Symbolic Reasoning for Hearthstone

Andreas Stiegler, *Member, IEEE*, Keshav Dahal, *Senior Member, IEEE*,
Johannes Maucher, *Member, IEEE*, and Daniel Livingstone, *Member, IEEE*

***Abstract*—Trading-Card-Games are an interesting problem domain for Game AI, as they feature some challenges, such as highly variable game mechanics, that are not encountered in this intensity in many other genres. We present an expert system forming a player-level AI for the digital Trading-Card-Game Hearthstone. The bot uses a symbolic approach with a semantic structure, acting as an ontology, to represent both static descriptions of the game mechanics and dynamic game-state memories. Methods are introduced to reduce the amount of expert knowledge, such as popular moves or strategies, represented in the ontology, as the bot should derive such decisions in a symbolic way from its knowledge base. We narrow down the problem domain, selecting the relevant aspects for a play-to-win bot approach and comparing an ontology-driven approach to other approaches such as machine learning and case-based reasoning. Upon this basis, we describe how the semantic structure is linked with the game-state and how different aspects, such as memories, are encoded. An example will illustrate how the bot, at runtime, uses rules and queries on the semantic structure combined with a simple utility system to do reasoning and strategic planning. Finally, an evaluation is presented that was conducted by fielding the bot against the stock "Expert" AI that Hearthstone is shipped with, as well as Human opponents of various skill levels in order to assess how well the bot plays. Evaluating how believable the bot reasons is assessed through a Pseudo-Turing test.**

I. Introduction

"Hearthstone: Heroes of Warcraft"[1] (or "Hearthstone" for short) is a digital Trading-Card-Game (TCG), comparable to other examples of its genre, such as "Magic: The Gathering"[2] or "Android: Netrunner"[3]. We propose a concept for a player-level AI playing the full game of Hearthstone. TCGs typically offer various types of cards, such as minions and spells, which cost resources to play. Minions battle against the minions of enemy players to achieve victory. An important aspect of TCGs – and Hearthstone in particular – is planning on how to use the available cards and which actions to perform. These planning aspects and the reasoning systems required for it are focus of our research.

A lot of reasoning and planning research is currently conducted for Real-Time-Strategy (RTS) games, such as "StarCraft"[4], as analyzed by Ontanón et al [1]. Reasoning and planning in games, RTS games in particular, was promoted as an interesting research problem by Buro [2, 3] and by many researchers since then [4]. In Hearthstone, as in RTS games, a lot of short and long term planning has to be done. RTS bots are typically split into micro- and macro-management, dealing with individual units or the large scale battle plan respectively. The move selection in Hearthstone is arguably of similar complexity to typical planning tasks in RTS. Selecting which card to play or selecting which unit to produce might be a similar problem. Yet, Hearthstone has some preferable properties for developing game AI, in contrast to an RTS. Hearthstone is non-spatial, as the position of cards are only discrete slots, not a continuous map. This allows the AI to skip micro-management altogether, focusing on the strategic and tactical reasoning of playing cards. Further, Hearthstone is static, so the game-state is not changing while the reasoning process runs, while still offering typical game AI planning challenges, such as being adversarial, partially observable and having temporal constraints on actions. The TCG game domain, however, comes with some interesting challenges. Uncertainty, for example, is of a different flavor in TCGs than in RTS games. In RTS games, partial information usually originates from not being able to perfectly observe the enemy. However, each game session consists of the same game entities, making guessing a good option. There are only so many openings a player can perform in an RTS and once a certain building was scouted, it is reasonable to infer the strategy behind the current enemy build. In TCGs, however, it is even uncertain which game entities – which cards – the enemy player might be using. There are certain baseline cards that can be expected, but in general guessing is harder in TCGs. In Hearthstone, a player's deck consists of 30 cards of a card pool of over a thousand and of the 30 cards, many will be duplicates. A player in a TCG game session only uses a tiny bit of the game's mechanics to deploy their strategy, whereas other domains, like RTS, utilize a large portion of their game mechanics in every single game.

Further, the impact of synergies and interactions between game mechanics has more impact on the flow of the game. In TCGs, it is not uncommon for a single card to gain a greatly increased amount of value out of it when synergizing it with

Andreas Stiegler is with the Stuttgart Media University, Nobelstraße 10, Stuttgart, Germany (e-mail: mail@andreasstiegler.com).

Keshav Dahal is with the University of the West of Scotland, PA1 2BE, Paisley, UK. He is also affiliated with Nanjing University of Information Science and Technology (NUIST), China (e-mail: keshav.dahal@uws.ac.uk).

Daniel Livingstone is with the Glasgow School of Arts, G3 6RQ, Glasgow, UK (e-mail: d.livingstone@gsa.ac.uk).

Johannes Maucher is with the Stuttgart Media University, Nobelstraße 10, Stuttgart, Germany (e-mail: maucher@hdm-stuttgart.de).

[1] Blizzard Entertainment, 2014 (http://us.battle.net/hearthstone/en/)
[2] Wizards of the Coast, 1993 (http://magic.wizards.com/en)
[3] Fantasy Flight Games, 2012 (http://www.fantasyflightgames.com/edge_minisite.asp?eidm=207)
[4] Blizzard Entertainment, 1998 (http://us.blizzard.com/en-us/games/sc/)

other cards. A minion could get 10 times the attack and health values if used correctly. Synergies also appear in other game domains, but in TCGs they are particularly escalating. This often goes hand in hand with the tendency of TCGs to alter their game mechanics at runtime. The rules by which attacks or spells work may be altered by other cards, allowing creative synergies and combos.

Due to these properties, TCGs are already used in academic work, such as using Monte Carlo Tree Search to cover uncertainty [5] or card selection [6]. Other work, such as HoningStone [7], focuses on the creative aspects of selecting card combos. Mahlmann et al [19] introduce the deck-building card game "Dominion"[5] as a testbed to construct interesting decks. Although RTS research saw a far greater variety of approaches deployed, we want to propose TCGs as a testbed for the higher-level planning problems, which are often hard to work on in RTS games, as they typically tie towards micro-management, thus requiring a multi-agent approach to cover all the aspects of a full RTS: If the underlying micro-management is not solved sufficiently well, higher-level planning will be limited, too. As described in [1], all academic StarCraft bots in the respective challenges used a multi-agent structure of some kind, typically splitting the reasoning task into higher-level planning and lower-level micro-management problems. Churchill, one of the developers of the TCG and RTS hybrid "Prismata"[6], dissects the architecture of their TCG-AI in [8] and also illustrates the challenges.

TCGs also share some similarities with other card games, such as Contract Bridge. Similar to TCGs, a player's available game entities, their hand, are note observable by others. Buro, Long and Furtak [24, 25] demonstrated a Monte Carlo approach for Skat and, along with the Perfect Information Monte Carlo approach of Ginsberg and Long et al. [23], both solutions are now considered expert-caliber players, as Long [26] reports. Thus, Monte Carlo Tree Search seems like a good candidate to be applied to TCGs, but wasn't yet able to replicate the success.

Reasoning problems similar to those found in Hearthstone are already solved via other approaches, most dominantly Machine Learning (ML), Case-Based-Reasoning (CBR), and Monte Carlo Tree Search (MCTS). The TCG game domain, however, is an example of game domains with some interesting properties: their tendency to alter game mechanics and even basic rules of the game at runtime, a complex action space including many seemingly suicidal actions and a high degree of synergies between individual game elements. Our hypothesis is that for this game domain – and perhaps similar ones – a symbolic AI can produce a good player-level AI playing to win the game. Here, we define a player-level AI as "good" if it is capable of playing effective – thus winning games against various opponents – and believable – producing strategies that are perceived as natural by human players. Other common metrics for player-level AI evaluation, such as runtime performance and controllability, are secondary. This chapter will briefly highlight ML, CBR and MCTS and gives some rational why we think a symbolic approach might perform on a competitive level in this particular game domain. Chapter II will introduce the knowledge representation for the proposed symbolic system. Chapter III then explains the actual reasoning process and Chapter IV gives an example of the approach in action. The evaluation chapter will compare the performance of our approach against an MCTS agent, as MCTS can be considered the good standard in similar game domains, producing effective reasoning results. The evaluation will also cover how well the AI plays different classes, plays against human players of different skill-levels and a pseudo-Turing test to assess how believable it performs.

A. Machine Learning

A ML approach to Hearthstone could analyze recorded human games to identify how cost effective individual cards are. Such a metric could then act as a utility function. A similar approach to value Hearthstone cards through ML was presented on the Defcon by Elie Bursztein [9] and is available on his website[7]. One could further search for good move sequences or even identify predominant strategies, such as classifying the opponent deck early through cards played. For similar problems in RTS games, such as strategy prediction where a strategy is a sequence of moves – very similar to a TCG – there are already some applications with promising results, such as the work done by Weber [10] or Wender [11]. Such ML approaches are effective solutions that often do most of their processing at development time and thus have very good runtime complexity. ML approaches might, however, encounter some challenges in game domains where the underlying mechanics and entities change frequently – such as in Hearthstone: A single card can turn all healing into damage or swap the attack and defense of all minions. If such fundamental rules of the game can change frequently, that makes modelling an actual game-state or strategic decision for a ML approach difficult.

B. Case-Based Reasoning

CBR approaches, on the other hand, put a database of recorded situations at their heart. These situations are often in a more abstract format and offer abstraction and concretization methods to move from a current game-state into the abstract domain of the database, perform a search for the most similar cases, and then concretize the actions performed in the recorded cases back to the current game-state. Looking again at similar planning problems in the RTS domain, we see solutions for selecting proper action sequences in tactical reasoning, such as the work done by Cadena [12] or even whole build orders such as the approach by Weber [13]. Particularly the concretization step in CBR, moving from an abstract action performed in a case to a concrete action available in the current action space is a vulnerable point of CBR approaches, in particular when the action space of a game is highly variable. This will lead to situations where cases very close to the current game-state were found, but the

[5] Rio Grande Games, 2008 (http://riograndegames.com/Game/278-Dominion)

[6] Lunarch Studios (http://blog.prismata.net/2014/12/17/the-prismata-ai/)

[7] https://www.elie.net/hearthstone/card_analysis

actions that lead to victory in those cases are not available in the current action space. This is particularly probable for TCGs, as the majority of their action space – the cards in a player's hand – is randomly selected.

C. Monte Carlo Tree Search

A particular well-researched category of reasoning and planning algorithms in games are tree-based approaches. They build on forward-modelling future game states and action spaces to arrive at a tree describing how the future of the game may evolve. On this tree, a search algorithm can now operate to find a move that pushes the overall expected future of the game into a favorable direction. The original version of these approaches became popular as soon as the 90s, such as the book by Allis [20] with many examples from physical turn-based games such as Chess or Go. In terms of digital games, tree-search saw many applications, but wasn't yet able to solve the high-level planning problems of a player-level AI in complex game genres, as Robertson and Watson [21] summarize for the RTS game domain. Recent research focuses more on Monte Carlo versions of game tree search, which solve constructing and expanding the tree in a stochastically sampled manner. Browne et al [22] give a good overview on the applications and current state of Monte Carlo Tree Search as of the year 2012. In general, the TCG game domain is suited for the application of tree based reasoning approaches: they are turn-based and thus temporally discrete, offering a clear way on how to map the causality of a game onto a tree. Further, they are often spatially discrete and have less complex overall game states compared to other tactical games such as RTS. In the Hearthstone reddit[8], there is a report of a MCTS implementation for Hearthstone, though with some limitations such as having perfect knowledge. In the broader scope of TCGs, there is also work on using MCTS by Ward et al [6] for a simplified version of "Magic: The Gathering" only allowing minion cards and a greatly reduced card pool. In their work, they illustrate a common approach, the bandit-based MCTS implementing an Upper Confidence Bound (UCB or UCT if applied to trees). They evaluate several combinations of using rule-based, random and Monte Carlo approaches for the different tasks in their minimalized TCG version (Attack, Blocking, Playing Cards) where the best result was produced by a rule-based attacker and blocker using MCTS for playing cards.

However, there are also some challenges in the TCG domain. In general, TCGs have a relatively large branching factor and a huge variety in actions. Knowledge about the hostile hand is usually inaccessible and thus there is a strong factor of uncertainty. With card pools of several hundreds or even thousands, this leads to a huge variety in the potential action space for the same game state, leading to an explosion in branching factors for TCGs, as Cowling et al [5] found. A common technique for MCTS approaches in situations with uncertainty is reducing the problem to a perfect information game through determinization. Cowling et al [5] detail such an approach for "Magic: The Gathering". In TCGs, the key stochastic element is usually drawing a card from the player's decks. A determinization strategy could create fixed deck lists and then span of a MCTS tree using a deck list to determinize all otherwise stochastic card draws. In their comparison of several different MCTS players, the naive UCT implementation scored worst with a win rate of 26% at most, whereas determinization-bots scored far better, in particular when using a binary MCTS tree (in which each node represents the decision whether to execute a specific action or not) compared to the otherwise common n-degree trees (in which each node represents the decision which of the available actions to execute). Besides the huge branching factor, another challenge of the TCG domain is more subtile. The action spaces of TCGs are usually "polluted" with actions that are suicidal in many situations: it is allowed by the rules of these games to use cards in seemingly awkward ways, such as killing one's own minions with a fireball or even casting the fireball at a player's own hero. This would be similar to an FPS allowing a player to shoot themselves with their guns. However, these seemingly suicidal actions exist for a reason. There are often opportunities to combine cards and effects in creative ways exploiting such actions. A popular example from Hearthstone is playing the minion "Sylvanas Windrunner" and then immediately destroying her with a spell. When "Sylvanas Windrunner" dies, she takes over a random enemy minion. Thus, a combination evolving around the seemingly suicidal action of killing one's own minion can be a powerful way to steal a particular threat the enemy deployed. In a MCTS simulation with a random simulation policy, however, many of these actions will be played in situations where they are truly suicidal. One could filter them, but that could render a bot less able to use many of the powerful combinations and moves they offer. The tendency of the TCG game domain to alter the rules and mechanics of the game during a game session renders pruning actions harder, as it is more difficult to assess at development time whether actions should be pruned or not. Taking a closer look at Hearthstone, its game domain offers another interesting property: a given situation might not have a significant impact on the end of the game. Hearthstone contains many cards that allow to completely change the game state with just a single move, such as cards like "Twisting Nether" that kills all minions on the board, or "Reno Jackson" that heals all damage inflicted to a hero, or even "Renounce Darkness" that allows a Warlock to omit all their Warlock-specific cards and replace them with some from another class, effectively switching a player's hero class during the course of a game session. Thus, the MCTS simulation, effectively searching for a path from the current state to a victory state, might not take these game-changer cards into account or a random-player simulation might imagine them too often. Due to its good results solving many planning problems in TCGs and related game domains, we chose an MCTS implementation as a measurement to ground the performance of a symbolic system. Details can be found in the evaluation chapter below.

[8] https://www.reddit.com/r/hearthstone/comments/3zdibn/intelligent_agents_for_hearthstone/

D. Symbolic Expert Systems

As Hearthstone is an example of a game domain with both highly variable game mechanics and an unforeseeable action space, we were interested whether other approaches can perform well. We opted to put an expert system using an ontology in the form of a semantic structure at the heart of our bot. During development, a key element in improving the performance of the bot was to tailor and design what kind of information is stored in which part of the semantic structure. Hereby we differentiate between several categories of knowledge: memories, domain knowledge and expert knowledge. Memories are the representation of the current state in which the game is and how we got there. Domain Knowledge is any objective information that describes the problem domain, such as the rules and mechanics of Hearthstone, whereas expert knowledge is information about the meta-game, such as how valuable certain cards are or which combos are dominant in strategies, which is often derived from long-term experience. However, we try to minimize the usage of expert knowledge, such as storing which cards offer synergies or which move sequences may form popular strategies. This decision allowed us a relatively cheap expansion of the bot to include new cards and features, as most of the reasoning is built upon Domain Knowledge and Memories which both don't require tremendous amounts of data mining or test games to get. Only the Memories and Domain Knowledge are stored directly in the Knowledge Base, whereas expert knowledge plays a role in the production rules as described below.

II. Dynamic and Static Knowledge

The heart of our bot is the ontology, the knowledge base which represents the state of the world and any further knowledge for the reasoning process. The knowledge base of the bot is organized in two primary segments: Static and Dynamic Knowledge. Static Knowledge is knowledge with which the bot is initialized, for example containing knowledge on which cards there are, how their effects work, what the rules of the game are etc. Static Knowledge represents generic information on the game that's not specific to a game session. The Static Knowledge merely describes the objective domain knowledge – the rules and mechanics of the game – just as a human could find them in a manual or Wikipedia. The Static Knowledge is authored manually and covers all recent cards and game mechanics up to the "Whispers of the Old Gods" expansion pack ("One Night in Karazhan" was not yet released when this paper was written).

Dynamic knowledge, in contrast, describes the entities which are currently active in a game session and thus the memory of the bot. While the Static Knowledge, for example, would encode that there is an "Auchenai Soulpriest" card, which is a minion with 3 attack and 5 health that turns healing abilities into damage, the Dynamic Knowledge would encode that there is one "Minion #17" in the current game session, that's an instance of "Auchenai Soulpriest" and has 3 health remaining. Both knowledge bases are represented in a single semantic net using a simplified format similar as described in [14]. Such a net consist of a list of nodes, a list of relations, and a list of attributes.

Nodes typically represent entities or concepts of the game world, such as "Card" or "Health". In some ontology formats, nodes can carry attributes to define further aspects of the represented concept, for example having a "cost" attribute to a node that represents a building. After some experimentation, we omitted node-attributes and instead stored such information entirely through relations, which still can have attributes. This leaves nodes to only having checks for identity, whereas most semantics are represented through relations and their attributes. Omitting node attributes made the reasoning rules much more simpler and the resulting ontology was still sufficient to represent the mechanics of Hearthstone.

Relations form a directed graph spanning over the nodes. Relations encode semantics between nodes, such as "Auchenai Soulpriest"-"is_a"-"Minion", and are the key data structure in our approach. They can be further decorated with attributes to add additional information, such as describing that "Auchenai Soulpriest"-"has:4"-"Cost".

Figure 1 illustrates how the three basic data types relate to each other and how they are implemented in the current version of the bot. The implementation contains three further tables for human readability, mapping node::ID, Relation::type and Attribute::type to strings.

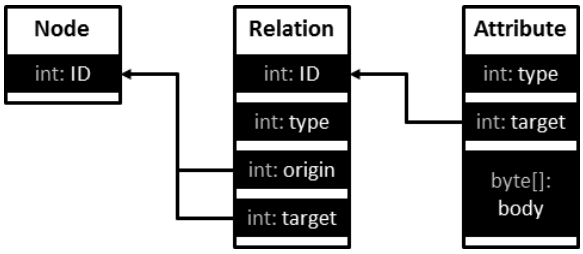

Figure 1: Semantic structure data specification, showing all required data fields.

Queries on the semantic structure now become search operations in the three datasets, such as answering the question if "Auchenai Soulpriest" "is_a" "Card" becomes a search query in the relations list, looking for a path that starts at the node of "Auchenai Soulpriest", ends at the node of "Card" and only uses relations of the type "is_a". The approach of putting a large data structure at the heart of the reasoning process is greatly influenced by the keynote of Jeff Orkin at the CIG2012 [15], proposing that reasoning systems for games can become closer to search engines performing lookups in a vast pool of memories. Our approach represents domain knowledge and memories in the same data structure, closely linking them to allow search operations that span over both, for example to answer the question if "Minion #02" "has" "Taunt". "Taunt" is an attribute that certain minions can have, forcing an opponent to attack them first before they can inflict damage to the enemy player. As "Taunt" is an attribute of a specific card, such as "Voidwalker", the "has" "Taunt" relation would be part of the Static Knowledge. The node

"Minion #02", however, is a specific instance of a minion in this very game session and is therefore part of the current set of Dynamic Knowledge, but "Minion #02" would have a "is_a" relation to the Static Knowledge "Voidwalker" node. Answering the above question now becomes a pathfinding query, looking for a path between "Minion #02" and "Taunt", only using "is_a" and "has" relations. This also covers cases where an attribute such as "Taunt" can either be a static property of a card, as in the case detailed above, or where "Taunt" can be added to a minion through special effects, such as the Spell "Mark of the Wild", granting a minion an attack and health bonus, as well as "Taunt". In this case, the minion node would have a dynamic "has" relation to "Taunt", without the need to climb up several "is_a" relations to ascend into Static Knowledge. Figure 2 shows an example of how static and Dynamic Knowledge are represented in a semantic structure.

The two fundamental relations in a semantic structure are abstraction ("is_a") and aggregation ("has"), with all other relations (such as "alters") being shortcuts with hardcoded semantics. When querying the semantic structure to return an attribute value there might be multiple, conceptually correct answers. Figure 2, for example, contains multiple "has"-"Health" relations, one with an "amount" attribute of 1, one with 5 and one without any "amount" attribute at all. The attribute retrieval algorithm, however, always returns the value of the attribute that is least abstract, thus that required to pass by as few "is_a" relations as possible. In the example in figure 2, the "Minion"-"has"-"Health" relation requires following two "is_a" relations from "Minion_013" to be reached, while the "Minion_013"-"has"-"Health" relation requires none. Thus, the latter would be the relation from which the attribute is extracted. The same process is used for all attribute retrieval.

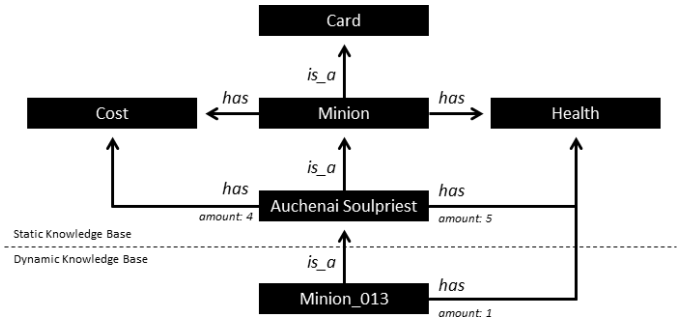

Figure 2: Excerpt of the semantic net, illustrating the relations between a Dynamic Knowledge node representing a concrete instance of a minion and the Static Knowledge related to this minion: A "Minion" just always has "Health", an "Auchenai Soulpriest" has 5 base "Health" and the instance "Minion_013" of the current game session has only 1 "Health" remaining.

The semantic net for Static Knowledge used in the current version of the bot covers all collectible cards of Hearthstone, including the expansions that were released after launch up until "Whispers of the Old Gods". It contains 2156 nodes, 10954 relations of 19 distinct types and 6384 attributes of 8 distinct types. During a typical game session, the Dynamic Knowledge allocates ~1200 additional nodes, representing active minions, the player's hand and knowledge of the enemy, for example through cards that were returned to an enemy's hand. These nodes span ~2500 additional dynamic relations to other dynamic or static nodes and introduce ~900 attributes. Both static and Dynamic Knowledge can be serialized, of which a sample is shown in Figure 3.

```
"Earth Elemental": {
 "relations": [
   {
     "type": "is_a",
     "target": "Minion"
   },
   {
     "type": "has",
     "target": "Cost",
     "attributes": [
       {"type": "amount", "value": 5}
     ]
   },
   {
     "type": "has",
     "target": "Overload",
     "attributes": [
       {"type": "amount", "value": 3}
     ]
   },
   {
     "type": "has",
     "target": "Attack",
     "attributes": [
       {"type": "amount", "value": 7}
     ]
   },
   {
     "type": "has",
     "target": "Health",
     "attributes": [
       {"type": "amount", "value": 8}
     ]
   },
   {
     "type": "has",
     "target": "Taunt"
   }
 ]
}
```

Figure 3: Excerpt of a serialized minion node "Earth Elemental" with its relations towards more abstract nodes such as "Minion" (superclass) or "Cost" (component). The strings used as labels for types and targets are translated into unique IDs when parsed.

At first glance, using a semantic net as a knowledge base might just look like a glorified look-up where a simple table would have been sufficient. However, the big benefit of semantic structures are that it's possible to describe the semantics of game mechanics instead of hard-coding them into the reasoning system of the bot. The "Earth Elemental" example shown in Figure 3, for example, could also have been described using fields with fixed semantics, such as a field "Cost" or "Overload". In such an AI system, the semantics of these fields are predetermined through the reasoning algorithm. One could see that as hard-coding the domain knowledge about the "Overload" game mechanic into the reasoning process. An AI system which later-on uses this field, for example in a cost calculation, would then rely on that the hard-coded semantics are still the same. While this is a well-established approach and works well for many game-AI problems, we argue it is not optimal for situations where the underlying game mechanics can change. In the case of Hearthstone, there are many cards which alter the fundamental

rules of the game, such as “Auchenai Soulpriest”, a minion transforming any source of healing into damage or “Feign Death” a spell which allows to trigger any “Deathrattle” effects on minions without having them to die (“Deathrattle” is an effect which is executed once a minion is destroyed). Using a semantic network as a knowledge base, game mechanics such as “Damage” or “Healing” can be further defined, for example by having “alters” relations towards “Health”. This way, game mechanics are described into more and more abstract layers of the Static Knowledge, with only few nodes remaining for which the bot has to have hard-coded semantics. “Health”, to spin the example further, has relations describing it as a “Property” of an “Entity”, both very abstract concepts which are not directly represented in the game. The bot only has to consult a set of hard-coded rules, describing how “Properties” and “Entities” relate to a game-state, and can infer what all further game mechanics, like what “Attack” or “Health”, imply from there on. This gives a bot great flexibility to react on changes in the core game mechanics as they happen often in TCGs.

A key point of an approach that works with a vast data structure is how this data structure is created and maintained. The Static Knowledge bases of the bot are manually created. While this seems like involving both a lot of effort and the potential for a bias – leaking in domain knowledge – we found that the way the knowledge bases are structured is beneficial for the creation process. The Static Knowledge base is just describing the rules of the game in an objective way, without the need of the knowledge base author to think about what a certain property of an entity actually implies. It only needs to be described what the properties actually do. In cases where the AI is developed alongside a game and not a post-release addon (like the Hearthstone bot we present here), the description of the game mechanics themselves could already be used to create a Static Knowledge base automatically. This requires the Game Mechanics to be in a declarative format, however, in contrast to the popular approach of covering many rules of the game in scripting languages such as LUA. We experimented with such an approach in “Civilization V”[9], where game entities are described via SQL tables. After creating a small semantic structure that explains what each column in the SQL tables means, a script was able to generate the majority of the Static Knowledge automatically. Even when developing a bot for an existing game which does not follow a declarative nature, much of the effort of maintaining the Static Knowledge base can be automated. For the Hearthstone bot presented in this paper, for example, all basic card stats are supplied by a crawler, reading the game data and supplying attack and health values, having a property such as “Taunt”, costs or other simple numeric values. Only the actual card texts, which seem to be covered through scripting in Hearthstone, are then left to be manually added to the semantic structure.

[9] Firaxis Games, 2010 (http://www.civilization5.com/)

III. Reasoning Process

While the knowledge base represents domain knowledge and memories for the bot, it does not yet perform reasoning. As with many expert systems, a network of rules is put to use. These rules come in different flavors, depending on whether they describe goals, game mechanics or expert knowledge. They all contribute in calculating a utility value for an action, which is then used in an overarching utility system. As such, the bot is essentially a simple utility system calculating utilities for every valid action at a given point of time, where each utility function is a combination of rules and inferences working on the knowledge base. This follows the notion of utility systems as described by Mark and Dill [28] by mapping objective sensor data onto subjective utilities for respective actions. Within the utility calculation, there are four important types of rules involved: Refinement Rules, Goal Rules, World-State Mappings and Planning Rules.

Algorithmically speaking, there are two points of interaction between the bot and Hearthstone: any game state change on Hearthstone will invoke an `Update` call at the bot, while a `Reasoning` call is triggered during the bot’s turn and is expected to return actions to be executed. For each `Update` call, the World-State Mappings adjust dynamic knowledge to match the Hearthstone game state and all Refinement Rules are triggered. These guarantee that the bot has a consistent and correct representation of the Hearthstone game state in dynamic knowledge. Then, for each `Reasoning` call, the currently available action space is constructed. For each action, World-State Mappings are triggered to populate Planning Knowledge with a prediction on what the outcome of the respective action will be. Once completed, Refinement Rules will run over the Planning Knowledge state to produce a consistent view of the fictitious outcome. Now, the utility calculation starts. Planning Rules and Goal Rules are executed, finding patterns in the Knowledge Base and returning utility contributions. The final utility of an action in the action space is then the sum if all Planning Rule and Goal Rule contributions. The action with the best utility is then executed and the process starts anew, until the action space is empty. This chapter will describe each of these four types of rules in more detail.

A. Refinement Rules

While the Static Knowledge describes some aspects of the game mechanics, it does not yet grasp the complete complexity of a game, in particular its dynamics: How events chain, which action triggers what and how side-effects are handled. These dynamics are covered via Refinement Rules. Such a rule only describes domain knowledge, just as the Static Knowledge does. They have a formal definition consisting of a Condition and an Operation. The Condition is a pattern to look for in the knowledge base, and the Operation then adds, removes or alters relations or attributes. They are inspired by macros as found in functional languages, such as LISP, where certain sequences of an abstract syntax tree are rewritten into different representations to allow higher-level syntax. Here, Refinement Rules search for patterns in the

knowledge base and rewrite them to express what a certain constellation actually means. The Refinement Rules are called whenever a change in the knowledge base happens, such as a human player doing a move or virtual planning nodes being spawned.

A typical example would be marking dead minions in the Dynamic Knowledge. A minion instance is represented by a node in the Dynamic Knowledge and is considered dead when its "Health" is 0 or negative. If so, a relation is added that marks the respective minion to "have" "Death", on which other rules can react on. These Refinement Rules are defined in a declarative way, for example through JSON. An example of such a Refinement Rule is shown in Figure 4. Its conditions block describes that the rule matches for "has" relations with an attribute "amount" in the range of "-inf" to and including 0. Further, additional conditions for its target and origin node have to be matched. The target node has to be "Health" and the origin node has to have a path through "is_a" relations to "Minion" as well as not having a path through "has" relations to "Death". These conditions can be chained recursively to describe more complex Refinement Rules: the target and origin blocks could contain additional conditions, to define that certain relations have to be present. The operation, finally, just adds a "has" relation between the origin of the "has"-"Health" relation (which would be the respective minion) and "Death".

```
{
 "conditions": [
   {
     "type": "has",
     "attributes": [
       {"type": "amount",
        "valueMin": "-inf",
        "valueMax": 0}
     ],
     "target": {
       "name": "Health"
     },
     "origin": {
       "requiredPaths": [
         {
           "allowed": ["is_a"],
           "target": "Minion"
         }
       ],
       "forbiddenPaths": [
         {
           "allowed": ["has"],
           "target": "Death"
         }
       ]
     }
   }
 ],
 "operations": {
   "add": [
     {
       "origin": "this.origin",
       "target": "Death",
       "type": "has"
     }
   ]
 }
}
```

Figure 4: Example of a Refinement Rule adding a "has" "Death" relation to dead minion nodes in the Dynamic Knowledge.

These Refinement Rules do not typically remove relations and nodes, but rather mark them as no longer relevant by adding relations as described in the example above. Once a relation was removed, other rules cannot match whether the rule was once there, whereas just marking entities keeps the knowledge about the entity intact, while still removing it from most reasoning. There are, for example, card effects which work with dead minions, such as "Summon 7 Murlocs that died this game" (where "Murloc" is certain type of minion). Therefore, the Dynamic Knowledge keeps growing as the game continues. While this is not the most ideal scenario, it helps a lot with debugging and inspecting the current state of the bot. Even with ever growing semantic structures, the memory footprint of the bot is not a problem, as each relation is only a few bytes of data. Comparing an ever growing semantic structure to a semantic structure using rules that actually truncate relations also had no impact on the reasoning speed of the bot. Our hypothesis is that most rules reject additional relations early, such as the rule above immediately checking if the node already "has" "Death", while additional rules to truncate and clean-up the Dynamic Knowledge also require additional processing time to be applied.

B. Goal Rules

Goal rules are a set of rules that contribute to the utility calculation. They describe domain knowledge on the victory conditions of the game, of which there is only a single one in Hearthstone: bringing the hostile hero's health to or below 0. It is important to highlight that the Goal Rules only describe domain knowledge, not expert knowledge which will also influence utility calculation. The Planning Rules described below focus on dealing with that.

Goal Rules have a similar formal definition as Refinement Rules: They search for a pattern in the knowledge base, but instead of manipulating it, they output a utility value. During the utility calculation, the World-State Mappings described below create "Planning_Nodes" which are fictitious entity states describing the result of an action. These nodes have a "planning" relation towards their respective actual entity as present in the current world-state. Figure 5 shows an excerpt of how the semantic structure would look like. The goal rule would just match for a node which "is_a" "Hero" and "is_a" "Planning_Node" and the utility would just be the delta in the "amount" attributes of the relations ending in "Health" and originating from the matched node and from the node to which it has a "planning" relation, respectively. In this example 24-21 = 3. Goal rules are defined in the same formal way as Refinement Rules seen above.

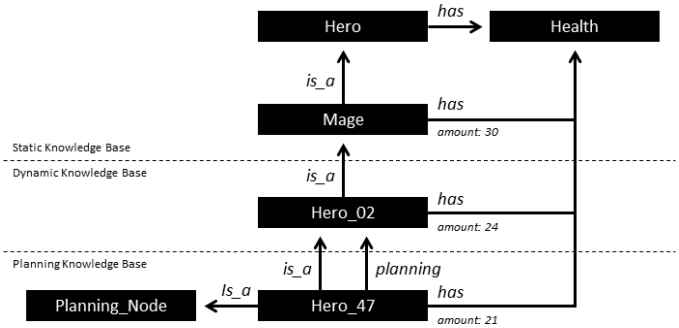

Figure 5: An excerpt of the knowledge base during utility calculation. A utility is currently calculated for an action that alters the "Health" of "Hero_02". Therefore, a fictitious "Hero_47" planning node was spawned for which other rules are now triggering and Goal Rules can produce a utility for reducing Hero "Health", by comparing the "Hero_47"-"has"-"Health" amount to the "has"-"Health" amount of node for which "Hero_47" is "planning".

C. World-State Mappings

In order to connect the bot effectively to a game, functionality needs to be implemented to connect it to a running game instance and reading its world-state. From a point of view of the reasoning process, World-State Mappings are optional: they just automate alterations on the semantic structure that, conceptually, could be performed manually to reconstruct the current game-state and apply some aspects of the game mechanics. Another important aspect of World-State Mappings is to generate the valid action space at a given point of time and spawning the planning nodes for each action when its utility calculation takes place. In the current version of the bot, World-State Mappings don't have a formal definition and are natively implemented, as they highly rely on the specific game and the connection used to read and write to it. Many of them, however, consult the Static Knowledge: When planning an "Attack" action and calculating its utility, the World-State Mappings check what an Attack actually means in the Static Knowledge and find that an "Attack" "alters" "Health", allowing them to spawn the respective planning nodes and relations.

D. Planning Rules

Planning Rules, finally, express expert knowledge and how it is applicable in a certain game-state. Such expert knowledge can be very subtle, however. A good example is a situation, where the bot has two minions on the board and it's opponent has one. The bot now has to infer whether just attacking the enemy hero is the best course of action, or whether removing the opponent minion, perhaps even sacrificing one of its own, is more effective. If reasoning would solely rely on the Goal Rules, the result would be obvious: Attacking a hero contributes towards the Goal Rule, thus netting utility, whereas attacking a minion doesn't. Such reasoning, however, ignores an important aspect of the game: Minions can attack each turn. Yet, the bot has no way to be sure whether the minion will still be available next turn, as the enemy player might remove it. This is where Planning Rules come into play: They generate utility values for actions serving as a forecast on utility value deltas for actions available next turn. In this example, a Planning Rule could check how much removal capability – such as attack value on minions – the enemy has available and how easy removing our minions for the enemy becomes. If attacking and destroying their minion with one of ours improves the odds for our minions to survive, this would net a utility value, as the bot can expect that these minions are still available next turn to potentially contribute towards a Goal Rule – dealing damage to the enemy hero. An example utility calculation with all rules involved is shown below.

Planning Rules are described in the same declarative format as Goal Rules and Refinement Rules are. Their requirements and behavior are identical to Goal Rules, they just differ in describing expert knowledge, rather than objective implications of game mechanics. Yet, at least for the Hearthstone implementation, Planning Rules ended being much more complex and computationally expensive than the Goal Rule, as they typically involve many queries towards the semantic structure. Planning Rules can also describe the meta-game, such as assigning extra utility for removing certain minions, as they might have popular synergies with spells or other mechanics.

E. Maintaining the rule-sets

All the rules described above are maintained manually. While the Planning Rules involve expert knowledge, the Refinement Rules and Goal Rules are another piece of objective information, being maintained similarly to the Static Knowledge sharing the advantage of only having to describe what something does, not what it means. Planning Rules, however, also don't describe a strategy, but just how a certain entity will probably contribute towards Goal Rules over time. As such, Planning Rules often utilize the Goal Rules. Yet, Planning Rules can always introduce a subjective bias to the bot. One such instance was actually encountered during development: An early version of the bot played some classes, particularly the Priest, very bad in comparison to other seemingly similar classes and decks. After investigating the recorded games, we noticed that the bot seems to be undervaluing healing abilities – a key concept of the Priest class. This happened due to the way healing was expressed in the Planning Rules, with its impact on keeping minions alive being forgotten. Once a few respective Planning Rules were altered and added, the win percentage of the Priest bot against the stock AI increased dramatically (from about 20% to about 80%).

IV. Utility Calculation Example

Consider a board as shown in Figure 6. The bot is playing hunter ("Rexxar") with two minions on the board: A "Boulderfist Ogre" with 6 attack and 4 health remaining and a "Kobold Geomancer" with 2 attack and 2 health. Its opponent is Mage ("Jaina Proudmoore") with a 4 attack and 2 health "Stormpike Commando" on the board. For the sake of simplicity, this example will ignore the respective player's hand and just illustrate how utility calculation for minions works. The utility calculation for playing cards is identical, just involves different Planning and Refinement Rules.

To begin reasoning, the World-State Mappings produce a

list of valid actions. Actions in Hearthstone originate from the hand, including the bot's Hero Power, and from the board. The bot's hero power, "Steady Shot", deals 2 damage to the hostile hero and does not allow specifying targets, so it ends up with two valid actions: using or not using it. In case of the game board, the bot ends up with two valid attack targets for its minions: the enemy hero and the enemy "Stormpike Commando" minion. Further, a minion could do nothing this turn, so the action space ends up consisting of 8 valid actions for which utilities have to be computed. The action with the best utility will then be executed. Afterwards, the whole process starts anew until an empty action space remains. The bot has to repeat the utility calculation, and cannot just do the n best actions, as many effects in Hearthstone have a certain randomness to them and thus are non-deterministic. Planning Rules will cover such randomness with some expectations, but once the action was actually executed, utility calculation uses the most recent world-state.

The utility calculation might start with the actions originating from the "Boulderfist Ogre". There are three actions available: "attack"-"Enemy Hero", "attack"-"Stormpike Commando", and "do nothing". The utility of each action is the sum of the utilities produced by each Goal Rule and the utilities produced by each Planning Rule.

For the "Boulderfist Ogre"-"attack"-"Enemy Hero" action, the World-State Mappings will produce the respective planning nodes, consulting the Static Knowledge on how the Game Mechanics would evaluate. No Refinement Rules are required here and, in this example, the only result of the attack is causing the enemy hero to drop by 6 health and not changing the board in any further way. Thus, the Goal Rule will trigger and produce a utility of 6. Planning Rules (which will be illustrated for the next action) would not trigger for this action so the utility for "Boulderfist Ogre"-"attack"-"Enemy Hero" ends up being 6.

The next action is "Boulderfist Ogre"-"attack"-"Stormpike Commando". Again, the World-State Mappings prepare the respective planning nodes. Then, the Refinement Rules will run and assign "has"-"Death" relations to both the fictitious planning instances of the minions, as their health would both drop to 0 as a result of the attack. In this case, the health of the enemy hero is not affected, and as such the Goal Rule does not trigger, producing a utility of 0. Yet, another change happened on the board: both the "Boulderfist Ogre" and the "Stormpike Commando" died. This is where one of the Planning Rules triggers: the capability of the enemy player to remove the remaining minion was reduced, thus increasing the likelihood for its survival. This rule iterates over all of the bot's minions and checks whether the opportunity for the enemy to remove them changed. This Planning Rule will check how much damage the enemy could deal to the minion and how that changed after executing the current action. In this example, there are two sources of damage: the enemy "Stormpike Commando" with 4 attack and the enemy hero power "Fireblast" which allows to deal 1 damage to a minion per turn. Before executing the action, the enemy had the opportunity to remove the "Kobold Geomancer", as 4 damage from "Stormpike Commando" are sufficient, thus the removal opportunity was 1. After executing the action, the "Stormpike Commando" is gone, so the removal opportunity dropped to 0, as the 1 damage from "Fireblast" is not sufficient to remove it. Thus, a delta in the expected utility for next turn took place: An attack with the "Kobold Geomancer" on the enemy hero (a Goal Rule contribution) would produce 2 utility, multiplied by the delta in removal (1-0=1). Thus, the Planning Rule will add a utility of 2*1=2 to attacking the "Stormpike Commando". Yet, the Planning Rule also triggers for the "Boulderfist Ogre". Before executing the action, the removal opportunity was 1 and after the attack we know that the minion will be dead, so it's also 1, netting a total delta contribution of 1-1=0 times its Goal Rule utilities.

Figure 6: A board situation for which the bot (Player A: "Rexxar") has to derive utilities. Player A controls two minions: "Boulderfist Ogre" on the left and "Kobold Geomancer" on the right. Player B controls only a single minion, a "Stormpike Commando"

In a real scenario, the removal opportunity of a minion is not a boolean of 0 and 1, but rather an expectation due to uncertainty: This example was fully observable and ignored the enemy hand, thus ignoring uncertainty. At game start, the World-State Mappings generate a list of deck candidates, which is a list of nodes representing cards which are allowed to be in the hostile deck (some cards are class specific or not collectible and spawned through other card's effects). If the enemy player would have had a card in their hand, the Planning Rule would have checked whether there is knowledge about this card (it could be a previously revealed card or a card returned from the board to their hand). In case of uncertainty – not knowing anything about the card – it is represented by having a "is_a" "Deck_Candidate" relation. This Planning Rule would then run over all valid "Deck_Candidates" and check whether they are sufficient to remove the minion in question. In the example of the "Boulderfist Ogre" minion instance, it would look for cards that deal 3 damage (+1 damage from "Fireblast"), 4 damage or remove the minion otherwise. Further, only cards are considered which could be played next turn, for example due to mana limitations. The contribution to the removal opportunity would then be a real value between [0,1] representing how probable it is that a such a spell – or combination of spells – is in the enemy hand. As an example, assume the removal opportunity for the "Boulderfist Ogre" is to be derived, but, unlike the example above, the opponent player B has a card in their hand and 6 Mana available. The unknown card is represented by a "Deck Candidate" and the Planning rules will now proceed in browsing through the potential cards, finding move sequences that could destroy the "Boulderfist Ogre". A Straightforward one is the spell "Fireball" that deals 6 damage for 4 Mana, sufficient to kill the Ogre. A "Frostbolt" deals 3 damage for 2 Mana, but the enemy Hero Power, "Fireblast", also deals 1 damage for 2 Mana, another valid removal combo. There are also more complex combinations, like freezing the Ogre with a "Frost Nova" and then using "Shatter" which immediately destroys a frozen minion. This "Deck Candidate" resolution algorithm is purely based on Dynamic and Static Knowledge and ignores cards that cannot be in the opponent Deck, for example as they don't belong to the Mage Class or because they were already observed twice (the maximum number a card may be contained in a Deck). Finally, the removal opportunity contribution is the probability that the respective cards are in the opponent hand. As this example only goes with one card, the removal opportunity contribution will be the number if found moves involving the "Deck Candidate" divided by the number of valid cards.

This naive approach just expected that all "Deck_Candidates" are equally probable, which is obviously not the case: some strategies dominate the current meta-game and some card combinations are much more likely to be encountered than others. This can be represented by weights being applied to the "Deck_Candidates": If some cards of a certain popular deck were encountered already, the odds for seeing more cards of such a deck would rise. This offers opportunities to combine the approach with ML or data mining techniques looking for cards that often appear together in decks.

If the utility calculation as described above is repeated for each of the other actions, the bot will end up with utility values as shown in Figure 7. In the first iteration, two actions are tied for the best utility: "Boulderfist Ogre"-"attack"-"Hero" and "Kobold Geomancer"-"attack"-"Stormpike Commando". In case of a tie, one of the winning actions is chosen randomly. Say the bot will attack with the ogre first, then the utility calculation will start again. Yet this time, the World-State Mapping will not produce any actions for the "Boulderfist Ogre", as it already performed an action this turn and is thus marked as exhausted, unable to do anything else. For this example, the utility calculation after the "Boulderfist Ogre"-"attack" will look the same for the "Kobold Geomancer" actions and the bot will now attack the "Stormpike Commando" with its "Kobold Geomancer". Once the Gamestate Mappings updated dynamic knowledge, only the "Steady Shot" actions remain and the bot will fire it at the enemy hero. Now, finally, the World-State Mapping will produce an empty action space and the bot will end its turn. In this small example just utilizing a hand full of rules and the semantic structure, the bot acted according to a fundamental principle of Hearthstone strategies: Board presence.

Action	Goal Rules Utility	Planning Rules Utility	Total Utility
Ogre -> Hero	6	0	6
Ogre -> Commando	0	0 + 2	2
Ogre do nothing	0	0	0
Kobold -> Hero	2	0	2
Kobold -> Commando	0	0 + 6	6
Kobold do nothing	0	0	0
Steady Shot -> Hero	2	0	2
Steady Shot do nothing	0	0	0

Figure 7: Utility values for all available actions in the example as described above. The attack-Commando actions consist of two planning utilities: One for the attacking minion – which dies in both cases, and one for the other minion which is now harder to remove.

This example illustrates how a utility system together with rules and a semantic structure implements a symbolic AI for Hearthstone. Yet, the bot currently only solves the runtime part of playing Hearthstone, without solving the meta-game itself: deck construction. Skipping the complete meta-game of deck building, the bot currently relies on having somebody to provide a deck for it. We just started work on a deck building system, similar to a recommender system that could construct decks based on the Static Knowledge exploiting cost efficiency, mana curves, synergies and other metrics between cards. This might serve as an interesting alternative to other researched approaches towards deck building, such as genetic algorithms or ML on human games. Such a deck-building mechanism could also be used to improve the calculation of probabilities for unknown entities, such as the deck candidates, as there are certain cards per class which are very likely to be contained in a deck. There are also certain groups of cards which are often used in a deck together, as they form

strong synergies. This could also be exploited to arrive at better probability calculations for the Planning Rules.

V. Experimental Evaluation

The evaluation of the symbolic approach consists of four major stages. At first, we will compare the performance against the built-in AI of Hearthstone and an MTCS implementation, with all players playing the Mage class. In a second round, we will assess how well the bot generalizes to different hero classes, each featuring some unique strategies and decks. As a final effectivity test, we will put the bot against human opponents of various skill levels. To gather information on the second important metric of a "good" player-level AI, we performed a pseudo-Turing test. For each of the evaluations, we also gathered runtime performance statistics to support this secondary, but engineering-wise vital metric in game development.

A. MCTS Comparison

Ward et al [6] have demonstrated an application of MCTS for a minimized version of "Magic: The Gathering", a popular physical TCG sufficiently similar to Hearthstone. The implementation we opted for follows the analysis by Cowling et al [5], representing an MCTS with UCT. We went for an n-degree tree, as this allowed us to easily reuse the action-space generation functionality we already built for the symbolic AI. The MCTS bot also uses determinization such as Cowling et al report in their paper, using the same ratio between number of determinizations and simulations they found to be optimal for their scenario: 40 determinizations with 250 simulations each. For each determinization, the outcome of all random events was fixed, such as the deck ordering or the roll for random damage numbers. To determinize the hostile decks, a random but valid hostile deck was constructed for each determinization set. We did not supply any kind of action filtering, though, as we did not want to take the ability away to use some of Hearthstone's more complex synergies, such as transforming damage into healing and then casting a damage spell on a friendly target or even the friendly hero itself. Noticeably, such seeming suicidal actions can be about half the action space in Hearthstone. For the simulations, we used a purely random player. Victories were awarded with a score of +1 and losses or draws (as rare as they are) were both rewarded with a score of 0.

In the test series, we fielded both the symbolic AI and the MCTS implementation against the expert AI of Hearthstone, in order to get some comparison if the symbolic AI can compete with a simple MCTS implementation. All three players were fixed to play the Mage hero-class. While the Hearthstone Expert AI used their deck as listed on the Hearthstone wiki[10], both the symbolic AI and the MCTS implementation used an optimized beginner's deck that does not include any cards that would have to be unlocked first. This deck is considered to be a powerful start deck by the community, although it lacks many of the powerful unlocked cards that other decks – such as the Expert AI – put to use. Its details are described on the community page IcyVeins[11]. Both the MCTS and the symbolic bot played 101 test games against the expert AI. As the rules of Hearthstone define, the starting player was chosen randomly. The results are detailed in figure 8 and show that the symbolic approach won 65% of the games, while the MCTS implementation won 36%. There are, however, many ways in which the MCTS implementation could be improved, for example by switching over to binary trees as Cowling et al recommended for their "Magic: The Gathering" implementation [5]. They also utilize more complex simulations and determinizations, such as trying to identify interesting deck compositions and not just performing random playouts. Thus, a more fine-tuned MCTS implementation would probably lead to a stronger win percentage. However, this test series further supports our claim that a Symbolic AI can compete with MCTS approaches in game domains such as Hearthstone. In particular, throughout the test games we observed some interesting behavior by the MCTS bot. In some occasions, it followed a suicidal action, such as casting a damaging spell on its own minions. A closer investigation of some of these situations yield an interesting result: most of them had a particularly rule-changing card in their determinization setup. In Hearthstone, there are many cards that dramatically alter the game state, such as destroying all minions on the board, converting healing into damage or swapping all health and attack values. This is a greater change in fundamental rules and game mechanics as encountered in many other game domains. If such a card, for example "Twisting Nether" destroying all minions, is included in the determinization deck, any action performed in the current state might actually have less or no impact on the future a few turns later. Hearthstone features many of these large-scale game changer cards and every expansion keeps adding more of them. Thus we argue that a symbolic approach might compete with MCTS in game domains in which the underlying rules can change to such a degree that it becomes difficult to assess what an action's outcome will be at runtime.

Bot	Games Played	Games Won	Win Percentage
Symbolic Expert System	101	65	65% (±9.2)
Monte Carlo Tree Search	101	36	36% (±9.2)

Figure 8: Comparison of a simple MCTS system and the symbolic Expert System both playing against the built-in Expert AI. Error margins are computed through the Adjusted Wald Method for a target confidence level of 95%.

B. Class Coverage

The first test series compared both the Symbolic Expert System and the MCTS implementation against the stock Hearthstone AI. These tests were conducted within the actual Hearthstone client. However, due to technical and license restrictions, these tests could not be entirely automated: while both AIs read from the Hearthstone game-state, their input still

[10] http://hearthstone.gamepedia.com/Practice_mode

[11] http://www.icy-veins.com/hearthstone/basic-mage-deck

had to be executed by a Human proxy, limiting the number of tests that could be conducted. Being able to play with the official client was important, in particular for tests versus Humans and to assess believability (see sections V-C and V-D). In order to achieve a better statistical coverage, we opted to implement a Hearthstone simulation – minus any visual representation – allowing fully automated test series.

In this test series, we let the Symbolic Expert System and the MCTS implementation compete directly against each other playing as the nine available classes. For each pairing, 400 games were conducted, leading to a total of 32400 games. Both bots used the same optimized beginner decks for the classes as described in section V-A. They can be found on IcyVeins[12]. The win rates of the Symbolic AI for each pairing are illustrated in Figure 9. At first glance, the Symbolic Expert System beats the MCTS implementation in all test series. Interestingly though, the performance varies greatly from class to class. Some of these variances might originate in the natural imbalance of Hearthstone, in particular for the chosen decks: a deck is usually not equally effective against each possible opponent playstyle. Shaman, for example, was a deck that showed strong plays for both contestants. However, while the Symbolic AI operates on a similar level with Rogue and Shaman (overall win percentages of 86% and 83%), the MCTS implementation performs far better as a Shaman – with an overall win percentage of 35% it's actually the favored class of the MCTS bot by far, followed with some distance by Hunter (overall win percentage of 29%) and Paladin (overall win percentage of 27%). Rogue is in fact the third weakest of the MCTS classes, performing at a low 17% win percentage. It's interesting to see that, while the Symbolic bot performs at about equal level with the two classes, there is a great discrepancy in the MCTS results. An explanation might be found in the different playstyles of these classes. The Shaman class in Hearthstone, as well as the Hunter, usually plays very aggressively, trying to maximize damage dealt over the course of a game. This is a pretty strong contrast to the combo and synergy oriented gameplay of classes like the Rogue. This could hint at the MCTS bot being more effective when it comes to longer-term planning to maximize damage and board presence, whereas the Symbolic Expert System might excel in short-term decision making, even for complex synergies and game mechanics relations.

Looking at the overall results again shows that the Symbolic Expert System plays all classes well against the MCTS implementation, ranging from an overall win percentage of 73% while playing Druid to a win percentage of 85% while playing Rogue. The best performance was observed in the Symbolic Shaman vs. MCTS Priest series and the worst performance surfaced in the Symbolic Warrior vs. MCTS Shaman series.

[12] http://www.icy-veins.com

Symbolic Druid victories against MCTS …								
Dr	**Hu**	**Ma**	**Pa**	**Pr**	**Ro**	**Sh**	**Wl**	**Wa**
273	262	324	237	355	317	271	300	292
68%	66%	81%	59%	89%	79%	68%	75%	73%
±4.5	±4.6	±3.8	±4.8	±3.1	±4.0	±4.6	±4.2	±4.3
Symbolic Hunter victories against MCTS …								
Dr	**Hu**	**Ma**	**Pa**	**Pr**	**Ro**	**Sh**	**Wl**	**Wa**
341	286	351	314	375	333	243	311	313
85%	72%	88%	79%	94%	83%	61%	78%	78%
±3.5	±4.4	±3.2	±4.0	±2.4	±3.7	±4.8	±4.1	±4.0
Symbolic Mage victories against MCTS …								
Dr	**Hu**	**Ma**	**Pa**	**Pr**	**Ro**	**Sh**	**Wl**	**Wa**
312	320	342	292	381	337	255	312	318
78%	80%	86%	73%	95%	84%	64%	78%	80%
±4.1	±3.9	±3.5	±4.3	±2.2	±3.6	±4.7	±4.1	±4.0
Symbolic Paladin victories against MCTS …								
Dr	**Hu**	**Ma**	**Pa**	**Pr**	**Ro**	**Sh**	**Wl**	**Wa**
299	259	336	259	363	310	252	296	289
75%	65%	84%	65%	91%	78%	63%	74%	72%
±4.2	±4.7	±3.6	±4.7	±2.9	±4.1	±4.7	±4.3	±4.4
Symbolic Priest victories against MCTS …								
Dr	**Hu**	**Ma**	**Pa**	**Pr**	**Ro**	**Sh**	**Wl**	**Wa**
319	274	341	303	375	327	240	295	318
80%	69%	85%	76%	94%	82%	60%	74%	80%
±3.9	±4.5	±3.5	±4.2	±2.4	±3.8	±4.8	±4.3	±4.0
Symbolic Rogue victories against MCTS …								
Dr	**Hu**	**Ma**	**Pa**	**Pr**	**Ro**	**Sh**	**Wl**	**Wa**
337	306	376	319	386	356	317	347	348
84%	77%	94%	80%	97%	89%	79%	87%	87%
±3.6	±4.2	±2.4	±3.9	±1.9	±3.1	±4.0	±3.3	±3.3
Symbolic Shaman victories against MCTS …								
Dr	**Hu**	**Ma**	**Pa**	**Pr**	**Ro**	**Sh**	**Wl**	**Wa**
324	316	367	316	387	349	283	312	345
81%	79%	92%	79%	97%	87%	71%	78%	86%
±3.8	±4.0	±2.7	±4.0	±1.8	±3.3	±4.4	±4.1	±3.4
Symbolic Warlock victories against MCTS …								
Dr	**Hu**	**Ma**	**Pa**	**Pr**	**Ro**	**Sh**	**Wl**	**Wa**
315	282	359	302	381	335	261	323	315
79%	71%	90%	76%	95%	84%	65%	81%	79%
±4.0	±4.5	±3.0	±4.2	±2.2	±3.6	±4.6	±3.9	±4.0
Symbolic Warrior victories against MCTS …								
Dr	**Hu**	**Ma**	**Pa**	**Pr**	**Ro**	**Sh**	**Wl**	**Wa**
291	255	349	278	376	337	233	288	332
73%	64%	87%	70%	94%	84%	58%	72%	83%
±4.4	±4.7	±3.3	±4.5	±2.4	±3.6	±4.8	±4.4	±3.7

Figure 9: Effectiveness evaluation against the MCTS AI in the Hearthstone simulation. The figure shows victories / victory percentage / error margin of the Symbolic Expert System of a fixed class against the MCTS implementation playing each of the 9 hero classes (Dr: Druid, Hu: Hunter, Ma: Mage, Pa: Paladin, Pr: Priest, Ro: Rogue, Sh: Shaman, Wl: Warlock, Wa: Warrior). For each pairing, 400 test games were conducted. Error margins are computed through the Adjusted Wald Method for a target confidence level of 95%.

C. Human opponents

To test how well the bot fares in games against human players, we performed a series of test games against players of different skill and experience levels. The bot used a randomly selected class and the same premade decks that were used in the experiment against the stock AI detailed above. The human players were allowed to play whatever class or deck they prefer and usually play with. To assess their skill level, we used the player rank assigned through Hearthstone. When playing against other humans in the typical Hearthstone game modes, a player accumulates points for each victory, raising in rank. Ranks start at 25 and ascend to rank 1, finally reaching Legend, while losing games past rank 20 will cause you to lose points. According to the developer, Blizzard

Entertainment, 75% of all players are in the rank brackets between 25 and 15, 17.5% between 15 and 10, 5.5% between 10 and 5 with the remaining 2.5% of the player base scoring at rank 5-Legend[13]. The exact number of players actively playing Hearthstone is unknown, but in 2015 Blizzard Entertainment announced reaching a playerbase of 30 million[14]. The bot performed well in the Rank 25-20 bracket with a win rate over 90%, then decreasing down to only winning a single game against the Legend-ranked player. A detailed chart is shown in Figure 9. A frequent feedback we got was that, while the bot was playing its cards sufficiently well, it was lacking many of the powerful cards that players unlock as they rise in ranks. A rank 1 player noted that "Die KI hat schlechte Karten und ein paar Fehler gemacht, würde es aber sicher auf Rang 10 schaffen" (en: "the AI had bad cards and did some mistakes, but would certainly be able to reach Rank 10"). We are not that optimistic, as the bot only won 25% of the games in the rank 14-10 bracket, but reaching rank 15 seems to be realistic, which, according to the chart published by Blizzard, would place the bot among the top 25% of human players as players stabilize at a rank winning about half the games they play.

Rank Bracket	Number of Human Players	Games Played	Games Won	Win Percentage
25-20	5	45	41	91% (±9.2)
19-15	5	45	31	68% (±13.2)
14-10	2	24	6	25% (±16.8)
9-5	2	24	2	8% (±12.9)
4-1	1	12	1	8% (±19.1)
Legend	1	15	1	6% (±16.3)

Figure 10: Effectiveness evaluation against Human players, detailed results per rank bracket. Error margins are computed through the Adjusted Wald Method for a target confidence level of 95%.

D. Believability

While the above study with human players showed that the bot can play Hearthstone sufficiently well to achieve victory, it didn't yet show whether the bot follows strategies and executes moves that a human player would also do. We therefore performed another round of tests to assess the bot's believability through pseudo-Turing tests [16]: Human participants of various skill levels played against either the bot or a rank 14 player and, after five games, had to give their impression against whom they played. Whether a game was played by the bot or the rank 14 player was chosen randomly and both used a randomly selected deck from the Hearthstone Expert deck pool. Human participants were allowed to use any deck and were told that winning was not important for this test series, but rather should they aim to find out against whom they play, leading to some players actually forging "probing" decks where they aimed to confront a potential AI with difficult decisions. The result of the believability evaluation is shown in Figure 11. Of the 41 games the bot played, it was mistaken for a humans in 41% percent of the games, indicating that the bot plays believable.

[13] http://us.battle.net/hearthstone/en/blog/15955974/hearthside-chat-youre-better-than-you-think-9-18-2014

[14] https://twitter.com/PlayHearthstone/status/595619019593416704/photo/1

		Human player identified their actual enemy ...			
Rank	**Games Played**	**Bot as Bot**	**Human as Human**	**Bot as Human**	**Human as Bot**
25	5	2	0	1	2
25	5	3	2	0	0
25	10	2	4	4	0
20	10	3	2	3	2
20	5	1	1	3	0
17	5	1	2	1	1
14	10	3	2	3	2
12	10	3	4	2	1
6	10	6	4	0	0
Sum	*70*	*24 (34%)*	*21 (30%)*	*17 (24%)*	*8 (11%)*

Figure 11: Results of a Pseudo-Turing test. Each row represents a test series with one participant.

E. Runtime Performance

Runtime performance was not a primary criterion for evaluation, as Hearthstone is turn-based and gives players a lot of time to think. The current implementation of the bot in VB.Net still has potential for optimization, but already performs its complete reasoning in less than 50ms on a 2.5 GHz core, which is well sufficient for Hearthstone. Many of the queries on the semantic structure could be parallelized, of which we expect great potential to improve the runtime performance by utilizing multiple cores. Another direction we are currently investigating is introducing several caches to avoid computation of some expensive queries on the semantic structures. In particular within the planning rules, when taking card candidates of the hostile player's hand into account, many queries could be cached, at least within the current Planning Rule: Hearthstone is static and thus the world-state will not change during planning. Yet, exploiting the declarative nature of how cards are described in the Static Knowledge, their side effects could be tracked and further delay the cache invalidation even through several turns. In the test games of which results are shown in Figure 9, the peak memory consumption was 71 MiB, recorded during a Planning Rule playing against a Warlock deck which typically plays with a large hand size. The average memory consumption throughout all games was 55 MiB. The longest reasoning time on a single 2.5 GHz core was 47 ms, recorded during a game session with a full enemy board and many triggered abilities causing chain reactions. The fastest reasoning time encountered, typically at the start of a game session, was 3 ms. Figure 12 sums up these findings.

Metric	Min	Max	Average
Memory Usage	21 MiB	73 MiB	55 MiB
Reasoning Time	3 ms	46 ms	18 ms

Figure 12: Runtime Performance metrics of 279 test games fielding the Symbolic Expert System against the Stock Hearthstone AI.

Throughout the 202 test games the bots performed in the MCTS comparison (101 per bot implementation), we also kept record of some performance statistics. The MCTS implementation used a budget of 10000 simulations, split to 250 simulations per 40 determinization sets. Hereby, the reasoning time of the MCTS was significantly slower than the symbolic implementation, but still sufficiently fast for a turn-

based game such as Hearthstone. Additional optimizations, particularly when fusing the trees of several determinization sets, could greatly reduce the reasoning time. Details are shown in figure 13.

Bot	Min	Max	Average
Symbolic Expert System	5 ms	67 ms	24 ms
Monte Carlo Tree Search	257 ms	21050 ms	12088 ms

Figure 13: Reasoning time metrics for the 202 test games performed in the MCTS comparison test.

VI. Conclusion and Outlook

We described how semantic structures can be an effective representation for a symbolic AI, mapping both static domain knowledge and dynamic runtime memories in the same data structure. Using a simple reasoning algorithm we were able to show a bot for Hearthstone that bases most of its reasoning on operations on the semantic structure and a small set of rules. In a comparison we could show that such a symbolic agent can compete with MCTS for this specific game domain. The effectiveness evaluation against the Hearthstone Expert AI and Human players showed that the bot can play the game sufficiently well with all classes and would probably end up in the top half of players if it would participate in the Hearthstone online league. The believability evaluation via a Pseudo-Turing test showed that the bot plays sufficiently human, too. Thus, we conclude that the presented approach can be used to implement a good player-level AI, if "good" is defined to cover an AI that plays both efficiently and believable.

While this paper followed an implementation of a Symbolic AI for Hearthstone, the approach is also applicable for other games of the TCG game domain and perhaps even other genres. In general, as mentioned above, the approach seems to be well suited for game domains in which the game mechanics can change significantly at runtime. We have already demonstrated an application for the RTS "StarCraft 2"[15] [17, 18], where a symbolic reasoning system derives build items to produce from a Static and Dynamic Knowledge Base and a collection of rules. The StarCraft 2 implementation is very similar, but did not yet do the split between Expert and Domain knowledge as explicitly. This is due to the fact that the game mechanics of StarCraft 2 are less prone to change during a game session and if such changes happen, like an upgrade being researched that alters how an ability works, they are more predictable than card effects in Hearthstone. The StarCraft 2 implementation focused especially on teaming the bot up with a human player, thus mapping human statements like "I will take care of air defense" on knowledge of the bot, interpreting it as Game Mechanics changing. Another application of this approach is currently being developed for the 4X game Civilization 5 used for an assistant AI acting as a governor to which a player can transfer the control of a city to. The Civilization 5 version of the bot comes with another category of rules mapping human information, as human and bot cooperation is a vital aspect when using the bot to serve as an assistant to the human player.

In the TCG game domain, one of the aspects we didn't yet mention is deck construction. Unlike the runtime reasoning presented here, deck construction is highly dependent on the "meta game", thus which cards and strategies are currently popular. Thus, it is harder to formulate explicit Static Knowledge about this aspect of Hearthstone. We are currently working on a deck construction system and reported early finding recently [27]. Our overall concept is to integrate deck construction and runtime play. We hope to achieve this by transferring knowledge on why a specific card was included from deck construction to the runtime system. This seems relevant as there are many ambiguous cards in Hearthstone that can be used in very different ways.

In terms of runtime reasoning, we hope to increase the reasoning speed further. Our plans are to combine the symbolic system with MCTS. Here, MCTS could decide which cards to play, while the symbolic system selects targets for those cards and performs attacks and similar actions. This would greatly reduce the branching factor of the resulting MCTS tree, as each action now only consists of whether to play a card or not, without having to deal with the different targets a card could be used at. On the symbolic end, the whole reasoning is split into much smaller, more parallelizable steps. Our rationale is that a combination with MCTS might help the bot in executing moves that require longer-term planning. If the symbolic agents become fast enough, they might even be suitable for a simulation policy. Combining the symbolic AI with other approaches already used in the field seems like an interesting research direction to continue this work.

[15] Blizzard Entertainment, 2010 (http://us.battle.net/sc2/en/)

References

[1] Ontanón, Santiago, and Synnaeve, Gabriel, and Uriarte, Alberto, and Richoux, Florian, and Churchill, David, and Preuss, Mike. "A survey of Real-Time Strategy Game AI Research and Competition in Starcraft." IEEE Transactions on Computational Intelligence and AI in Games (2013): 293-311.

[2] Buro, Michael. "Call for AI Research in RTS Games." Proceedings of the AAAI-04 Workshop on Challenges in Game AI (2004): 139-142.

[3] Buro, Michael, and Furtak, Timothy. "RTS Games as Test-Bed for Real-Time AI Research." Proceedings of the 7th Joint Conference on Information Science (2003): 481-484.

[4] Yannakakis, Geogios N. "Game AI revisited." Proceedings of the 9th conference on Computing Frontiers (2012): 285-292.

[5] Cowling, Peter I., and Ward, Colin D., and Powley, Edward J. "Ensemble Determinization in Monte Carlo Tree Search for the Imperfect Information Card Game Magic: The Gathering." IEEE Transactions on Computational Intelligence and AI in Games (2012): 241-257.

[6] Ward, Colin D., and Cowling, Peter I. "Monte Carlo Search applied to Card Selection in Magic: The Gathering." Proceedings of the IEEE Symposium on Computational Intelligence and Games (2009): 9-16.

[7] Goes, Luis Fabricio Wanderley, and da Silva, Alysson Ribeiro, and Rezende, Joao, and Amorim, Alvaro, and Franca, Celso, and Zaidan, Tiago, and Olimpio, Bernardo, and Ranieri, Lucas, and Morais, Hugo, and Luana, Shirley. "HoningStone: Building Creative Combos with Honing Theory for a Digital Card Game." IEEE Transactions on Computational Intelligence and AI in Games (2016).

[8] Churchill, David, and Buro, Michael. "Hierarchical Portfolio Search: Prismata's Robust AI Architecture for Games with Large Search Spaces." Proceedings of the Artificial Intelligence in Interactive Digital Entertainment Conference (2015).
[9] Bursztein, Elie. "I am a Legend: Hacking Hearthstone with Machine Learning." Talk at Defcon (2014).
[10] Weber, Ben George, and Mateas, Michael. "A Data Mining Approach to Strategy Prediction." Proceedings of the IEEE Symposium on Computational Intelligence and Games (2009): 140-147.
[11] Wender, Stefan, and Watson, Ian. "Applying Reinforcement Learning to small scale Combat in the Real-Time Strategy Game Starcraft: Broodwar." Proceedings of the IEEE Conference on Computational Intelligence and Games (2012): 402-408.
[12] Cadena, Pedro, and Garrido, Leonardo. "Fuzzy Case-Based Reasoning for Managing Strategic and Tactical Reasoning in Starcraft." Advances in Artificial Intelligence (2011): 113-124.
[13] Weber, Ben George, and Mateas, Michael. "Case-Based Reasoning for Build Order in Real-Time Strategy Games." Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (2009).
[14] Schwartz, Steven. "Principles of Semantic Networks." (1984).
[15] Orkin, Jeff. "Data-Driven Digital Actors." Keynote on the IEEE Conference on Computational Intelligence and Games (2012).
[16] Livingstone, Daniel J. "Turing's Test and Believable AI in Games." Computers in Entertainment (2006): 6-19.
[17] Stiegler, Andreas, and Livingstone, Daniel J., and Maucher, Johannes, and Dahal, Keshav "Starcraft II Build Item Selection with Semantic Nets." Proceedings of the GAMEON conference (2015): 55-60.
[18] Stiegler, Andreas, and Livingstone, Daniel J. "AI and Human Player Cooperation in RTS games." Proceedings of the Foundations of Digital Games conference (2013): 449-450.
[19] Mahlmann, Tobias, and Togelius, Julian and Yannakakis, Georgios N. "Evolving Card Sets towards Balancing Dominion." Proceedings of the IEEE Congress on Evolutionary Computation (2012): 1-8.
[20] Allis, Louis V.. "Searching for Solutions in Games and Artificial Intelligence." Ponsen & Looijen (1994).
[21] Robertson, Glen, and Watson, Ian. "A Review of Real-Time Strategy game AI." AI Magazine 35.4 (2014): 75-104.
[22] Browne, Cameron B, and Powley, Edward, and Whitehouse, Daniel, and Lucas, Simon M, and Cowling, Peter I, and Rohlfshagen, Philipp, and Tavener, Stephen, and Perez, Diego, and Samothrakis, Spyridon, and Colton, Simon. "A Survey of Monte Carlo Tree Search Methods." IEEE Transactions on Computational Intelligence and AI in Games (2012): 1-43.
[23] Ginsberg, Matthew L. "GIB: Imperfect Information in a computationally challenging Game." Journal of Artificial Intelligence Research 14 (2001): 303-358.
[24] Buro, Michael, and Long, Jeffrey R., and Furtak, Timothy, and Sturtevant, Nathan R. "Improving State Evaluation, Inference, and Search in trickbased Card Games." Proceedings of the International Joint Conference on Artificial Intelligence (2009): 1407–1413
[25] Furtak, Timothy, and Buro, Michael. "Recursive Monte Carlo Search for Imperfect Information Games." Proceedings of the IEEE Conference on Computational Intelligence in Games (2013): 1-8.
[26] Long, Jeffrey R., and Sturtevant, Nathan R., and Buro, Michael, and Furtak, Timothy. "Understanding the Success of Perfect Information Monte Carlo Sampling in Game Tree Search." Proceedings of the AAAI Conference on Artificial Intelligence (2010).
[27] Stiegler, Andreas, and Messerschmidt, Claudius, and Maucher, Johannes, and Dahal, Keshav. "Hearthstone Deck-Construction with a Utility System." Proceedings of the 10th International Conference on Software, Knowledge, Information Management & Applications (2016).
[28] Mark, Dave, and Dill, Kevin. " Improving AI Decision Modeling Through Utility Theory", Talk at the Game Developers Conference (2010).