspect of the system is hard to account for in a good way without extensive knowledge, simple justifications about why certain units were picked, or why they were exchanged with other units may be sufficient.

Chapter 3

Architecture and Implementation

In an attempt to accomodate the different aspects of WFB into a CBR system, several considerations need to be made. How can WFB army lists be reduced into simple comparable cases? What aspects of WFB army lists can be simplified or even ignored for this system to still function as intended? How can we ensure that we have all the relevant data for the domain knowledge?

In the following chapter these aspects are explored and accounted for in the design and implementation of this system.

3.1 System design and implementation

The final system is a result of several different components working together to create a CBR application. The system as a whole is based on beeing somewhat general, thus a decision was made to find external data files for WFB to build the domain knowledge.

myCBR is used to create a model for the cases and the similarity functions. It is also used as a tool for containing the cases in different case bases and it has a very strong presence in the retrieval phase. Its retrieval engine does the heavy lifting in the retrieval phase. The user is presented with options in what to include in the query, the query is run with this data and the engine returns some results that are then used in the adaptation phase.

However to use the model the case bases need to be populated with some data. The following are some of the case bases: Complete armies, all the units of each faction in seperate case bases to specify the posibilites for each faction,

the actual units used in army lists, and unique profiles for each different unit. Thus domain knowledge was needed, this has been provided by parsing complete data files for each faction and extracting the useful information. This is further explained in Chapter 3.3 and Chapter 3.4.

It is important to note that all the cases in the case base are assumed to be good ones. This means that they should have been tested extensively in actual battles. All the cases are also specified under a 2400 point limit, this is because the standard tournament point limt is set to this number. The final cases used in the model fulfill these requirements.

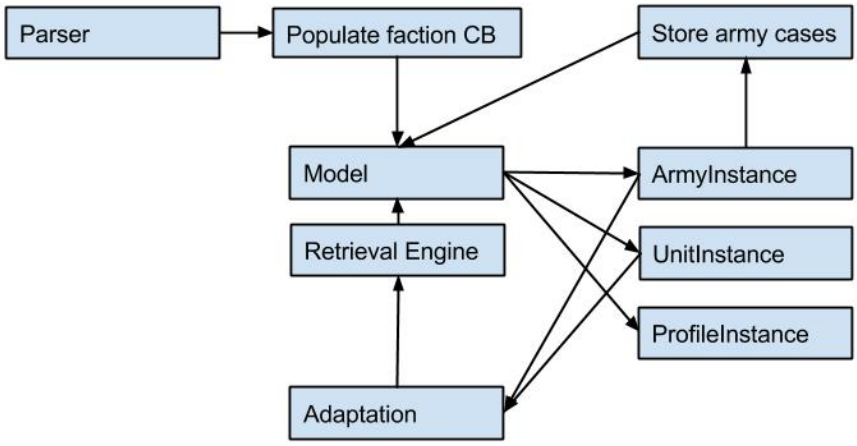

Figure 3.1: Class diagram

A simplified look on the system can be viewed in Figure 3.1. The parser is what primarily provides data that is used to populate the faction case bases of the model. It is also possible to add new army cases and this is done primarily by interacting with the model. Where the instances in the faction case bases define what units can exists for the faction. Any new units that are created will be in the image of one of these instances, copying all attributes that are similar and updating those that are specific for this unit. It will also reuse the profile, which guarantees no extra instances are created. This also ensures that the the same units will only differ on a few select fields.

In order to easier access the model instances, three seperate classes were created to match the model, ArmyInstance, UnitInstance and ProfileInstance. These clases simplify updating and retrieving information from the model. They do so by working with a single instance of the appropriate type, and use methods

defined to match the attributes in the model. They handle the myCBR SDK interaction, which is cumbersome to use repeatedly in instance interaction. These classes also simplify how the queries are run, due to various copy methods. For example, you might want to retrieve the faction of a army case, which when using the SDK directly would imply first finding the correct Instance, then create a attributeDescription for the attribute you wish to find by name, and lastly you have to get the matching attribute using this attributeDescription. Instead you can instantiate a ArmyInstance object with the Instance as an argument and simply call getFaction().

The UnitInstance class also contain some fields that duplicate data from the Instances, this is so it is possible to alter the UnitInstance data for use with adaptation, but not alter the Instance itself until the final solution is found.

The system currently has working retrieval and reuse (adaptation) phases. Revision and Retain phases are currently not implemented, this is due to the ambition in trying to automate the Revision phase. This was found to be unviable by both interview subjects in Chapter 2.8, and manual revision is not suitable as it would entail playing the game with the proposed army numerous times to verify it. It is possible to store cases and as such Retain is somewhat functional, but is not part of the current CBR loop.

3.1.1 Retrieval

The retrieval phase consist of selecting what to include in a query and running the query. In this system the options in what to include in the query is selecting one of two factions, either Dwarf or High Elves. The model also allows for not specifying the faction, and as such get the best match by points and units. This functionality is not fully working in the code due to having specified only loading the units from the current faction for use in the query. The next option is to specify what the desired point limit for the created army should be. This again is possible to not specify in the model and perform retrievals, but when the adaptation phase does not know how much points it has available it cannot function as intended. The last option is to specify what units the suggested army have to contain. You may elect to not specify any units, or you can choose any combination of units that are legal. You may for example select an army containing almost only cores, yet there is a rule in Warhammer stating that you must have a commander, either a lord or a hero. Thus when you are about to select units that would make it impossible to add a commander for the current pointlimit, you are prohibited from doing so. You are also unable to add more of a unit than what is specified by the maxInRoster for that unit, this applies mostly to special and rare units per Table 2.2. The same table also specifies limits for the different groups, which again is taken into consideration in what

units you are able to add. You are simply prohibited in making queries that do not uphold the game rules.

After you have specified your query, the retrieval engine is run and the results are returned. The best result is selected for the adaptation phase.

3.1.2 Reuse - Adaptation

The adaptation phase is intended to work somewhat similar to a CSP, thus using some concepts from this method.

The adaptation phase will intially check whether the original case was good enough. It will check whether the point limit set is within the bounds of the original case, and that there are no extra points to spend somewhere. It will also check whether all the units specified are present. Which due to how the unit similarity is calculated may cause the retrieval engine to return an identical match when there are units missing if there are a multitude of one particullar unit. Say you for example specify in the query that you want to have 3 quarreller units in your army, the best case may only contain 2 units yet state a perfect match. Thus a manual check to see whether all the specified units are accounted for is required to be sure.

The first step of the actual adaptation is to replace the most similar units in the original case with those specified in the query. It will do this by creating a new case base and populate the case base with the original units of the case. Next it will run retrieval queries for each of the units in the desired unit list, find the best match, place the unit to replace in a deletion list and add the new unit to the army. Once a unit is placed in the deletion list it will be excluded as a possible replacement even if it is the best match. This is done due to some limitations with the myCBR SDK, and it ensures that all the units are accounted for. After all the desired units are accounted for in the new army, all the units in the deletion list are moved to a global removedList, thus we still know which units have been removed from the case and can reuse them later if needed. On the off chance that there are no units to replace, the desired unit is simply added to the new army and no replacement is made.

Next the adaptation phase will attempt to add cores untill over the limit set in Table 2.2, starting with any units that may have been removed during the replacement phase. If there are no cores to re-add from the removal list, it will add a new unit of from the set of the already selected cores in the army, it will not attempt to add any different core unit that is not already present. This is to ensure that the integrity of the original case is maintained.

Next a check is performed to see whether the current army is sufficent. If all the rules are followed and the point limits are within the bounds.

The next phase follows where the amount of points spent on cores are mini-

mized, ideally reaching the target and not spending a point more as per interview subject 1 in Chapter 2.8.1. First the model count is reduced for random cores untill it is impossible to reduce further either due to reaching the minimal number of the size of the unit, or untill the point limit is reached. Next the algorithm will try to remove any excess core units, choosing at random. Lastly it will attempt to remove any gear points spent on the unit.

Another check for sufficency then follows.

The next step is to ensure that all the point limits are reached as per Table 2.2, where the strategy is reducing count, then removing gear points and finally removing units themselves. This is with the intent of maximizing the points spent on these different groups as opposed to minimizing them with the cores. When gear is removed, a random multiple of 25 pts is choosen, alternatively the remaining gear points if less than 25 remain.

Now every category are within the upper limits of what they are allowed to be. If the army still has too many points spent, the algorithm will first attempt to randomly reduce the count of either a special or rare unit. If this is insufficent, it will attempt to remove units that are not core. It will elect to remove units that will bring it within the point target or the cheapest unit. This beeing units that cost just slightly above what is needed to bring the army within legal limits. The result of this strategy is that the impact on the army is kept minimal untill a unit presents itself that hit close to the target. If there are no more units that can be removed without interfering with the desired units, the last step is to try and remove excess gear of random units untill the army is within legal bounds. Which due to how the intial query is structured should always give a valid result.

On the occation that the army either lacks a commander or have points left to spend, the algorithm initiate the final phase to include new units untill all the points are spent. It will first try to add a commander if the army doesn't have one, selecting amongst the removed units first. Then attempting to add a new unit that previously was in the army, a gearless unit that is able to fit the point limit where a geared removed unit would not. Finally if none of the commanders in the original case are applicable, a random commander is selected that are within the point limits.

The next step consist of trying to readd units previously deleted if possible while mainaining the limits. Otherwise it will start to randomly add units that are part of the current army. No new units will be added that are not already part of the army. The idea with this is to ensure that the spirit of the army is not altered by adding units that may not fit well with the theme of the army. The final step of the algorithm is to increase the model count of a random unit if there are points to spare but not enough to add more units. It will focus on the units that are not core initially, but if there are no other units to increase the model count but the cores, it will attempt to increase the count of one of those

as well.

3.2 Modelling with myCBR

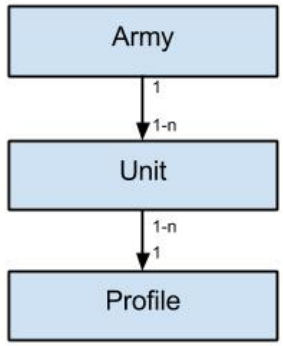

Figure 3.2: Model

Several different iterations has been done to end up with the final model used in the system, which can be viewed in Figure 3.2. The current model has the top concept Army, with Faction, TotalPoints, Units and UnwantedUnits attributes. Faction is defined as a symbol, where the model has a complete list of all the different available factions. TotalPoints is defined as a simple float attribute. Units is defined as a concept attribute with the posibility to add multiple instances, thus allowing it to act as an army list. UnwantedUnits was intended to be a negatively weighted Units, containing the same army list, but with the implication that any single unit found in this list would provide a negative similarity rather than a positive one. This functionality does not work as intended so UnwantedUnits is not used as an active field in the final model.

The army similarity function (3.1) is defined by the weighted sum of:

$$5.0 * Faction + 1.5 * TotalPoints + 10.0 * Units \tag{3.1}$$

Where Faction similarity is a defined symbol similarity where Bretonnia is considered 0.5 similar to Empire, Dark Elves 0.5 similar to High Elves and 0.3 similar to Wood Elves. The rest of the factions are only considered similar to themselves. This similarity is based upon what seems right rather than some correct measure. TotalPoints is a polynomical float similarity function with 2.0, similar to Figure 2.1, but with range up to 5000 points. Thus small variations

in points spent is not considered very different. Finally Units similarity is based upon set similarity as there are multiple units in each army. This is done by matching each unit by (3.2), where a single unit is allowed to correspond to several different units in the query case. The average similarity of all the units similarity is the final result.

The Unit concept is defined with ArmyType, CostPerModel, Equipment, Magician, ModelCount, Name, Point, Ranged, UnitProfile, maxInRoster, minInSelections. Where ArmyType is a symbol attribute, containing all the types a unit may be defined as, see Table 2.2. CostPerModel is a float, and defines what each model may cost. Equipment is a concept attribute not used actively as a field. Magician is a boolean attribute and defines if the unit is a magician. ModelCount is an integer defining how many models this unit consist of. Name is a String, defined as the unit name. Point is a float defining what the units total cost is. Ranged is a boolean attribute defining if the unit is ranged. UnitProfile is a concept attribute and corresponds to a Profile Instance. maxInRoster and minSelections are floats not used in similarity but are needed for adaptation.

The unit similarity function (3.2) is defined by the weighted sum of:

$$\begin{aligned} &2.0 * ArmyType + 0.5 * CostPerModel + 0.5 * Magician + 1.5 * ModelCount \\ &+ 2.0 * Name + 1.5 * Point + 0.5 * Ranged + 5.0 * UnitProfile \end{aligned} \tag{3.2}$$

Where ArmyType similarity is defined as 0.8 between lord and hero, and the rest are only similar to themselves. CostPerModel is a polynomial similarity with 2.0. Magician and Ranged are boolean similarities. ModelCount and Point are polynomials with 5.0, meaning they lower the similarity score much faster than CostPermodel for example. Name is a strict one-to-one string comparison. UnitProfile is the similarity defined in (3.3)

The Profile concept is defined with A (Attacks), ArmourSave, BS (Ballistic Skill), I (Initiative), Ld (LeaderShip), M (Movement), MR (Magic Resist), Name, S (Strength), T (Toughness), UnitType, W (Wounds), WS (Weapon Skill) and WardSave. This is a mirror of what is defined in the Profile in the data sets parsed. Thus all the attributes but Name and UnitType are floats, Name is a String and UnitType is a symbol attribute.

The profile similarity function (3.3) is defined by the weighted sum of:

$$\begin{aligned} &1.5 * A + 0.5 * ArmourSave + 1.0 * BS + 1.0 * I + 1.0 * Ld \\ &+ 1.5 * M + 0.5 * MR + 2.0 * Name + 2.0 * S + 1.0 * T \\ &+ 3.0 * UnitType + 1.0 * W + 1.0 * WS + 0.5 * WardSave \end{aligned} \tag{3.3}$$

A, BS, I, Ld, M, S, T, W, WS are all polynomials with 5.0. Whilst ArmourSave, MR and WardSave are polynomials with 2.0. This is because the differences in primary attributes are what has the biggest impact. Name is a one-to-one string comparison. UnitType is a symbol similarity function with Cavalry 0.8 similar to Infantry and 0.5 similar to Monsterous Cavalry. Monsterous Infantry is 0.5 similar to Infantry. All the others are only similar to themselves. These weights are educated guesses based on what seems right.

A more complex model which incorporates more details, but was ultimately not used due to complications can be found in Appendix C.

The myCBR retrieval engine uses these similarity function when running comparisons between army cases for example, where multiple different units are measured in a variety of ways and the enging uses the whole spectrum of similarity functions.

3.3 Data parser with StAX

To achieve access to all the relevant data for the races of WFB, a parser was written to read the data files originally intended to be used with the tool BattleScribe[1]. Due to the sheer magnitude of information that WFB envelops these data files contain several thousand lines of text in the form of XML files. The structure of these files are uniform, thus enabling a general parser for all these files and easy access to their contents. To ensure efficient traversal of the content of these files, the StAX (Streaming API for XML) Java library was choosen over the more common DOM (Document Object Model). When using a DOM parser you effectively load the whole file into memory and manipulate the data within as you wish, which, while effective for small files, makes parsing large files slow and memory consumption heavy. However using StAX you load the file line by line, streaming its contents and execute code based on the events these lines trigger. Instead of reading the whole file into memory and using it as an object, you traverse it once and extract the information on the go.

The parser creates objects in a tree structure identical to the structure in the data files. This was done to ensure that every piece of information was extracted properly and accounted for. The top level of the tree is shown in Figure 3.3, where *catalogue* acts as the root node and is connected to the nodes *entries*, *rules*, *links*, *sharedEntries*, *sharedEntryGroups*, *sharedRules* and *sharedProfiles*. All of these sub nodes are effectively lists containing one or more nodes of the specified type, which then again acts as a parent for a different set of children. Each node, including the root, may have a set of attributes that differ depending on what kind of node it is. The *entry* node for instance has the attributes "id", "name", "points",

[1]Battle Scribe data files - `http://battlescribedata.appspot.com/#/repos`

```
catalogue
    ATTRIBUTES
    entries
    rules
    links
    sharedEntries
    sharedEntryGroups
    sharedRules
    sharedProfiles
```

Figure 3.3: Root level node

```
entry
    ATTRIBUTES
    entries
    entryGroups
    modifiers
    rules
    profiles
    links
```

Figure 3.4: Entry node

"categoryId", "type", "minSelections", "maxSelections", "minInForce", "maxInForce", "minInRoster", "maxInRoster", "minPoints", "maxPoints", "collective", "hidden" and "page". Additionally *entry* is the parent node to the children seen in Figure 3.4. Further details pertaining to the structure of the XML can be found in Appendix A.

The parser is divided into 24 different classes, where around half of these solely exist to deal with the plurality tags such as *entries*, *entryGroups* etc. The traversal is done by reading the tags, where each tag is considered an event, and everything happening between the start and end event of this tag defines what actions should be taken. For instance if the parser is currently on the *catalogue* level, and a "<entries>" tag emerges, a new *entries* object is created and instantiated with the current event and the XMLReader which accounts for how far in the file the parser has gotten. The parser is now on the *entries* level, where only "<entry>" tags are allowed to exist. Whenever this tag emerges, a new *entry* object is created and instantiated with the current event and the XMLReader, which then defines what tags can emerge and what sort of actions must be executed to deal with these in agreement with Figure 3.4. When an end event is triggered by a "</entry>" tag, the *entry* level loop is terminated and the finished *entry* object is added to the list of entries in the *entries* level it belongs to. Once all the *entry* tags of the current *entries* level have been accounted for, an end event for *entries* is triggered by a "</entries>" tag, and the *entries* level loop is terminated. This in turn allows the *catalogue* level to continue its loop after extracting the list of entries from the *entries* object.

3.4 Incorporating the data parser with the model

In order to take advantage of the model, it has to be populated with data. The parser already has all the data from the data files stored in the parser object,

yet this data is now stored as several objects of a tree. Thus some tree traversal code was required to traverse this object and use the relevant data. As the model currently lacks some features in regards to weaponry and utility units, the most interesting parts are contained in the *entries*. Each single entry specify a seperate unit, yet an entry may contain one or several entries to account for utility units, extra units that are part of the larger unit and so on. Thus it is necessary to distinguishing the correct entry in relation to the parent in order to find the correct *profile* for that unit. The profile contains all the information regarding the statline of the unit, and also what sort of unit it is, such as Infantry, Cavalry etc. Finding this profile triggers the creation of a Profile Instance, which is then populated with the correct data and then stored in the case base *Profiles*. After this Profile Instance is created the Unit Instance is able to set its UnitProfile attribute to this Profile Instance. There may also be additional profiles for the weapons this unit may hold, these profiles are reviewed to examine whether the unit has ranged capabilities or not. The Unit Instance is populated with the important fields from the entry, such as armyType, minSelections, costPerModel and maxInRoster, Point (cost), ModelCount and Name.

The purpose of this is to populate the different faction case bases with the available units for this faction. The Dwarfs catalogue is traversed to populate the DwarfUnits case base for example.

Furthermore the case base must contain some actual army cases. These can be added through a seperate class which reads all files in a folder and adds the specified units from these cases into the Units case base. This collection of units is then stored in the Units attribute of the Army Instance. This is done by specifying the faction, points spent, how many units there are and listing the units in the file. As long as the faction is a legal one, the units from the faction case base will be loaded, and new unit instances will be made based upon the image of the default units in the faction case base. These new units are then updated with the correct point cost and modelCount. As these new units are actively used in army cases they are stored in the Units case base. These units do however not trigger the creation of a new Profile Instance, these instances can be reused by Unit Instances for the same unit.

Chapter 4

Experiments and Results

In order to verify how the work here has answered the research question posed in Chapter 1.2. *How well can a CBR system function within the boundaries of WFB army creation.* A series of experiments have been devised to handle the different aspects of the CBR cycle in this system. Thus, both the Retrieval Phase and the Adaptation Phase should be able to function under various conditions.

4.1 Experiment plan

4.1.1 Retrieval

For this system to function within the parameters set, the system should be capable of retrieving cases based upon similarity between army lists. Where these cases consist of information about what faction this army list describes, how many points were spent to build this army, and most importantly, which units were included in this army. To test whether this functionality is present we must make sure the system is capable of using the retrival engine and retrieve the expected cases, this is done by following the steps of Figure 4.1.

1. Retrieve cases based upon faction and units
2. Retrieve cases based upon points spent
3. Retrieve cases based upon faction, points spent and units

Figure 4.1: Testing retrival engine

4.1.2 Adaptation

Next a series of tests were devised to determine the capabilities of the adaptation phase. It is important to test whether the adaptation phase detects that the best case may be sufficent enough already without alterations, meaning it already fullfils the query in regards to faction, points spent and a set of units that must be a part of this case. The steps in Figure 4.2 should ensure that this feature works as it should.

1. Adapt case with only faction and points specified
2. Adapt case with faction, points and a subset of units specified. (Multiple times for different subsets of units and factions)

Figure 4.2: Testing adaptation, sufficient case

The more interesting parts of the adaptation phase is concerned with actually adapting a case to fulfill requirements the original case does not. Thus the adaptation phase should be able to create a new army by altering or even removing units to meet point goals and unit inclussion goals. Yet it is important that the new case is not changing in such a way that the spirit of the old case is lost. Thus the tests must account for a varied setup of inputs. To fully account for these capabilites the test will be ran 1000 times each, where each unique result will be counted and if there are any invalid results these will also appear.

1. Adapt case with only faction and a new point limit specified
2. Adapt case with faction, points and a subset of units specified. (Multiple times for different subsets of units, factions and point limits)

Figure 4.3: Testing adaptation, demanding queries

4.2 Experiment setup

4.2.1 Retrieval Setup

Table 4.1 shows the selected inputs for the experiment defined in Table 4.1

Step Number	Parameter	Input
1	Faction	Dwarfs
	Point Limit	2400
1	Faction	High Elves
	Point Limit	2400
2	Point Limit	2400
2	Point Limit	2000
3	Faction	High Elves
	Point Limit	2400
	Units	Great Eagle
		Silver Helms
3	Faction	Dwarfs
	Point Limit	2400
	Units	Cannon
		Gyrocopter

Table 4.1: Retrieval Test Setup

Step Number	Parameter	Input
1	Faction	Dwarfs
	Point Limit	2400
1	Faction	High Elves
	Point Limit	2400
1	Faction	High Elves
	Point Limit	2397
2	Faction	Dwarfs
	Point Limit	2400
	Units	Cannon
		Gyrocopter
2	Faction	High Elves
	Point Limit	2400
	Units	Great Eagle
		Silver Helms
2	Faction	High Elves
	Point Limit	2400
	Units	Prince
		Silver Helms

Table 4.2: Adaptation test, sufficient case Setup

4.2.2 Adaptation Setup

Table 4.2 shows the selected inputs for the experiment defined in Table 4.2

Step Number	Parameter	Input
1	Faction Point Limit	Dwarfs 1500, 2000, 3000
1	Faction Point Limit	High Elves 1500, 2000, 3000
2	Faction Point Limit Units	Dwarfs 1000 Cannon Flame Cannon Lord
2	Faction Point Limit Units	Dwarfs 3000 Dwarf Warriors Slayers Lord
2	Faction Point Limit Units	High Elves 1900 Archmage Prince Loremaster of Hoeth
2	Faction Point Limit Units	High Elves 2600 Prince Sisters of Avelorn Teclis Korhil

Table 4.3: Adaptation test, demanding queries Setup

Table 4.3 shows the selected inputs for the experiment defined in Table 4.3 Keep in mind every single test here is run a 1000 times.

Step Number	Parameter	Input	Result
1	Faction Point Limit	Dwarfs 2400	Success
1	Faction Point Limit	High Elves 2400	Success
2	Point Limit	2400	Success
2	Point Limit	2000	Semi-Success
3	Faction Point Limit Units	High Elves 2400 Great Eagle Silver Helms	Success
3	Faction Point Limit Units	Dwarfs 2400 Cannon Gyrocopter	Success

Table 4.4: Retrieval Test Results

4.3 Experiment results

4.3.1 Retrieval Results

The result of the retrieval test appear in Table 4.4. The results of this test was positive for the most part, yet the second test of test 2 was only partially successful. Both High Elf cases were returned with almost full similarity, even though they were 400 points away from the target.

4.3.2 Adaptation Results

The result of the adaptation test where the case itself was sufficient can be seen in Table 4.5. No surprises here, this test builds directly on the retrieval test.

The result of the more demanding adaption test is show in Table 4.6. This test is what shows the system actually performing under varied and difficult demands.

Step Number	Parameter	Input	Result
1	Faction Point Limit	Dwarfs 2400	Success
1	Faction Point Limit	High Elves 2400	Success
1	Faction Point Limit	High Elves 2397	Success
2	Faction Point Limit Units	Dwarfs 2400 Cannon Gyrocopter	Success
2	Faction Point Limit Units	High Elves 2400 Great Eagle Silver Helms	Success
2	Faction Point Limit Units	High Elves 2400 Prince Silver Helms	Success

Table 4.5: Adaptation test, sufficient case Result

Step Number	Parameter	Input	Result
1	Faction	Dwarfs	-
	Point Limit	1500	(Success, 309 unique)
		2000	(Success, 8 unique)
		3000	(Success, 12 unique)
1	Faction	High Elves	-
	Point Limit	1500	(Success, 9 unique)
		2000	(Success, 8 unique)
		3000	(Success, 15 unique)
2	Faction	Dwarfs	(Success, 7 unique)
	Point Limit	1000	
	Units	Cannon	
		Flame Cannon	
		Lord	
2	Faction	Dwarfs	(Success, 122 unique)
	Point Limit	3000	
	Units	Dwarf Warriors	
		Slayers	
		Lord	
2	Faction	High Elves	(Success, 14 unique)
	Point Limit	1900	
	Units	Archmage	
		Prince	
		Loremaster of Hoeth	
2	Faction	High Elves	(Success, 6 unique)
	Point Limit	2600	
	Units	Prince	
		Sisters of Avelorn	
		Teclis	
		Korhil	

Table 4.6: Adaptation test, demanding queries Result

Chapter 5

Evaluation and Discussion

The goal of this project was defined in Chapter 1.2. *Use CBR-method to create complete Warhammer Fantasy Battle army list configurations (rosters).* This chapter will attempt to clarify whether this goal has been met, and assess whether the research question *How well can a CBR system function within the boundaries of WFB army creation?* has been answered.

5.1 Evaluation

The majority of the experiments conducted in Chapter 4 had a succesful result. The retrieval tests all gave the most expected result for each test, but a revision of the similarity function may be in order to better distinguish cases on total spent points. The current model has a strong bias for providing high similarity values as long as either faction or units are met. This may be somewhat obvious per the similarity function (3.1). Yet some of this bias also stems from the fact that myCBR have trouble distinguishing small differences when the range of possible values is so vast. A 400 point difference in a 5000 point range, a mere 8% difference should rightfully not trigger a major similarity difference in any normal scale. Yet it should be of some significance in this domain, a 400 point difference can drastically alter what army you may build.

The first adaptation test was a continuation from the retrieval test, and the expected cases were returned unaltered. The more interesting results can be found in the second adaptation test. In this test the system was thoroughly tested on multiple fronts. The first part of the test consisted of several runs of different setups as shown in Table 4.3. With a varied setup of points available, the adaptation either had to reuse the points spent in the army to meet the set limits, or wisely attempt to spend extra points. The Table 4.6 have the results listed.

Starting with the "Step 1" tests, the Dwarf test on a 1500 point limit produced a very vast set of different solutions which was slightly unexpected. This may be due to the case having a varied set of relatively low cost units, which then could be combined in various ways. Yet both the 2000 and 3000 point limit tests had a limited result set. The High Elves however had few differences in the result set accross the board. Overall "Step 1" performed within the bounds set by the program, and provided legal results.

The "Step 2 " tests were intended to test the adaptions ability to both exchange units based on similarity, and adjusting the armies based on a varied set of point limits. With the exception of the second "Step 2" test, the algorithm provided a small set of possible solutions. The second test however provided a very varied result set. All the results from the "step 2" tests were legal end results with some variation in either unit composition or points spent on gear. Thus performing as expected.

Some of the adaption tests intially failed due to some minor bugs in the system, these were corrected and the tests were conducted again, with succesful results and all providing legal results. These final results are what is shown in Table 4.6.

Some work also went into devising explanations for the system, these are justification explanations that are shown whenever a unit is altered or removed. These explanations are however not complete, or fully implemented. They do for the most part explain what was done and why, but they are currently not actively recorded for later use.

5.2 Discussion

To assess how the research question can be answered, the goals are assessed and then discussed in relation to the reasearch question.

5.2.1 Goal

The goal of this project has been to create complete Warhammer Fantasy Battle army lists. The current system is capable of doing this, but it does not fully account for gear or utility units. However the current system performs as it should, and is able to produce varied army lists that all meet the requirements set by a user in the query, whilst also abiding by the rules of the game. As the goal of the project is reached in part, it is possible to state that CBR at least functions as a way to create a decision support system for WFB. As the model is somewhat limited, and because of the difficulty in creating a good revision phase, the goodness of the system is up for debate. Currently the functionality is good enough to suggest armies, but they may not be useful for a player. Thus it is

not possible to claim that CBR functions well for WFB army creation with the current system.

5.2.2 Subgoals

The project also had a few sub goals specified in Chapter 1.2.

Sub Goal 1 Determine what kinds of explanations are useful in this setting

Sub Goal 2 Adapt the proposed design to specific implementation plans

Sub Goal 3 Create a system capable of the proposed functionality

A great deal of work went into determining what kinds of explanations could be useful, looking at some previous systems, getting feedback through the interviews and finally tinkering about it. For the most part justification explanations seem most useful, a user would want to know why a certain unit is in the army more so than the other options. It is unfortunate though that this knowledge was not brought more into the final system, yet it is somewhat understandable. To fully be able to justify a units presence in a army, the required domain knowledge is very high. Some units serve a very vast range of roles depending on certain equipment or other units. Other units may seem out of place in a army, yet the creators thought process in including the unit may be sound. There are many possibilites, thus the explanation may either be wrong in the current context and simply not match what the intended purpose was.

The system implementation plans have been revised a few times, ultimately focusing the task at hand to a implementation plan as seen in Chapter 3. There were plans beyond what the current system is capable of. The original design incorporated the possibility to specify both wanted faction, points, units, unwanted units, specific equipment to focus on adding, and equipment that should not be a part of the final army. This design proved to be too ambitious, which promptly resulted in some simplifications. Another slightly different design was also proposed, in this design the case did not store data beyond a few indexing fields, and retrieve the information corresponding to that index from the data files. Yet this was ultimately discared when chosing to base the system on myCBR, as it needs the actual data to perform as expected.

The current system is capable of the important parts of the proposed functionality, yet as mentioned above some features are not present in the current system. What this system strongly supports though, is the relative ease of adding data to the case base, both different factions and new cases have quite simple procedures. This is partly due to getting domain knowledge through parsing already existing data files. It makes the system as a whole far more general in use, but this feature also subtracts some of faction specific information that could be accounted for

if building the domain knowledge from scratch. That would however be a very tedious process so the decision to base the system on a general approach has benefited this system greatly.

It should be mentioned that the current case base is somewhat lacking in different compositions of armies, which may have a great impact on what sort of results this system provides. Thus the system may behave somewhat differently, and perhaps better with more varied data. Furthermore the current method of adaptation may be somewhat limiting. As it will attempt to alter the army without destroying the spirit of the original army it is confined into not adding any other units beyond those already in the army, with the exception of the specified units in the query of course. The intent of this functionality is that without extensive domain knowledge to say otherwise, the case should not be altered massively. The original case creator had a plan when composing the army, and the case is already proven to be a good army.

Chapter 6

Conclusion

This work establishes that it is possible to use a CBR system with partial queries in an effort to reuse and build new cases of WFB army lists, thus fulfilling the goal set for this thesis. However, due to the information requirement and the knowledge needed to build a successful army, not just a legal one, it is very difficult to claim that CBR is a particullary good approach to WFB army list building, thus providing an answer for the research question from Chapter 1.2. WFB inherently possesses too many properties that cannot be justified by numbers alone, which makes accounting for these properties difficult. At its best the case base of armies functions as a collection of multiple proven good configurations due to the success the original users had with this army. However synergies may be lost when creating a new army with substantially different units or point limits, which in turn may reduce a very decent army into something that any WFB player could instantly spot as mediocre at best. This can partially be blamed on the user for insisting on using units that themselves do not work well together. Yet a recommender system should be able to inform the user in such an event.

6.1 Future Work

As the current system has only incorporated the retrival and adaption phases, the revision and retain phases still need to be created to fully envision the complete CBR system. The issues with revision will have to be accounted for in some way, preferably by some form of automation through simulating the game. This may however prove to be too complex, based on the very nature of WFB and its many intricacies and rules, which was brought up by both interviewees in Chapter 2.8. So the suggested short therm revision phase would be to have the user make alterations where he would deem fit, use the army list in a couple of matches and

return to retain the case if it was a success. Finding another method for revision may be beneficial.

Furthermore the system should ideally be able to better account for utility units and equipment as currently these items are not considered properly. The model already has the posibility to deal with some of this information, but further work on extending this model is necessary to fully incorporate these details.

Enabling the user to refuse a result, make a few alterations and re-run the system as discussed in the interviews, Chapter 2.8.1, is another feature that should be included. This would empower the user to a larger degree, ensuring their input is taking into consideration and the final result will be a part of their creation as well.

Due to the many variables that impact the creation of good armies, CBR may not be the best approach for this problem in its entirety. It may prudent to take another look at how a recommender system could be used with WFB with regards to unit synergies rather than complete armies. The system could recommend a subset of units that previously have worked well together with the units the user specifies. This could be achieved through statistics in form of how often some units are used in the same army, and some AI approach to distinguish when these patterns are important enough to make them noteworthy. The data requirements for this would be quite massive though.

Bibliography

Aamodt, A. and Plaza, E. (1994). Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI communications*, 7(1):39–59.

Bach, K., Sauer, C., Althoff, K. D., and Roth-Berghofer, T. (2014). Knowledge modeling with the open source tool mycbr.

Cunningham, P. (2009). A taxonomy of similarity mechanisms for case-based reasoning. *Knowledge and Data Engineering, IEEE Transactions on*, 21(11):1532–1543.

Hinkle, D. and Toomey, C. (1994). Clavier: Applying case-based reasoning to composite part fabrication. In *IAAI*.

López, B. (2002). Combining cbr and csp: A case study on holiday scheduling. Report, Technical Report, University of Girona, Spain.

Marling, C., Sqalli, M., Rissland, E., Muñoz-Avila, H., and Aha, D. (2002). Case-based reasoning integrations. *AI magazine*, 23(1):69.

myCBR (2014). Tutorial. `http://mycbr-project.net/downloads/myCBR_3_tutorial_slides.pdf`.

Qin, Y.-h. and Wei, G.-x. (2009). Product configuration based on cbr and csp. In *Measuring Technology and Mechatronics Automation, 2009. ICMTMA '09. International Conference on*, volume 3, pages 681–684.

Roth-Berghofer, T. (2004). *Explanations and Case-Based Reasoning: Foundational Issues*, volume 3155 of *Lecture Notes in Computer Science*, book section 29, pages 389–403. Springer Berlin Heidelberg.

Russell, S. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, 3rd edition.

Sørmo, F., Cassens, J., and Aamodt, A. (2005). Explanation in case-based reasoning–perspectives and goals. *Artificial Intelligence Review*, 24(2):109–143.

Sqalli, M. H. and Freuder, E. C. (1998). Cbr support for csp modeling of interoperability testing. In *Workshop on CBR Integrations, AAAI, Madison, Wisconsin, USA*.

Strandbråten, G. R. and Kofod-Petersen, A. (2011). Myrmidia–case-based reasoning for warhammer fantasy battle army building.

Tintarev, N. and Masthoff, J. (2007). A survey of explanations in recommender systems. In *Data Engineering Workshop, 2007 IEEE 23rd International Conference on*, pages 801–810.

Warhammer (2010). *the game of Fantasy Battle rule book*. Games Workshop, 8th edition.

Weber, B., Wild, W., and Breu, R. (2004). *CBRFlow: Enabling Adaptive Workflow Management Through Conversational Case-Based Reasoning*, volume 3155 of *Lecture Notes in Computer Science*, book section 32, pages 434–448. Springer Berlin Heidelberg.

Appendices

A XML structure

```
catalogue
    ATTRIBUTES
    entries
    rules
    links
    sharedEntries
    sharedEntryGroups
    sharedRules
    sharedProfiles
```

Figure A.1: Catalogue

```
entry
    ATTRIBUTES
     entries
     entryGroups
     modifiers
     rules
     profiles
     links
```

Figure A.2: Entry

```
entryGroup
     ATTRIBUTES
     entries
     entryGroups
     modifiers
     links
```

Figure A.3: EntryGroup

```
modifier
     ATTRIBUTES
     conditions
     conditionGroups
```

Figure A.4: Modifier

In Figure A.1 *sharedEntries* has the same structure as *entries*, *sharedEntryGroups* the same as *entryGroups*, *sharedRules* the same as *rules* and *sharedProfiles* the same as *profiles*. The only difference is in the tag, leading to needing seperate code for catching this and forwarding to the same code as used with the standard cases.

```
rule
    ATTRIBUTES
    description
    modifiers
```

Figure A.5: Rule

```
profile
    ATTRIBUTES
    characteristics
    modifiers
```

Figure A.6: Profile

```
link
    ATTRIBUTES
    modifiers
```

Figure A.7: Link

```
characteristic
    ATTRIBUTES
```

Figure A.8: Characteristic

```
conditionGroup
    ATTRIBUTES
    conditions
```

Figure A.9: ConditionGroups

```
condition
    ATTRIBUTES
```

Figure A.10: Condition

The tree structure of these nodes are explained in detail in Chapter 3.3. The *description* in Figure A.5 is a simple field containing text.

B Cases

In this appendix all the cases are accounted for, with all features listed. Although the system currently does not incorporate utility and equipment properly, listing these with their cost is important for future use and to ensure that the reader understands where the surpluss points are going. In Table B.3 a mage is classified with Wizard level 4. This is not part of the BattleScribe Data files, thus it must be an amendment in special tournament rules.

Unit	Utility and equipment	Count	Points
RuneSmith	Rune of spellbreaking (1st) (25pts)	1	85pts
RuneSmith	Rune of spellbreaking (1st) (25pts) Rune of the Furnace (5pts)	1	90pts
Thane	Master Rune of Valaya (65pts) Rune of Slowness (1st) (35pts) shield (3pts)	1	193pts
Quarrellers		12	144pts
Quarrellers		10	120pts
Thunderers	Full Command (30pts) shields (24pts)	24	342pts
Cannon	Rune of Burning (5pts) Rune of Forging(25pts)	1	150pts
Cannon	Rune of Forging(25pts)	1	145pts
Grudge Thrower	Rune of Forging(25pts) Rune of Penetrating (1st) (40pts) Rune of Penetrating (2nd) (10pts)	1	155pts
Gyrocopter		1	80pts
Gyrocopter		1	80pts
Hammerers	Musician (10pts) Standard Bearer (10pts) shields (19pts)	19	305pts
Irondrakes	Musician (10pts) Standard Bearer (10pts) Rune of slowness (1st) (35pts) Rune of slowness (2nd) (15pts)	18	340pts
Organ Fun	Rune of Accuracy (25pts) Rune of Forging (25pts)	1	170pts

Table B.1: Case 1 - Dwarfs 2399[1]

[1]Listed at `http://s3.zetaboards.com/WarTrond/topic/7725176/1/`

[2]Listed at `http://s3.zetaboards.com/WarTrond/topic/7725176/1/`

[3]Listed at `http://www.ulthuan.net/forum/viewtopic.php?f=67&t=45081&p=808516#p808516`

Unit	Utility and equipment	Count	Points
Loremaster of Hoeth	Book of Hoeth (55pts) Shield of the Merwyrm (15pts) Sword of Anti-Heroes (30pts)	1	330pts
Lothern Sea Helm	Battle Standard Bearer (25pts) Standard of Discipline (15pts)	1	140pts
Mage	Golden Crown of Altrazar (10pts) Ironcurse Icon (5pts) Power Stron (20pts) Wizard Level 2 (35pts)	1	155pts
Mage	Dispel Scroll (25pts) Wizard Level 4 (135pts)	1	210pts
Archers	Full Command	27	300pts
Ellyrian Reavers		5	80pts
Ellyrian Reavers	Bow (5pts)	5	85pts
Silver Helms	Standard Bearer (10pts) champion (10pts) shields(10pts)	5	135pts
White Lions of Chrace	Full Command (30pts) Banner of the World Dragon (50pts)	28	444pts
Eagle Claw Bolt Thrower		1	70pts
Eagle Claw Bolt Thrower		1	70pts
Eagle Claw Bolt Thrower		1	70pts
Eagle Claw Bolt Thrower		1	70pts
Frostheart Phoenix		1	240pts

Table B.2: Case 2 - High Elves 2399[2]

Unit	Utility and equipment	Count	Points
Prince	Golden Crown of Altrazar (10pts) Star Lance (30pts) Dragonhelm (10pts) The Other Trickster's Shard (15pts) Star Dragon (390pts) shield	1	598pts
Mage	Power Scroll (35pts) Wizard Level 2 (35pts)	1	155pts
Mage	Ironcurse Icon (5pts) Wizard Level 2 (35pts)	1	135pts
Noble	Battle Standard Bearer (25pts) Banner of the World Dragon (50pts) Ithilmar Barded Elven Steed (15pts) heavy Armour (4pts), shield(2pts), lance (6pts)	1	172pts
Noble	Ithilmar Barded Elven Steed (15pts) Enchanted Shield (5pts) Ogre Blade (40pts) heavy Armour (4pts)	1	134pts
Ellyrian Reavers		5	80pts
Ellyrian Reavers	Bow (5pts)	5	85pts
Ellyrian Reavers	Champion (10pts)	5	90pts
Silver Helms	Full Command (30pts) shields(32pts)	16	398pts
Eagle Claw Bolt Thrower		1	70pts
Eagle Claw Bolt Thrower		1	70pts
Eagle Claw Bolt Thrower		1	70pts
Frostheart Phoenix		1	240pts
Great Eagle		1	50pts
Great Eagle		1	50pts

Table B.3: Case 3 - High Elves 2397[3]

C Extended Model

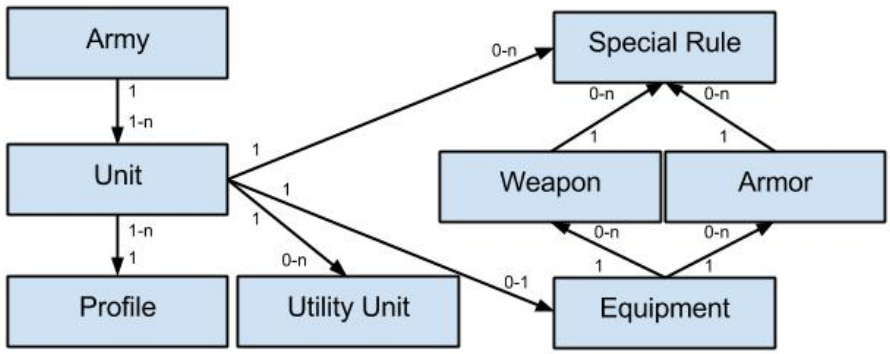

Figure C.11: Extended Model

Figure C.11 presents a slightly more detailed model, which accounts for special rules, equipment and utility units. It was ultimately not used due to some complications. It is also missing various magic items in equipment. The idea was to incorporate the cost from utility units into the overall cost in the unit, and similar for the equipment. The equipment may consist of several different weapons and armors. A unit may for example have both a gun and a sword.

Storing all the special rules in the model may not be viable due to how many there are, it might be easier to just get these from the parser data at runtime.