Spring 6-14-2024

Cards with Class: Formalizing a Simplified Collectible Card Game

Dan Ha
Portland State University

Recommended Citation
Ha, Dan, "Cards with Class: Formalizing a Simplified Collectible Card Game" (2024). *University Honors Theses.* Paper 1500.
https://doi.org/10.15760/honors.1532

Cards with Class: Formalizing a Simplified Collectible Card Game

by

Dan Ha

An undergraduate honors thesis submitted in partial fulfillment of the

requirements for the degree of

Bachelor of Science

in

University Honors

and

Computer Science

Thesis Advisor

Bart Massey, Ph.D.

Portland State University

2024

1 Introduction

How can one minimize errors and ambiguities in a system with thousands of interacting parts? If that system is a game, how can designers ensure that it remains fun to play as well? These are the challenges that the designers behind *Hearthstone* [1, 2], *Magic: The Gathering* [3, 4], and many similar collectible card games (CCGs) tackle every day.

With thousands of cards and countless more interactions between cards to manage, it is a tall order to ensure that *every single one* is implemented properly and that its card text unambiguously conveys its behavior in all cases. If this implementation and clarity check is done inadequately, the resulting confusion interferes with players' enjoyment of the game, and designers must spend extra resources to fix the errors and clarify any ambiguities. I propose formal methods as a way to help alleviate these challenges. In this work, I present a specification for a reduced version of *Hearthstone* that can be given to general theorem provers and expanded upon as a proof of concept.

2 Background

Because collectible card games and formal methods have historically been fields with little overlap, no knowledge of either is assumed in this paper. Brief introductions to both fields and to my proposed connection between them will therefore be provided.

2.1 Collectible Card Games

CCGs are adversarial, usually 2-player games. They typically involve players taking turns placing cards on a game board and applying the effects written on each one. These effects could be as simple as "deal 6 damage [to a target of your choice]" as in *Hearthstone*'s Fireball (Fig. 1a), or as complex as the multiple paragraphs on *Yu-Gi-Oh*'s Astrograph Sorcerer (Fig. 1b).

(a) The "Fireball" card [5]

(b) The "Astrograph Sorcerer" card [6]

Most CCGs split gameplay into 2 distinct domains: choosing what cards to include in one's deck (deck-building) and using those decks in matches against other players. In each match, players must reach some win condition with the added challenge of only having access to different subsets of their randomly-shuffled deck at any moment. Players must constantly adapt their strategies based on

what they currently have available and what their opponents may have. Within the genre, each game may create different kinds of cards, have different rules dictating how they interact, and guide deck-building with different restrictions. Nevertheless, the core gameplay of building decks and competing against other players with them remains constant.

(a) The final match of the 2023 *Magic: The Gathering* World Championships. [7]

(b) Professional *Legends of Runeterra* player MaijinBae on his Twitch live stream. [8]

Figure 2: Screenshots of some CCG gameplay

Although the concepts were explored in earlier games such as Steve Jackson's *Illuminati*, the release of Wizards of the Coast's *Magic: the Gathering* (also known as simply "*Magic*") in 1993 is widely considered the start of the CCG genre [9, 10]. The genre has since flourished with the creation of many new games, both physical and digital. Other primarily physical CCGs (better known as "trading card games" or "TCGs") include Legend Story Studios' *Flesh and Blood* as well as Bandai Namco's *Dragon Ball Super Card Game* and *Digimon Card Game* [11, 12, 13].

Digital CCGs are video games that simulate the TCG experience, often with new twists or added visual flair. They have lower barriers to entry as players do not need dozens of physical cards to play. This format also allows designers to experiment with new card effects that would be impossible or cumbersome to carry out physically. The most common example of this is randomly generating cards that did not start in either player's decks.

Figure 3: The "Lonely Poro" [14]

For instance, *Legends of Runeterra*'s Lonely Poro (Fig. 3) could add *any* card that fits the description of "1 cost Poro" to its player's hand, even if the player does not have it in their deck or collection. In a TCG, this would be difficult to carry out without constantly referencing an online card database and using a random number generator. Players would have to own more cards than they strictly need to play, thus further raising the barrier to entry. However, in a digital CCG, the database and a generator can be built into the game, making these kinds of effects trivial to implement.

Examples of entirely digital CCGs include Blizzard Entertainment's *Hearthstone*, CD Projekt's *Gwent: The Witcher Card Game*, and Riot Games' *Legends of Runeterra* [15, 16, 17]. Some titles

like *Magic*, Konami's *Yu-Gi-Oh*, and the *Pokémon Trading Card Game* are even popular enough to support massive playerbases both in person and on online simulators of the physical games [18, 19].

The main way that CCGs keep players interested and grow is by occasionally releasing sets of new cards. New cards bring a whole new set of interactions for players experiment with, inspiring new ways to build decks and play the game. This increasing complexity is what makes these games so enticing for many; it sets up creative playgrounds for both players and designers to display their ingenuity and personality through card design, deck-building choices, and playstyle.

2.2 Natural Language in CCG Rules

As more new cards experiment with some new twist on old rules and design tropes, it is exceedingly easy for overlooked edge cases and ambiguous interactions to sneak in unexpectedly. Situations where it is unclear how to complete an action or resolve an apparent conflict between card effects become more likely with every new set. These conflicts cause confusion among players, which interrupts their fun as they must stop playing to figure out what should happen based on unclear texts. It can also be frustrating for players to develop an exciting new strategy only to find out that it does not work due to a misunderstanding of ambiguous wording or due to the edge case not being properly explained. Such ambiguities also weigh on game designers, who must take time away from developing new card sets to decide what should happen and quickly announce the clarification to players to prevent further misunderstandings.

For example, in *Hearthstone*, the Acolyte of Pain card (Fig. 4a) lets its owner draw 1 card every time it takes damage. However, what happens if the damage would kill the Acolyte? Does it have to survive for its owner to benefit from its effect, or can they draw even if the Acolyte would have already died? In this case, the answer is the latter, and this effect can be taken at face value.

A much more ambiguous example involves the Shattered Sun Cleric (Fig. 4b), which permanently increases both of an ally minion's stats by 1 point. What happens if an effect that "silences" (nullifies effects and stat increases) like the Ironbeak Owl's (Fig. 4c) is then used on the Cleric's target? There are two possible scenarios: one where the target minion's stats have not changed since being boosted, and one where they have. In the former, the result is simple to infer: the minion's stats return to the values they had before the Cleric's boost.

However, in the second scenario, the answer is much less obvious. Should the target minion's health stay at the same value when the silence is applied, or should it decrease by the boost's value? If it is the latter, what happens if the effect would reduce the minion's health to 0? Should silence effects be able to kill their targets in situations like this? The answer to that has significant ramifications for what designers must do to keep these effects from being too powerful. In this case, the affected minion's health will remain at the current value even if the boost is removed as silences are purely meant to negate effects.

In physical TCGs, these situations are usually resolved with addendums to the existing rulebook that codify certain impromptu ruling decisions or by updating cards to have clearer text. *Yu-Gi-Oh* and *Legends of Runeterra* commonly update cards according to modern clarity standards, while *Magic* instead has a massive document of rulings online for players to reference. Digital CCGs that

(a) The "Acolyte of Pain"

(b) The "Shattered Sun Cleric"

(c) The "Ironbeak Owl"

Figure 4: The *Hearthstone* cards mentioned in the examples from Subsection 2.2 in order of appearance from left to right [20, 21, 22]

handle card interactions automatically can mitigate this issue, but often the only way to clarify these situations is to start a match and investigate the results for oneself. Learning of and fixing these issues early helps ensure that designers can focus on delivering high-quality gameplay experiences and better enables players to focus on crafting and playing with exciting and interesting decks. This early awareness of and communication about similarly complex situations can be facilitated by formalizing the game rules and card effects.

2.3 Formal Methods

Formal specifications are a tool used by computer scientists to facilitate the design and implementation of complex computer systems and software. They allow developers to describe systems in formal languages built from mathematical and logical principles, then rigorously test and analyze those models using techniques from those fields [23]. Such analyses often use special tools associated with the chosen language and typically involve proving various properties about the system like whether every state it can exist in is valid or possible. These analyses are meant to help ensure that the software will run as desired and expected under any circumstance.

As the existence of digital CCGs and simulators shows, these games can be represented as computer programs, which implies that they can also be described in a formal language. It would therefore be possible to use that language's techniques and tools to ensure that the game rules are complete and address every possible card interaction with less painstaking and time-consuming play-testing.

There are many ways to create formal specifications and perform formal analysis. Each paradigm abstracts away different parts of the described system and has distinct sets of tools to aid in generating specifications and proofs. Different paradigms allow users of the model to better focus on different facets of the system; there is no "one size fits all" solution. Decisions about which one to use are influenced by which system features are crucial to model, what types of analyses must be done, and what the capabilities of the available tools are.

Many paradigms exist, as outlined by Lamsweerde [24]. For instance, history-based specifications focus on the system's behavior over time, making assertions on its past, present, and future states [24]. Functional models describe the system as a collection of functions; they may be more focused on the connections between each function's possible inputs and outputs or on the paths through which data flows through the system. [24]. State-based paradigms describe the system as a state machine, focusing on key state snapshots and how the system can change states [24]. For modeling CCGs, this may be the most intuitive choice. Taking various game actions could be modeled as transitioning the game from one board state to the next.

3 Literature Review

CCGs and formal methods specifically have little history together, so hardly any research exists that links the two. However, many researchers in both fields do share a goal of making game design and development easier; they have just uses different approaches to do so.

3.1 Game Formalization

Most existing game formalization efforts have been concentrated on creating general "video game description languages" (VGDLs) rather than any particular game or genre. All games, including video games, take immense amounts of resources to design, manufacture, and distribute. VGDLs aim to facilitate the design process by helping designers quickly build and iterate on their game ideas—even without extensive programming knowledge—using formal languages that computers can generate bare-bones prototypes from. Many VGDLs have been created in the past decade, each building upon the capabilities of and overcoming the limitations of the previous [25, 26, 27, 28, 29, 30]. XVGDL, one of the most recent, is notable for being the most powerful of its kind yet [30]. It is based on XML, a well-known and commonly-used markup language with lots of existing tools, making it quite easy to start learning and using. After describing how it works, creators Quiñones and Fernández-Leiva then demonstrate its capabilities using the interpreter they also built that can read XVGDL specifications and output basic ASCII graphics implementations of the described game. XVGDL has already found use in research and appears to be an extremely influential technology for the intersection of video game design and software engineering.

There is very little literature about formalization for specific game genres, however. Jamil and their team's specification of infinite runner games is a notable exception. They use Z-notation to describe the core features of these games and how they change during gameplay [31]. Like Quiñones' and Fernández-Leiva's XVGDL, their work is also meant to facilitate the game development process through faster iteration and implementation. My formalization of *Hearthstone* is intended to do the same for CCGs, but with more emphasis on the game's design. While it can guide the implementation of CCG prototypes, it is better suited to helping designers understand their CCG's cards and mechanics more deeply.

3.2 Machine Learning, Artificial Intelligence, and CCGs

Instead of formalization, the vast majority of CCG-related literature involves machine learning (ML) techniques and artificial intelligence (AI). These approaches are centered around teaching a computer how to play and improve at various aspects of CCGs from deck-building to meta-game strategies like predicting opponents' decks. While the research reviewed here was conducted using digital CCGs, the concepts could be applied to most other CCGs due to their core similarities.

In the deck-building space, researchers have been exploring how to use AI to build a wide variety of decks automatically and improve the AI's deck-building skills. Bhatt et al. confirmed many nuggets of common knowledge in CCG communities, including the facts that strategies with different goals favor different cards and that some decks (even if they share an overall strategy) are inherently better than others [32]. The fact that AIs can also reach the same conclusions is promising for using them to replicate the ways player communities find the best decks at any given moment. Fontaine's and Zhang's teams showcased several quality diversity algorithms and their ability to create "high-performing decks in a variety of strategy spaces" [33, 34]. Altogether, automatic deck builders can greatly facilitate CCG iteration, trying every card combination imaginable and thus helping designers find flaws and ambiguities faster.

A related problem is that of "drafting". In this style of playing CCGs, instead of deck-building with a known set of available cards, players build improvised decks, picking (drafting) cards one by one from a series of random options. This problem is characterized by its "vast state space and omnipresent non-determinism", which drastically changes the criteria for what makes a card strong [35]. Kowalski's team uses an active genes approach while Vieira's uses reinforcement learning, and both achieve some success. Vieira et al. note how having the correct strategy for playing with the drafted deck is also key to a good win rate, while Kowalski et al. intend to further improve how reliably their approach does just that [35, 36, 37]. Understanding these deck-building scenarios can help CCG designers craft better draft modes and new player experiences as both involve less synergistic decks built using fragmented card pools.

Lastly, other authors have been using ML and AI techniques to explore broader meta-characteristics of CCGs. Experienced players can often recognize what type of deck they are playing against just by noticing how the opponent plays their first few turns. Eger and Chacón try using a recurrent neural network to replicate this process and find great success, with their model performing 20% better than the baseline [38]. Knowing the opponent's deck and goal is helpful for deducing the best way to approach a match, and incorporating the opponent's strategy information can improve AI players as well. With a large enough data set, these techniques can also help CCG designers monitor their game's "balance". For example, if every deck starts identically while their strategies are very disparate, it may indicate that early-game card pool is imbalanced with those common cards greatly overpowering the rest. In a similar vein, Silva et al. focus on game balance but on a deck level, creating an AI that can successfully keep many varied decks in general parity with each other [39]. This is particularly helpful for CCG designers and balancers as metagames are notoriously volatile, and small changes have potentially massive ripple effects; designers can use these techniques to test out various changes before implementing them.

Like the aforementioned researchers, I intend to further facilitate the CCG designer's job through the creation of an extensible formalization that is compatible with many useful tools. By maintaining a strong specification and making regular use of type checkers, theorem provers, and solvers, designers can verify various properties of their ideas even before implementing a prototype. This all helps with catching issues with card design or game rules earlier, saving time, money, and stress.

4 Modeling *Hearthstone*

I chose Blizzard Entertainment's *Hearthstone* as the subject of this analysis due to a personal familiarity with the game and its simplicity compared to other popular CCGs. The description presented here uses the Z specification language for its state-based paradigm and robust community-built tool set, but no prior knowledge of the language is assumed here. Additionally, it focuses on the game's combat mechanics and thus omits anything that is not directly related to that, including deck-building restrictions, non-minion cards, mana, and minion effects.

The Z language draws heavily on constructs and notation from set theory and first-order logic. Like many other specification languages, it is *not* an executable programming language. Z specifications describe a system's state and how that changes, but there can be many possible code implementations of a particular specification. This language is not a tool for implementing systems, but rather one for describing systems and ensuring their quality.

In Z, specifications consist of "paragraphs" with optional "data" and "constraint" sections that are separated by a horizontal line if both are included. The data section describes state variables and their types, the (mathematical) set of values that they could take on. Small "function" signatures that link different types of data together can also be defined here. The constraint section contains first-order logic predicates that the state variables and functions must always satisfy.

Z paragraphs fall into one of three categories based on what they define: types, axiomatic data, and schemas. The first defines new types and their associated names; complex types can be defined using existing types and predicates in set notation. Axiomatic paragraphs, marked by a bar on their left side, typically denote globally scoped constants, variables, and functions. Schemas, enclosed in a box missing its rightmost edge, make up the core of all Z specifications and name a series of variables and predicates that represent some state or state transition.

Schemas are a particularly powerful and versatile construct. Z specifications outline a state machine using schemas to describe the system's possible states and transitions between them. However, schemas also represent the set of possible states the object they describe could take, and can thus be used as a type. They can even be used in expressions to represent the constrained variables within. In these ways, schemas are reminiscent of structs in executable programming languages and can be used as macros for the state they contain.

For this model, state schemas will be defined first, followed by transition schemas. These state transitions map cleanly onto the game actions players can take on their turn: summoning minions, declaring attacks, and ending their turn. A *Hearthstone* match can be conceptualized as a series of these transitions, continuing until one player wins.

4.1 Cards and Minions

CCGs unsurprisingly revolve around the "cards" that players collect, build decks, and play with. Their properties while not in play are not important here, so they can be represented by a "basic type" in Z, which simply declares that a set of cards exists.

$$[CARD]$$

Summoning minions to gradually reduce their opponent's health to 0 is the main way players win *Hearthstone* matches. Minion cards are a special type of card that become persistent entities on the board with "attack" and "health" values that describe their combat prowess. Their attack stat describes how much damage they deal while health indicates how much they can take before dying. A minion can never have less than 0 attack, but its health can go below 0 if it is killed by a stronger opponent, hence their differing representations despite both being just numbers. It makes little sense for a minion to immediately die after it is summoned, so their health must start above 0. This specified by *InitMinion*, which can be read as "a minion is properly initialized if its health is greater than 0".

Minion

$$attack : \mathbb{N}$$
$$health : \mathbb{Z}$$

InitMinion

$$Minion$$

$$health > 0$$

4.2 Player State

Each player in every *Hearthstone* match has their own hand, deck, board, and health stat. The hand is a set of cards in some order. This order does not matter, so a player's hand can be represented as a simple subset of $CARD$. Decks are also sets of cards, but the order does matter here as drawing cards always takes from the top of the deck. A sequence of cards—a set of ordered pairs mapping position numbers to the cards at those positions—will therefore be used to represent a deck. The "top" card will be at position 1.

Hand

$$hand : \mathbb{P}\, CARD$$

Deck

$$deck : \text{seq}\, CARD$$

The board is the set of summoned minions that are still alive (have more than 0 health). Each player will have their own board to easily indicate which minions they do and do not control. The order of minions on a board does not matter, so it will simply be some subset of all possible living minions. Players can only have 7 minions on their board, but this limit is arbitrary barring visual clarity constraints, so no concrete limit will be specified here.

$$max\_board\_size : \mathbb{N}$$

Board

$$on\_board : \mathbb{P}\, Minion$$

$$\#\, on\_board \leq max\_board\_size$$
$$\forall\, m : Minion \bullet$$
$$\quad m \in on\_board \Rightarrow m.health > 0$$

The previous three schemas can be combined with a health stat to fully describe a player's state. Just like minions, players can go below 0 health if attacked by a strong enough minion, so their representations will look alike.

PlayerState

$$health : \mathbb{Z}$$
$$Hand$$
$$Deck$$
$$Board$$

To simplify the model, the moment at which players enter a match will be separated from when it starts. This distinction can be thought of as separating the process of setting up the play area from the actual start of the game. When setting up a match, all of a player's cards should begin in their deck, not in their hand or on their board. That deck should also not start empty; combined with the previous stipulation, this ensures that each player actually has cards to play the match with. A player's health should also start above 0 as it would not make sense for a player to join the match already defeated.

InitPlayerState

$$PlayerState$$

$$health > 0$$
$$hand = \varnothing$$
$$deck \neq \langle\rangle$$
$$board = \varnothing$$

4.3 Turns and the Game State

In *Hearthstone*, there are exactly 2 players, and players cannot act outside of their turn. Tracking whose turn it currently is can therefore be done with a simple single-variable schema storing a player label.

$$PLAYER ::= P1 \mid P2$$

TurnState

$$curr\_player : PLAYER$$

For simplicity, $P1$ will always represent the player going first. *InitTurnState* will start the turn cycle, and *ChangeTurnPlayer* will handle the turn swapping. An axiomatic *opponent* function that maps each player to their opponent will be helpful for defining the latter.

InitTurnState

$$TurnPlayer$$

$$curr\_player = P1$$

$$opponent : PLAYER \to PLAYER$$

$$\begin{aligned} opponent = \{ & \\ & P1 \mapsto P2, \\ & P2 \mapsto P1 \\ \} & \end{aligned}$$

ChangeTurnPlayer

$$\Delta TurnState$$

$$curr\_player' = opponent(curr\_player)$$

The overall game state is comprised of each player's individual state and the *TurnState*. However, there is currently no way to link together a player and their state. A function within the *GameState* schema can handle that mapping. The inner workings of this *pl_state* function are not given here because they are unnecessary; the important part—the fact that it maps a *PLAYER* to some instance of *PlayerState*—is already clear in its signature. While there are no explicit variables here that store any particular player's state, a combination of *curr_player* as well as the *opponent* and *pl_state* functions can retrieve any player's state in any scenario. Alongside the player states and *TurnState*, it will also be helpful to keep the set of minions that have attacked each turn here. Each minion can only attack once per turn, and tracking that here prevents the additional complexity of doing so individually.

GameState

$$
\begin{array}{l}
TurnState \\
battled\_this\_turn : \mathbb{P}\, Minion \\
pl\_state : PLAYER \rightarrow PlayerState
\end{array}
$$

InitGameState

$$
\begin{array}{l}
GameState \\
InitTurnState \\
\hline
battled\_this\_turn = \varnothing \\
pl\_state(P1) \in InitPlayerState \\
pl\_state(P2) \in InitPlayerState
\end{array}
$$

Placing the *pl_state* function inside the *GameState* schema notably contrasts with the axiomatic *opponent* function, which exists outside of any schema and always maps each player to the same other player. A player's state will naturally change throughout a match, which is recognized in Z as different instances of the *PlayerState* schema—different elements of the set of possible player states. If *pl_state* were axiomatic, that would imply that each *PLAYER* always maps to the *exact same* element of *PlayerState* and thus prevent its natural fluctuation. Placing *pl_state* inside of *GameState* instead signals that its output may change depending on the rest of the *GameState*; different instances of *GameState* will map each player to a different *PlayerState*.

4.4 Starting a Match

With the game state set up, the match can begin. To start, both players draw their starting hand. There are several scenarios enabled by the model that must be accounted for when specifying starting hands, so in typical Z fashion, one smaller schema for each situation will be made separately. These individual schemas will then be combined into one composite schema that describes the action as a whole.

State transitions like these are signified by the presence of delta (Δ) and xi (Ξ) schemas references. Delta indicates that the marked schema will change as a result of this transition; xi indicates that the marked schema will not, but its variables may still be referenced. Both introduce primed (') and unprimed variables, which denote the "after" and "before" states of the marked variable or schema, respectively.

In the basic case, both players draw the same size starting hand, but player 2 may draw one more than player 1 to mitigate the inherent disadvantage of going second. However, like the maximum size limit for the board, this is arbitrary and subject to the designer's desired game balance or play experience.

$$
hand\_init\_size : \mathbb{N}
$$

Because the hand is a power set of $CARD$ while the deck is a sequence, this specification must take some extra steps to convert between the two slightly different types. An axiomatic *draw* function can help with this, especially when drawing multiple cards at once. It will take the number of cards to draw and the *Deck* (schema type) variable to draw from, then return the set of cards drawn.

This conversion is done using the number range $(..)$ and sequence extraction operator $(\upharpoonright)$. With the number range, one can easily refer to a set of consecutive numbers; for instance, the set $\{1, 2, 3, 4\}$ can be abbreviated as $1..4$. The extraction operator takes any set of numbers indicating positions and a sequence to extract from, then returns a new sequence of the items formerly at those positions. Lastly, the range operator (ran) is needed to decouple the cards from their positions, returning the set of cards drawn, rather than another sequence.

$$
\begin{array}{|l}
draw : (\mathbb{N}_1 \times Deck) \rightarrow \mathbb{P}\, CARD \\
\hline
\forall\, n : \mathbb{N}_1;\ d : Deck \bullet \\
\quad draw(n, d) = \mathrm{ran}((1..n) \upharpoonright d.deck)
\end{array}
$$

One schema will be created for each player to easily delineate the different starting hand sizes. After the cards are drawn, those same cards must also be removed from the deck so that one card is not in multiple places at once. This is done by using set difference and the sequence filter operator $(\upharpoonright)$ together to isolate the set of removed cards from the deck's previous state. To maintain clarity and minimize redundant rewriting, a **let** statement is used to abbreviate the *pl_state* function call.

$$
\begin{array}{|l}
\hline
P1NormalStart \\
\hline
InitGameState \\
\Delta GameState \\
\Xi TurnState \\
\hline
\textbf{let}\ p1 == pl\_state(P1) \bullet \\
\quad \#\, p1.deck > hand\_init\_size \\
\quad p1.hand' = draw(hand\_init\_size,\ p1.Deck) \\
\quad p1.deck' = p1.deck \setminus (p1.deck \upharpoonright p1.hand') \\
\quad p1.board' = p1.board \\
\quad p1.health' = p1.health \\
battled\_this\_turn' = battled\_this\_turn \\
\hline
\end{array}
$$

P2NormalStart

$InitGameState$
$\Delta GameState$
$\Xi TurnState$

---

let $p2 == pl\_state(P2)$ •
$\#\, p2.deck > hand\_init\_size$

$p2.hand' = draw(hand\_init\_size + 1,\ p2.Deck)$
$p2.deck' = p2.deck \setminus (p2.deck \upharpoonright p2.hand')$

$p2.board' = p2.board$
$p2.health' = p2.health$

$battled\_this\_turn' = battled\_this\_turn$

This case assumes that the deck is larger than the starting hand size, as indicated by the first predicate which compares the deck's cardinality (#) to the starting hand size. Since no concrete minimum or maximum values were specified for the hand and deck sizes, this actually may not always be true. While this is not a typical concern, it still must be accounted for to create a complete specification here. The player will simply draw their entire deck in this scenario. Since this could happen to either player independently, separate schemas will be created for each.

P1DrawDeck

$InitGameState$
$\Delta GameState$
$\Xi TurnState$

---

let $p1 == pl\_state(P1)$ •
$\#\, p1.deck \leq hand\_start\_size$

$p1.hand' = ran(p1.deck)$
$p1.deck' = \varnothing$
$p1.board' = p1.board$
$p1.health' = p1.health$

$battled\_this\_turn' = battled\_this\_turn$

$$
\begin{array}{|l}
\hline
P2DrawDeck \\
\hline
InitGameState \\
\Delta GameState \\
\Xi TurnState \\
\hline
\textbf{let}\ p2 == pl\_state(P2) \bullet \\
\quad \#\, p2.deck \leq hand\_start\_size \\
\quad p2.hand' = ran(p2.deck) \\
\quad p2.deck' = \varnothing \\
\quad p2.board' = p2.board \\
\quad p2.health' = p2.health \\
battled\_this\_turn' = battled\_this\_turn \\
\hline
\end{array}
$$

Altogether, the entire process of drawing starting hands can be described as a combination of the previous four schemas. Both players draw their starting hand independently, which could be the entire deck or a subset of it depending on how big their deck is and how big starting hands are defined to be. The full process is defined as the result of both occurring in some appropriate combination.

$$P1Start == P1NormalStart \lor P1DrawDeck$$
$$P2Start == P2NormalStart \lor P2DrawDeck$$

$$DrawStartingHands == P1Start \land P2Start$$

While match setup was separated from the start of the match to simplify model design, the actual game makes no distinction between these phases. They always occur immediately in sequence as no gameplay can truly begin until starting hands have been drawn. To match this, the setup and start schemas will be joined sequentially and atomically with the schema composition operator ($\fatsemi$).

$$FullMatchStart == InitGameState \fatsemi DrawStartingHands$$

4.5 Passing Turns in Full

At the start of each turn, the turn player draws 1 card from their deck (if they still have a deck to draw from). This draw helps ensure that they continue to have options to play with as the match progresses and they use up more cards from their hand. Now that the player likely already has other cards in hand, the newly drawn card must be added to the existing set with a set union. Also, because only one card is being drawn, the simpler operators *head* and *tail* can be used instead of the more complex extraction and filtration. The *head* operator returns the sequence's first element, while *tail* returns all elements *but* the first.

TurnDraw

$\Delta GameState$
$\Xi TurnState$

---

let $p == pl\_state(curr\_player)$ •
$p.deck \neq \langle\rangle$

$p.hand' = p.hand \cup head(p.deck)$
$p.deck' = tail(p.deck)$

$p.board' = p.board$
$p.health' = p.health$

$pl\_state(opponent(curr\_player))' = pl\_state(opponent(curr\_player))$

$battled\_this\_turn' = \varnothing$

If the turn player runs out of cards to draw, they will take "fatigue" damage. In *Hearthstone* proper, this damage increases every turn to help ensure that every match actually ends, but in this model the player will just take 1 damage each turn for simplicity.

TurnFatigue

$\Delta GameState$
$\Xi TurnState$

---

let $p == pl\_state(curr\_player)$ •
$p.deck = \langle\rangle$

$p.health' = p.health - 1$

$p.hand' = p.hand$
$p.deck' = p.deck$
$p.board' = p.board$

$pl\_state(opponent(curr\_player))' = pl\_state(opponent(curr\_player))$

$battled\_this\_turn' = \varnothing$

At the start of a new turn, the set of minions that battled must also be reset, allowing the new turn player's minions to attack again. This is a simple operation, so it was simply included at the end of both the previous schemas.

Ending one's turn has no inherent procedures besides the turn player simply declaring that they are finished. Without nothing else to specify here, the model only needs to swap who the turn player is, which was defined earlier with the specification of turns and players. Having defined turn start and end procedures, the model for passing turns can be fully defined as well. This will also be done as a sequential yet atomic procedure using the ⨾ operator.

$$StartTurn == TurnDraw \lor TurnFatigue$$

$$PassTurn == ChangeTurnPlayer \fatsemi StartTurn$$

4.6 Summoning Minions

To get minions onto the board and able to reduce their opponent's health, players must summon them from their hand. Since everything in the hand is a card, there must be a way to transform them into minions, which is exactly what the *to_minion* function will do. The function will also ensure that each minion is properly initialized when it is summoned.

$$to\_minion : CARD \to Minion$$

$$\forall m : Minion \bullet m \in \mathrm{ran}(to\_minion) \Rightarrow m \in InitMinion$$

Naturally, players can only summon a minion if its associated card is currently in their hand, and they can only do so if they have enough room on board to accommodate the new entity. If both of those preconditions are met, then the chosen card is removed from the player's hand and transformed into a minion on the board.

SummonSuccess

$$\Delta GameState$$
$$\Xi TurnState$$
$$minion? : CARD$$

$$\textbf{let } p == pl\_state(curr\_player) \bullet$$
$$minion? \in p.hand$$
$$\# p.board < max\_board\_size$$
$$p.hand' = p.hand \setminus \{minion?\}$$
$$p.board' = p.board \cup \{to\_minion(minion?)\}$$
$$p.deck' = p.deck$$
$$p.health' = p.health$$
$$pl\_state(opponent(curr\_player))' = pl\_state(opponent(curr\_player))$$
$$battled\_this\_turn' = battled\_this\_turn$$

If either condition is not met, then the *GameState* (including the hand and board state) will remain unchanged as the summon would be impossible.

$$SummonMinion == SummonSuccess \lor \Xi GameState$$

4.7 Combat Against Players

Much like *SummonMinion*, several preconditions must be met for the declared attack to be valid: the turn player must choose a minion on their board to attack with, and that minion must not have attacked yet this turn. If all those preconditions are met, then the declared battle can occur. The minion will then deal damage the opposing player's health equal to its attack stat. All other aspects of the game state are unaffected by combat.

MinionAttackPlayerSuccess

$$
\begin{array}{l}
\Delta GameState \\
\Xi TurnState \\
attacker? : Minion \\
\hline
\textbf{let}\ opp = pl\_state(opponent(curr\_player)) \bullet \\
\quad attacker? \in pl\_state(curr\_player).board \\
\quad attacker? \notin battled\_this\_turn \\
\quad opp.health' = opp.health - attacker?.attack \\
\quad battled\_this\_turn' = battled\_this\_turn \cup \{attacker?\} \\
\quad opp.hand' = opp.hand \\
\quad opp.deck' = opp.deck \\
\quad opp.board' = opp.board \\
pl\_state(curr\_player)' = pl\_state(curr\_player)
\end{array}
$$

Also similar to summoning a minion, if either condition is not met, then nothing happens as it is an invalid attack declaration. A full specification of attacking a player will therefore appear very similar to *SummonMinion*:

$$MinionAttackPlayer == MinionAttackPlayerSuccess \lor \Xi GameState$$

4.8 Combat Between Minions

Combat between minions builds on this, but now a more complex damage calculation and the potential for minions to die from combat must be accounted for. Like with starting hands, these components will be described in separate schemas, then combined into a larger schema that fully describes the battle. When two minions battle, both take damage equal to the other's attack stat.

MinionCombatDamage

$$
\begin{array}{l}
attacker? : \Delta Minion \\
defender? : \Delta Minion \\
\hline
attacker?'.health = attacker?.health - defender?.attack \\
defender?'.health = defender?.health - attacker?.attack \\
\\
attacker?'.attack = attacker?.attack \\
defender?'.attack = defender?.attack
\end{array}
$$

After a battle, the board must be updated to contain the correct minion states. This will be achieved by simply removing the old minion from the board and adding the updated minion state using the *update_minion* function. The function will also check the minion's post-combat stats to ensure that it is still alive after the battle as the board can only contain living minions. If it did die in the battle, then it will simply be removed from the board.

$$
\begin{array}{l}
update\_minion : \Delta Minion \times \mathbb{P}\, Minion \to \mathbb{P}\, Minion \\
\hline
\forall\, m : \Delta Minion;\ b : Board \bullet \\
\quad update\_minion(m, b) = \\
\qquad \textbf{if}\ m.health' > 0 \\
\qquad\quad \textbf{then}\ (b.board \setminus \{m\}) \cup \{m'\} \\
\qquad\quad \textbf{else}\ b.board \setminus \{m\}
\end{array}
$$

UpdateBoard

$$
\begin{array}{l}
\Delta GameState \\
\Xi TurnState \\
attacker? : \Delta Minion \\
defender? : \Delta Minion \\
\hline
\textbf{let}\ p == pl\_state(curr\_player) \bullet \\
\quad p.board' = update\_minion(attacker?, p.Board) \\
\\
\textbf{let}\ opp == pl\_state(opponent(curr\_player)) \bullet \\
\quad opp.board' = update\_minion(defender?, opp.Board)
\end{array}
$$

In addition to the previously defined preconditions, a successful battle between minions also requires that the combatants belong to opposing players. If all these conditions are met, then the combat damage and board states will be updated accordingly.

Finally, to update the *GameState* correctly, the attacker's updated state should be added to the *battled_this_turn* set to mark that it has successfully attacked. The updated state must be added instead of the old state as subsequent attack declarations with that minion will check for that new state in *battled_this_turn*. On the other hand, defending minions can be the target of multiple

attacks and thus can participate in many battles per turn, so they do not need to be added to or checked against the *battled_this_turn* set.

MinionAttackMinionSuccess

$\Delta GameState$
$\Xi TurnState$
$attacker? : \Delta Minion$
$defender? : \Delta Minion$

$\textbf{let}\ p = pl\_state(curr\_player);\quad opp = pl\_state(opponent(curr\_player))\ \bullet$

$attacker? \in p.board$
$attacker? \notin battled\_this\_turn$
$defender? \in opp.board$

$MinionCombatDamage$
$UpdateBoard$

$p.hand' = p.hand$
$p.deck' = p.deck$
$p.health' = p.health$
$opp.hand' = opp.hand$
$opp.deck' = opp.deck$
$opp.health' = opp.health$

$battled\_this\_turn' = battled\_this\_turn \cup \{attacker?'\}$

Just like summoning minions and attacking players, nothing should happen if the declared battle is invalid in any way.

$$MinionAttackMinion == MinionAttackMinionSuccess \lor \Xi GameState$$

4.9 Win and Lose Conditions

Hearthstone keeps its win condition simple: players must reduce their opponent's health to 0 (or below) to win. One schema for each player will describe these win states.

P1 Win

$GameState$

$pl\_state(P1).health > 0$
$pl\_state(P2).health \leq 0$

$P2\,Win$

$GameState$

$pl\_state(P1).health \leq 0$

$pl\_state(P2).health > 0$

If both players are brought to 0 health or below at the same time, the match will be considered a tie. While this situation is highly unlikely, it must be included for a complete model.

Tie

$GameState$

$pl\_state(P1).health \leq 0$

$pl\_state(P2).health \leq 0$

Matches always end in one of these three outcomes.

$$MatchEnd == P1\,Win \lor P2\,Win \lor Tie$$

4.10 The Full Model

Combining all the previous schemas yields a complete model of this extremely simplified version of *Hearthstone*. Every match starts with some setup and players drawing their starting hands. Players can then spend their turns summoning minions, attacking their opponent directly, attacking enemy minions, or passing turn control to their opponent. Players then continue taking turns until one player's health is depleted, ending the match.

$$\begin{aligned} &TurnActions == \\ &\quad SummonMinion \lor \\ &\quad MinionAttackPlayer \lor \\ &\quad MinionAttackMinion \lor \\ &\quad PassTurn \end{aligned}$$

$$\begin{aligned} &SimpleHearthstoneMatch == \\ &\quad FullMatchStart \lor \\ &\quad TurnActions \lor \\ &\quad MatchEnd \end{aligned}$$

5 Evaluation: Formal Model vs. Original Game

Despite its simplicity, this specification already clarifies some properties about the game that are omitted from the tutorial. These include the board size limit and what happens when both players reach 0 health at the same time.

The board size limit is never explicitly mentioned, but astute players may notice the in-game limit is 7 when the tutorial enemy uses The Scourge card (Fig. 5).

Designers must carefully consider this property as different board sizes directly affect the power level of board-wide effects like that of The Scourge. Players must also be mindful of the limit to make the most of the available space. The specification ensures that this property is always tied to the board itself, so any future operations that involve the board must also take the maximum size into account. With the way *SummonMinion* is specified, players also do not need to worry about losing a minion card they accidentally tried to summon while their board was full. The game state remains unchanged, so implying that the card stays in the player's hand.

Figure 5: "The Scourge" [40]

This model also explicitly describes what happens if both players' health stats are depleted at the same time while the official game rules do not. If the individual *Win* cases did not have predicates stating that the winner must stay alive, then a match could have 2 winners or losers at the same time, which could be considered valid states depending on one's interpretation of the situation and how they design the model. Players may not know what to expect when this first happens to them because of that potential ambiguity; they may be disappointed if they are denied their expected post-match rewards such as rank-up points or mission completion progress. An explicit and separate *Tie* case makes it clear that this should be treated differently from the single winner cases. However, what to do with that information is beyond the scope of this specification and left to the designer's discretion.

While these properties can be easily communicated or inferred, stating them explicitly and formally here helps developers ensure that no new features accidentally violate or contradict them. The ways in which this specification highlights or clarifies these properties can also be extrapolated for more complex properties as more systems and cards are added to the specification. Once an ambiguous case is found in the specification, designers can devise a sound solution before proceeding too far with any implementations and update the game rules or affected card text(s) accordingly.

6 Future Work

While complete and self-contained, this specification does not model the vast majority of *Hearthstone*, let alone CCGs in general. The entire realm of deck-building and its restrictions are not specified at all here despite being a core part of the CCG experience. More work is required to expand this specification to include these and *Hearthstone*'s many other card types and systems, including spell cards, hero powers, and the mana system.

Furthermore, the power of Z's tools in this context has yet to be demonstrated. For instance, type checkers can ensure that the model is syntactically consistent and valid. Meanwhile, theorem provers and formal proof techniques can be leveraged to investigate more complex properties about

the game such as whether invalid, conflicting, or overlapping states are possible anywhere in the model. These tools can be used while iterating on the model to ensure that every new card or system works well with all existing ones.

The process of using formal methods in this context can also be simultaneously refined and streamlined to make it more viable for and accessible to those in industry. Ideally, this specification will eventually be generalized to, and usable for, all CCGs. This would enable it to serve as a template for creators to use that allows them to focus on what makes their game unique, instead of the boilerplate shared characteristics of the genre.

7 Conclusion

Collectible card games have been a popular genre for decades, and more new titles are still being released now both physically and digitally. As fans and designers continue to develop new CCGs or expand the massive card pools of long-standing ones, the need for robust tools to help manage the extreme complexity also continues to grow. Formal methods are one potential solution, and this paper serves as a proof of concept of the power that this set of tools offers.

By describing a CCG's rules and cards in a formal language, one gains access to tools that can help ensure that the game has no contradictory or ambiguous situations. If any are found, proofs and theorem checkers can help narrow down where the conflict is and facilitate the development of a clearer design. Altogether, formal methods can help CCG designers manage their game's complexity and keep the game as straightforward as possible, allowing them to focus on their development work and players to focus on enjoying all that the game has to offer.

Acknowledgements

Thank you to my family for their support throughout my academic endeavors. Thank you to my friends, new and old, for their encouragement and support as well; I greatly appreciate how willing they were to read this work several times over as I wrote and revised it bit by bit. Their assistance, aid, and companionship have been instrumental throughout this project and my undergraduate experience overall.

Thank you to the Portland State University Honors College for offering opportunities to conduct and publish exciting research, especially since this is now my second published work [41]. Last but certainly not least, thank you to Bart for being a fantastic advisor. He introduced me to this subfield of computer science and offered great patience, insight, and feedback throughout this process even as deliverables were occasionally delayed.

References

[1] Magic Madhouse, "How many cards are there in Magic: The Gathering?," February 2024. [Online]. Available: https://magicmadhouse.co.uk/how-many-cards-are-there-in-magic-the-gathering. [Accessed 2024-05-23].

[2] Scryfall, LLC, "Scryfall Magic The Gathering Search," 2024. [Online]. Available: https://scryfall.com/search?q=cmc%3E%3D0+is%3Afirstprint&as=grid&order=name. [Accessed: 2024-05-23].

[3] Blizzard Entertainment, Inc, "Wild Cards - Hearthstone Card Library," 2024. [Online]. Available: https://hearthstone.blizzard.com/en-us/cards?set=wild&viewMode=table&collectible=1. [Accessed: 2024-05-23].

[4] Hearthstone Wiki, "Card — Hearthstone Wiki," April 2024. [Online]. Available: https://hearthstone.wiki.gg/index.php?title=Card&oldid=795812. [Accessed 2024-05-23].

[5] Hearthpwn, "Fireball - Cards - Hearthstone," 2024. [Online]. Available: https://www.hearthpwn.com/cards/522-fireball. [Accessed: 2024-05-23].

[6] Yugipedia, "Astrograph Sorcerer - Yugipedia," February 2024. [Online]. Available: https://yugipedia.com/index.php?title=Astrograph_Sorcerer&oldid=5013362. [Accessed: 2024-05-23].

[7] Play MTG, "Jean-Emmanuel Depraz vs. Kazune Kosaka — Finals — Magic World Championship XXIX," September 2023. [Online]. Available: https://www.youtube.com/watch?v=XlgUO8ET618. [Accessed: 2024-05-23].

[8] MajiinBae, "This is the Swain deck you've been waiting for...," November 2023. [Online]. Available: https://www.youtube.com/watch?v=mn_CP9CYx7A, [Accessed: 2024-06-02].

[9] Steve Jackson Games, "Illuminati: The Game of Conspiracy," 2020. [Online]. Available: https://www.sjgames.com/illuminati/. [Accessed: 2024-05-23].

[10] Wizards of the Coast LLC, "Magic: The Gathering — Official site for MTG news, sets, and events," 2024. [Online]. Available: https://magic.wizards.com/en. [Accessed: 2024-06-02].

[11] Legend Story Studios, "Flesh and Blood TCG," 2024. [Online]. Available: https://fabtcg.com/. [Accessed: 2024-06-02].

[12] Bandai Namco, "DRAGON BALL SUPER CARD GAME," 2024. [Online]. Available: https://www.dbs-cardgame.com/us-en/. [Accessed: 2024-06-02].

[13] Bandai Namco, "Digimon Card Game," 2024. [Online]. Available: https://world.digimoncard.com/. [Accessed: 2024-06-02].

[14] RuneterraFire, "Lonely Poro :: Legends of Runeterra Card :: RuneterraFire," 2019. [Online]. Available: https://www.runeterrafire.com/cards/lonely-poro. [Accessed: 2024-05-27].

[15] Blizzard Entertainment Inc., "Hearthstone," 2024. [Online]. Available: https://hearthstone.blizzard.com/en-us. [Accessed: 2024-06-02].

[16] CD Projekt, "GWENT: The Witcher Card Game," 2024. [Online]. Available: https://www.playgwent.com/en. [Accessed: 2024-06-02].

[17] Riot Games Inc., "Legends of Runeterra," 2024. [Online]. Available: https://playruneterra.com/en-us/. [Accessed: 2024-06-02].

[18] Konami, "Yu-Gi-Oh! TRADING CARD GAME - Official Website," 2024. [Online]. Available: https://www.yugioh-card.com/en/. [Accessed: 2024-06-02].

[19] Pokémon, "Pokémon Trading Card Game — Pokemon.com," 2024. [Online]. Available: https://www.pokemon.com/us/pokemon-tcg. [Accessed: 2024-06-02].

[20] Hearthpwn, "Acolyte of Pain - Cards - Hearthstone," 2024. [Online]. Available: https://www.hearthpwn.com/cards/1024953-acolyte-of-pain. [Accessed: 2024-05-23].

[21] Hearthpwn, "Shattered Sun Cleric - Cards - Hearthstone," 2024. [Online]. Available: https://www.hearthpwn.com/cards/434-shattered-sun-cleric. [Accessed: 2024-05-23].

[22] Hearthpwn, "Ironbeak Owl - Cards - Hearthstone," 2024. [Online]. Available: https://www.hearthpwn.com/cards/475169-ironbeak-owl. [Accessed: 2024-05-23].

[23] R. M. Hierons, K. Bogdanov, J. P. Bowen, R. Cleaveland, J. Derrick, J. Dick, M. Gheorghe, M. Harman, K. Kapoor, P. Krause, G. Lüttgen, A. J. H. Simons, S. Vilkomir, M. R. Woodward, and H. Zedan, "Using formal specifications to support testing," *ACM Comput. Surv.*, vol. 41, February 2009. Available: https://dl.acm.org/doi/10.1145/1459352.1459354.

[24] A. v. Lamsweerde, "Formal specification: A roadmap," in *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, (New York, NY, USA), p. 147–159,

Association for Computing Machinery, 2000. Available: https://dl.acm.org/doi/10.1145/336512.336546.

[25] C. Browne and F. Maire, "Evolutionary game design," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 1, pp. 1–16, 2010. Available: https://ieeexplore.ieee.org/document/5404867.

[26] A. M. Smith, M. J. Nelson, and M. Mateas, "Ludocore: A logical game engine for modeling videogames," in *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, pp. 91–98, 2010. Available: https://ieeexplore.ieee.org/document/5593368.

[27] M. Ebner, J. Levine, S. M. Lucas, T. Schaul, T. Thompson, and J. Togelius, "Towards a Video Game Description Language," in *Artificial and Computational Intelligence in Games* (S. M. Lucas, M. Mateas, M. Preuss, P. Spronck, and J. Togelius, eds.), vol. 6 of *Dagstuhl Follow-Ups*, pp. 85–100, Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013. Available: https://doi.org/10.4230/DFU.Vol6.12191.85.

[28] T. Schaul, "An extensible description language for video games," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 4, pp. 325–331, 2014. Available: https://ieeexplore.ieee.org/document/6884801.

[29] M. Abbadi, F. Di Giacomo, A. Cortesi, P. Spronck, G. Costantini, and G. Maggiore, "Casanova: A simple, high-performance language for game development," in *Serious Games* (S. Göbel, M. Ma, J. Baalsrud Hauge, M. F. Oliveira, J. Wiemeyer, and V. Wendel, eds.), (Cham), pp. 123–134, Springer International Publishing, 2015. Available: https://link.springer.com/chapter/10.1007/978-3-319-19126-3_11.

[30] J. R. Quiñones and A. J. Fernández-Leiva, "Xml-based video game description language," *IEEE Access*, vol. 8, pp. 4679–4692, 2020. Available: https://ieeexplore.ieee.org/document/8945249.

[31] A. Jamil, Z. Murtza, M. K. Nazir, M. Waseem, Z. Ghulam, and R. U. Farooq, "A generic formal specification of an infinite runner games for handheld devices using z-notation," in *2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS)*, pp. 409–413, 2019. Available: https://ieeexplore.ieee.org/document/8821750.

[32] A. Bhatt, S. Lee, F. de Mesentier Silva, C. W. Watson, J. Togelius, and A. K. Hoover, "Exploring the hearthstone deck space," in *Proceedings of the 13th International Conference on the Foundations of Digital Games*, FDG '18, (New York, NY, USA), Association for Computing Machinery, 2018. Available: https://dl.acm.org/doi/10.1145/3235765.3235791.

[33] M. C. Fontaine, S. Lee, L. B. Soros, F. De Mesentier Silva, J. Togelius, and A. K. Hoover, "Mapping hearthstone deck spaces through map-elites with sliding boundaries," in *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '19, (New

York, NY, USA), p. 161–169, Association for Computing Machinery, 2019. Available: https://dl.acm.org/doi/10.1145/3321707.3321794.

[34] Y. Zhang, M. C. Fontaine, A. K. Hoover, and S. Nikolaidis, "Deep surrogate assisted map-elites for automated hearthstone deckbuilding," in *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '22, (New York, NY, USA), p. 158–167, Association for Computing Machinery, 2022. Available: https://dl.acm.org/doi/10.1145/3512290.3528718.

[35] J. Kowalski and R. Miernik, "Evolutionary approach to collectible arena deckbuilding using active card game genes," in *2020 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8, 2020. Available: https://ieeexplore.ieee.org/document/9185755.

[36] R. Vieira, L. Chaimowicz, and A. R. Tavares, "Reinforcement learning in collectible card games: Preliminary results on legends of code and magic," in *Proceedings of the 18th Brazilian Symposium on Computer Games and Digital Entertainment, SBGames*, pp. 611–614, 2019. Available: https://www.sbgames.org/sbgames2019/files/papers/ComputacaoShort/198299.pdf.

[37] R. Vieira, A. R. Tavares, and L. Chaimowicz, "Drafting in collectible card games via reinforcement learning," in *2020 19th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, pp. 54–61, 2020. Available: https://ieeexplore.ieee.org/document/9291616.

[38] M. Eger and P. Sauma Chacón, "Deck archetype prediction in hearthstone," in *Proceedings of the 15th International Conference on the Foundations of Digital Games*, FDG '20, (New York, NY, USA), Association for Computing Machinery, 2020. Available: https://dl.acm.org/doi/10.1145/3402942.3402959.

[39] F. d. Mesentier Silva, R. Canaan, S. Lee, M. C. Fontaine, J. Togelius, and A. K. Hoover, "Evolving the hearthstone meta," in *2019 IEEE Conference on Games (CoG)*, p. 1–8, IEEE Press, 2019. Available: https://ieeexplore.ieee.org/document/8847966.

[40] Hearthpwn, "The Scourge - Cards - Hearthstone," 2024. [Online]. Available: https://www.hearthpwn.com/cards/1994648-the-scourge [Accessed: 2024-05-25].

[41] D. Ha, "The Experiences that Most Affected the Political Socialization of US Undergraduates," *Anthós*, vol. 11, June 2022. Available: https://archives.pdx.edu/ds/psu/37831.