A Knowledge-based Configurator for Building MAGIC: THE GATHERING Card Decks

Robert Tieber[1] and **Alexander Felfernig**[2]

Abstract. The card game MAGIC: THE GATHERING is a trading card game (TCG) where players are required to build their own deck of cards to play a game. This is a time-consuming and skill-intensive task. In this paper, we demonstrate how a deck building task can be described as a configuration task. Furthermore, we show how this task can be supported in terms of a configurator application. We sketch related configuration knowledge representations, the user interface of our configurator, and point out issues for future work.

1 Introduction

MAGIC: THE GATHERING is one of the earliest *trading card games* (TCGs). In a TCG, players design a deck comprised of cards chosen before the game starts. For the entire game, the players are allowed to use only the chosen deck. Due to this process, game winning not only depends on the player's skill while playing the game, but also on their ability to construct a well-balanced and tuned deck of cards. The game can be played in a *competitive fashion* – in such situations, the number of decks is limited and related decks are considered as "best of breed". In MAGIC: THE GATHERING terms, this kind of setting is denoted as *Meta*. In such competitive environments, players can just choose predefined decks that have proven to work well.

MAGIC: THE GATHERING can also be played *casually*, i.e., not in a competitive fashion. In such settings, the overall goal is still to win the game but, however, there is also a strong interest in being creative, i.e., players want to design decks they would have the most fun playing with which are not necessarily the strongest or most effective ones. In this paper, we focus on the mentioned casual settings of MAGIC: THE GATHERING which allow a larger variety of decks to be used. Specifically, we introduce the so-called *Commander* format which includes two specific rules: (1) only a single copy of each card is allowed to be part of a deck, and (2) a *commander* has to be chosen which is responsible for setting a *theme* for a deck [6].

Playing MAGIC. Discussing the rules of MAGIC: THE GATHERING in detail is beyond the scope of this paper. In the following, we summarize major aspects of the game which are needed to understand our discussions of the *deck building problem*. A game can be played by *2..4 players*. Each player starts with a *life total* and a hand of cards. Once a player's life total reaches *zero*, he/she looses the game. The last player who remains in the game is the winner. MAGIC: THE GATHERING cards represent *resources* (=*Lands* generating *Mana*), *creature cards* which can be used to attack and defend, and *effects* that can be used to influence any part of the game (both called *Spells*). Creature cards (creatures) and effect cards (effects) are in the need of resources. Consequently, any reasonable (operable) deck should include both, cards in the need of resources and corresponding cards representing resources. Effects are either of type *one-time* or *permanent*, i.e., cards that remain on the playfield (i.e., battlefield). Cards with permanent effects can change the rules of the game (e.g., *creatures you control have a power of +2* or *each player may only play a single spell each turn*).

Building a Deck. Unlike traditional board games where the entire game is delivered as a single package, cards for *trading card games* (TCGs) such as MAGIC: THE GATHERING are purchased through card packs containing a random selection of cards. Interestingly, the prices of such cards on second-hand markets can range from 0.01€ to thousands of Euros making it important to also take into account available financial resources when building a *casual deck*. In order to design an effective deck, a player should identify a good mix of cards that fulfil different roles, for example, cards that allow a player to draw more cards (= card advantage), cards that interfere with the plan of opponents, and also cards that enable a player to prevent opponent interferences. There are over $25k$ unique cards available in the *Commander* format where many cards can fulfill more than one role. For this reason, card selection in the context of deck construction (100 cards are required) is a challenging and effortful process.

Related Work. Most frequently, decks are either designed in a complete manual fashion or with the support of a deck building software that usually allows a user to create categories and then search for individual cards that fit the selected categories. In existing solutions, additional information is displayed which summarizes major properties of the current card selection, for example, in terms of ratios between different card types and thus already helps to increase the efficiency of deck design processes. In contrast to existing deck building software solutions, some research prototypes support the automated generation of decks where generated decks can be evaluated in terms of the performance when playing against other automatically generated decks [1, 10]. In order to be more competitive in real-world scenarios, for example, as automated playing agents, these systems are in the need of further developments specifically related to game playing strategies.

The approach presented in this paper focuses on combining the advantages of the two mentioned approaches of deck construction. (1) decks are not configured in an automated fashion but rather interactively, (2) the decks built on the basis of the system presented in this paper can be used in regular MAGIC: THE GATHERING games with human players, (3) configuration technologies [2, 8] provide intelligent assistance for users in the deck design process.

The remainder of this paper is organized as follows. In Section 2, we discuss the basic knowledge representation used for representing a deck configuration problem and corresponding problem solving aspects. Thereafter, in Section 3, we provide an overview of the user

[1] TU Graz, Austria, Email: robert.tieber@student.tugraz.at
[2] TU Graz, Austria, Email: alexander.felfernig@tugraz.at

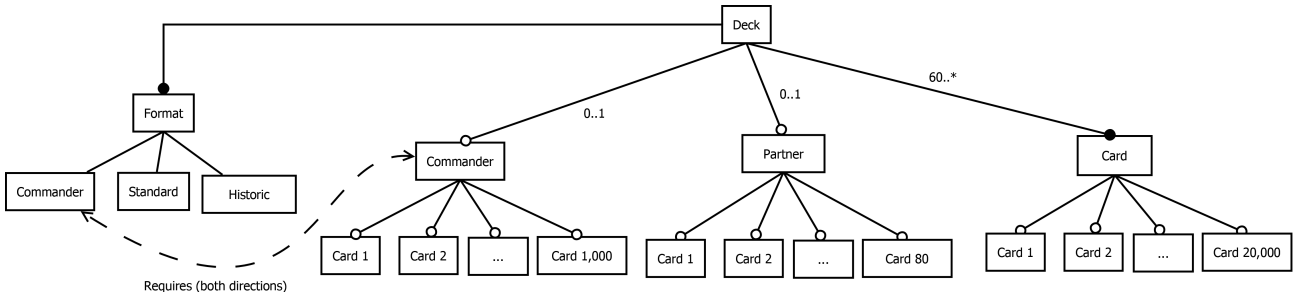

Figure 1. A feature model representing basic aspects of MAGIC: THE GATHERING.

interface of our configuration environment. The paper is concluded with an outlook of future work in Section 4.

2 Configuration Knowledge Representation

Deck Configuration Knowledge Base. A configuration knowledge base in our deck configuration environment is represented in the form of a Constraint Satisfaction Problem (CSP) [9]. Basic features as depicted in Figure 1 are represented as finite domain variables, where each feature is represented by the domain $\{true, false\}$ [3]. The solver used as a basis for our implementation is the Google SAT SOLVER.[3]

As shown in Figure 1, there are four major aspects a user has to take into account when defining a deck for MAGIC: THE GATHERING. First, a game format has to be selected. For simplicity, we will focus on the *Commander* format within the scope of this paper. In contrast to the other game formats, the *Commander* format needs a single card defined as *The Commander*. Furthermore, when selecting a certain card as *Commander*, a *Partner* (i.e., a second commander) can be included. For some cards defined as commander, only a specific second commander is allowed (this aspect is not represented in the feature model depicted in Figure 1). The total number of cards allowed to be part of a single deck is 100 – if commanders and partners are included, the number of included commanders and partners reduces the total number of allowed cards correspondingly. An aspect not taken into account in the feature model (Figure 1) is the following: cards of a deck have so-called color identities, where included cards are allowed to have only color identities specified as legal by the commander (often, 3-color commanders are used).

In addition to the constraints already introduced, the following constraints are part of our configuration knowledge base used to support the configuration of MAGIC: THE GATHERING decks. (1) *budget constraints* are used to define restrictions regarding the overall price of the cards part of a deck. In this context, each card has an associated *price* – a *resource constraint* then helps to limit the overall price (sum of the individual card prices) to the maximum price specified by the user – see, for example, Formula 1.

$$\Sigma_{c \in Cards}\ c.price \leq maxprice \tag{1}$$

If a card is already in a user's card collection, the price of deck inclusion is 0. In configuration terms, this is a typical example of a resource constraint. In this context, *Cards* represent the cards of a configuration. (2) in a similar fashion, *type constraints* help to specify the amount of cards of a specific type to be part of a deck configuration. An example of such a constraint is the following: *exactly one card in the final configuration should have the property* $isLand = true$. The corresponding logical representation is shown in Formula 2.

$$|\{c \in Cards : c.isLand = true\}| = 1 \tag{2}$$

Each such constraint is defined by a lower and an upper bound for card inclusion. Inclusion criteria can also be represented by more complex logical expressions – see Formula 3 (*a configuration must not include more than two creature cards costing* ≥ 5 *mana*).

$$0 \leq |\{c \in Cards : c.type = creature \wedge c.mana \geq 5\}| \leq 2 \tag{3}$$

There are some cards that are *always good to have*. Examples thereof are *draw cards*, *cards to provide resources*, and *cards supporting the interaction with opponents*. In future versions of our prototype implementation, we will include such constraints as recommendations to the user in order to avoid the re-invention of the wheel. Such constraints serve as a useful starting point of a deck configuration process and also help to reduce the overall effort of deck design.

Configuration Selection. Although the discussed constraints contribute to a significant reduction of the set of possible configurations, we are still confronted with a large amount of valid configurations out of which we should select a relevant one for the user. In our current implementation, configuration ranking is based on a quality score that is available for each card (derived from publicly available datasets[4]), where a maximum of $1 \leq n \leq 5$ candidate configurations (of equal quality) is generated by the solver on the basis of a predefined set of user requirements (constraints). In the current implementation, configuration selection is based on the idea of adding up the individual quality scores of the cards selected as a part of a configuration. The quality score per card is represented by the percentage of decks this card appears in as a *Commander*. In addition, users of our configurator application are allowed the adapt the quality score of individual cards manually.

Handling Unsolvable Requirements. Allowing users to specify constraints such as the discussed ones can lead to situations where these constraints become inconsistent. For example, if a user requires the inclusion of specific higher-priced cards but defines a strict overall price limit that does not allow the inclusion of these cards, no solution can be identified by the underlying constraint solver. In such a situation, we are in the need of identifying alternatives that help to support as many requirements as possible. For example, a card fulfilling two roles at a cost of 30€ could be replaced by two cards each at cost of 1€ supporting the same set of roles. On the solver level, we support this feature by encoding a corresponding relaxation mechanism a.o. for resource and type constraints (see Formula 4).

$$constraint_i \wedge costs_i = s_i \vee \neg constraint_i \wedge costs_i = s_i + k_i \tag{4}$$

[3] developers.google.com/optimization.

[4] edhrec.com.

In order to support such constraint relaxations on the solver level, we interpret the constraint relaxation task as a *minimization problem* with the overall goal of minimizing additional costs k_i (Formula 5).

$$Min \leftarrow \Sigma_{c_i \in Constraints}\ costs(c_i) \tag{5}$$

This is a specific approach for determining a so-called Maximum Satisfiable Subset (MSS) which can be regarded as the complement of a Minimal Correction Subset (MSS) which is often also denoted as diagnosis [4, 7]. We are also able to take into account the *quality of configurations* in terms of different optimization functions (an example thereof is depicted in Formula 6).

$$Max \leftarrow \frac{\Sigma_{ca_i \in Cards}\ quality(ca_i)}{\Sigma_{c_i \in Constraints}\ costs(c_i)} \tag{6}$$

3 Prototype Implementation

The deck configurator presented in this paper supports the creation of decks in two modes. First, the *easy deck creator* supports the creation of decks based on existing themes (from web sources) where the major task of a user is to find a relevant theme and define the preferred adaptations. Second, the *complex deck editor* supports the creation of new decks from scratch, i.e., users can create completely new decks based on their own ideas.

When using the *easy deck creator* mode, users select one theme from a set of predefined themes. Thereafter, the user receives a recommendation of preconfigured card categories with corresponding set of suggested cardinality limits. These limits can be adapted by the user. Finally, the user is asked the define the upper limit regarding the overall budget. On the basis of this information, a corresponding deck can be configured. A configured deck can thereafter be viewed and adapted with the *complex deck editor*. With this editor, users are allowed to specify more complex constraints regarding the inclusion and exclusion of specific card types (see, e.g., Formulae 2 and 3). An example of the graphical representation of a user-defined constraint of Formula 3 is depicted in Figure 2.

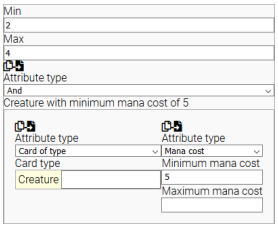

Figure 2. Definition of the card criteria *between 2 and 4 creatures with mana cost* $>= 5$ (see Formula 3).

Generated Deck. As soon as the system can generate a deck that satisfies the defined constraints, the configurator determines a complete deck which is shown via graphical user interface – a corresponding example is depicted in Figure 3. In some cases, there are multiple configurations of equivalent quality (score). In this case, the result presentation shows the differences between the configurations and a user can select one out of the presented alternatives.

4 Conclusions and Future Work

In this paper, we have presented a constraint-based configurator implemented for the MAGIC: THE GATHERING game. The approach

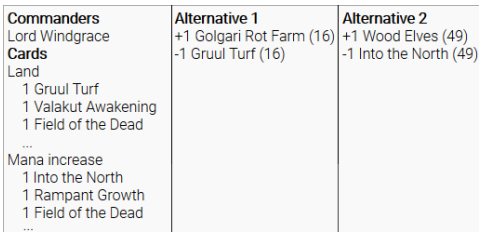

Figure 3. For the deck configuration in the left column, two alternatives are proposed: *Alternative 1* proposes the additional inclusion of *Golgari Rot Farm* and the exclusion of *Gruul Turf*. Furthermore, *Alternative 2* proposes the inclusion of *Wood Elves* and the exclusion of *Into the North*.

has already been empirically tested by experienced users and shows potential to support a more efficient deck design process. Our prototype implementation can help users to decrease the time needed to complete deck design processes. This can be achieved especially in the context of using the *easy deck creator* mode with the downside of low design flexibility and a resulting potential low deck quality. In contrast, the *complex deck creator* allows for higher flexibility with the downside of higher design efforts.

Major issues for future work are the following. First, we will develop and compare different alternative optimization functions with regard to their ability to generate user-relevant configurations. Second, for the purpose of predicting the preferences of users with regard to individual cards, we will analyze to which extent the concepts of model-based collaborative filtering (specifically matrix factorization [5]) can be applied for the prediction of user preferences. Third, the current version of our configurator prototype focuses on a specific game format. For future versions, we plan an extension to further game formats. Finally, we expect significant improvements in the user interface (UI) since the current version is not designed to be used by unexperienced users.

REFERENCES

[1] S. Bjørke and K. Fludal, *Deckbuilding in Magic: The Gathering Using a Genetic Algorithm*, Master's thesis, NTNU, 2017.

[2] A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, *Knowledge-based Configuration - From Research to Business Cases*, Elsevier, 2014.

[3] A. Felfernig, V.M. Le, A. Popescu, M. Uta, T.N.T. Tran, and M. Atas, 'An Overview of Recommender Systems and Machine Learning in Feature Modeling and Configuration', in *15th Intl. Working Conference on Variability Modelling of Software-Intensive Systems (VaMoS'21)*, pp. 1–8, Virtual, Krems, Austria, (2021). ACM.

[4] A. Felfernig, M. Schubert, and C. Zehentner, 'An Efficient Diagnosis Algorithm for Inconsistent Constraint Sets', *AIEDAM*, **26**(1), 53–62, (2012).

[5] Y. Koren, R. Bell, and C. Volinsky, 'Matrix Factorization Techniques for Recommender Systems', *IEEE Computer*, **42**, 30–37, (2009).

[6] S. Larabee, T. Elliott, G. Duggan, and S. Menery. Commander Rules – Official Commander Website, 2021.

[7] R. Reiter, 'A Theory of Diagnosis from First Principles', *Artif. Intell.*, **32**(1), 57–95, (1987).

[8] M. Stumptner, 'An overview of knowledge-based configuration', *AI Communications*, **10**(2), 111–125, (1997).

[9] E. Tsang, *Foundations of Constraint Satisfaction*, Academic Press, London, 1993.

[10] C. Ward and P. Cowling, 'Monte Carlo Search Applied to Card Selection in Magic: The Gathering', in *2009 IEEE Symposium on Computational Intelligence and Games*, pp. 9–16. IEEE, (2009).