

Developing New Track for Strategy Card Game AI Competition: Constructed Mode

(Opracowanie nowego trybu do zawodów
Strategy Card Game AI: Constructed Mode)

Krzysztof Bednarek

Praca magisterska

Promotor: dr. Jakub Kowalski

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

March 7, 2024

Abstract

Collectible Card Games present intriguing opportunities for Artificial Intelligence research, but their human versions pose challenges as platforms for researchers due to their complexity. Consequently, the initial iteration of Legends of Code and Magic was developed. In this paper, I aim to present the modifications implemented in version 1.5 to enhance the environment for research purposes.

Kolekcyjne Gry Karciane są interesującą bazą dla badań nad Sztuczną Inteligencją. Niestety, ich wersje stworzone dla ludzi są niewygodne w użyciu w celach badawczych. Pierwsza wersja Legends of Code and Magic została stworzona w celu wypełnienia tej luki. W poniższej pracy opiszę zmiany wprowadzone w wersji 1.5 w celu lepszego dostosowania jej jako środowiska do badań.

Contents

1	Introduction	7
1.1	Collectible Card Game	7
1.2	Popular CCGs	7
1.3	Why it is interesting for Artificial Intelligence research	9
1.4	CodinGame	10
1.4.1	Framework	11
1.4.2	Technical limitation	11
2	Legends of Code and Magic	13
2.1	History	13
2.2	Research	14
2.3	Rules	15
2.3.1	Cards	16
2.3.2	Mana	18
2.3.3	Runes	18
2.3.4	Draft Phase	19
2.3.5	Battle Phase	20
2.4	Communication	20
3	Changes	21
3.1	Problems	21
3.2	Constructed Mode	22
3.3	Area	23

3.4	Runes	26
4	Generation	27
4.1	Card Generation	27
4.2	Example of Generation	28
4.3	Generation Configuration	31
4.4	Card Images	31
5	Summary	35
	Bibliography	37

Chapter 1

Introduction

1.1 Collectible Card Game

A Collectible Card Game (CCG) is a genre of games built around the idea of collecting cards and using them to battle other players. Most of them revolve around three basic ideas.

First, players acquire packs of cards, typically containing 5–16 cards per pack, to build up their card collection..

Second, players can use cards from their collection to build up decks. The size of such a deck has to be between set boundaries. For example, the Hearthstone deck has to comprise exactly 30 cards, while in Magic: The Gathering in Standard, you are required to have at least 60 cards in your deck. And while in the online version deck size is limited to 250 cards, in the tabletop the only limit is the player’s ability to shuffle their deck without assistance.

And last but not least, players can use those decks to battle against each other. Typically, players take turns using cards from their decks to lower the opponent’s Health Points with the final goal of reducing the opponent’s health to 0.

1.2 Popular CCGs

There are several popular CCGs. First, what is considered by many the first real CCG is Magic: The Gathering. It was initially released in 1993 by Wizards of the Coast as a tabletop game, and later in digital — first as Magic: The Gathering Online in 2002 and second as Magic: The Gathering Arena in 2019. Originally, it was influenced by Dungeon and Dragons, but now it developed its own world and lore. In 2018, it hit 35 million players around the world in 70 different countries. And only between 2008 and 2016, it printed over 20 billion cards.

Next is Hearthstone, developed by Blizzard Entertainment and published in 2014 as a digital-only game. All graphics and characters are based on their earlier game, World of Warcraft. In the year 2020 alone, it boasted an active player base exceeding 23 million individuals.



Figure 1.1: Hearthstone statistics for 2020 Source: <https://hearthstone.blizzard.com/en-us/news/23625669/year-of-the-phoenix-in-review>

Another popular CCG is the tabletop game Yu-Gi-Oh! Trading Card Game, published in 1999 by Konami. It is based on the game Duel Monsters presented in the manga Yu-Gi-Oh! created by Kazuki Takahashi. Moreover, there are over 50 video games based on the same Duel Monsters game, both multi- and single-player. Currently, the most popular online one is Yu-Gi-Oh! Master Duel. In 2021

1.3. WHY IT IS INTERESTING FOR ARTIFICIAL INTELLIGENCE RESEARCH⁹

Yu-Gi-Oh! Trading Card Game reached 35 billion cards sold.

CCGs are popular not only among players but among the viewers of streams as well. As we can see in the graphic below, viewers together watched over 6 million hours of streams over the span of a single week.











#	Name	Viewer Hours	Hours Streamed	ACV	Creators	Streams
1	 Hearthstone	4,397,756	33,090	26,177	5,608	14,445
2	 Yu-Gi-Oh! Master Duel	1,238,072	17,286	7,369	4,267	8,671
3	 Magic: The Gathering	682,689	12,546	4,063	2,280	5,981
4	 Pokémon Trading Card Game	280,641	6,481	1,670	1,285	2,665
5	 Gwent: The Witcher Card Game	236,783	1,954	1,409	301	818
6	 Slay the Spire	188,353	4,144	1,121	1,112	2,199
7	 Legends of Runeterra	162,491	3,593	967	785	1,832
8	 Pokémon Trading Card Game Online	23,995	1,191	142	337	721
9	 Yu-Gi-Oh! Forbidden Memories	21,841	1,012	130	121	307
10	 Kingdom Hearts Re:Chain of Memories	3,922	228	23	38	70

Figure 1.2: Statistics of streams about most popular CCGs during single week Source: <https://gamesight.io/leaderboards/collectible-card-games>, state for 09.12.2022

1.3 Why it is interesting for Artificial Intelligence research

A game of the CCG genre is interesting for several reasons.

First, it is highly strategic — a player needs to plan what they will do and use their cards reasonably. For instance, advisable to strategically reserve some “removals” (cards intended to eliminate the opponent’s *Creatures*) for important targets and not exhaust them on less significant ones.

Second, it has a big branching factor. In the worst-case scenario, we look at around 200 possible actions. That is: which of the 10 cards in hand play, each of them can be a spell with around 10 potential targets, as well as which of the *Creatures* on the board should attack which of the opponent’s *Creatures*. In addition, each turn comprises several actions, since we can play a few cards, as well as use some *Creatures* from the board to attack the opponent, all in the same turn. According to [7], even in a much simpler situation, with five cards and five targets, we look at a branching factor of 375000 for the whole turn.

Next, there is quite a lot of hidden information. A player does not know what his opponent has in his hand or the deck. Moreover, the deck is shuffled before the start of the game, so the player does not know the order in which they will receive

their cards.

Last but not least is non-determinism. The first part of it is already mentioned shuffling of the decks. The second is connected with the special abilities of cards and highly depends on the game: in some games, almost every card has a random effect, in others, only a few have such effects.

Altogether, we get a type of game that is the middle ground of difficulty to play for an AI. It is harder than chess, which has been thoroughly researched (starting from the DeepBlue [2], through Stockfish [16], up to Alpha Zero [17]). At the same time, it is considerably easier than RTS games, in which real success has appeared only recently [15, 20].

1.4 CodinGame

CodinGame is a platform designed around the concept of facilitating programming learning in a more engaging and enjoyable manner. It serves as a resource for learning various programming languages (with over 25 languages available) as well as different algorithms and algorithmic concepts. This is achieved by encapsulating diverse programming tasks within different types of games. The platform offers three main types of games: solo puzzle games, optimization tasks, and multiplayer games. The inclusion of levels and achievements motivates users to solve more problems, while rankings foster competition and the pursuit of superior results.

Many games on CodinGame feature animations to visualize program execution, along with visual debugging information to aid in debugging processes.

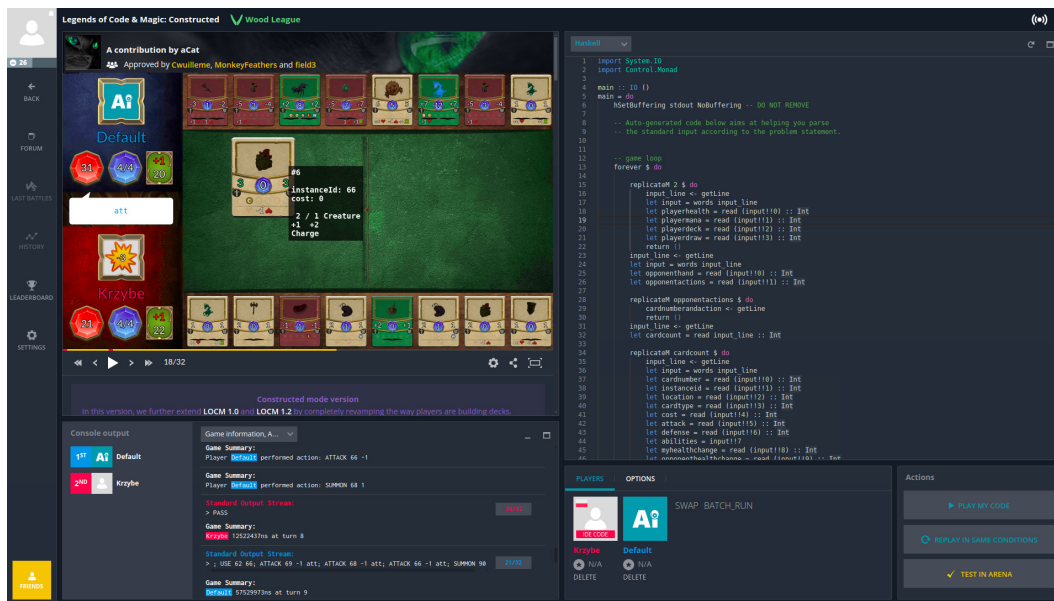


Figure 1.3: Interface of CodinGame

1.4.1 Framework

CodinGame provides a framework on their GitHub repository [5]. This framework includes several classes to use during implementation:

- **GameManager** – The main class responsible for managing the game and transmitting information about input and output to the AI players. There are several subclasses of GameManager, corresponding to the different types of games
- **GameRunner** – Enables local execution of the game. It includes an HTML package for viewing replays in web browsers.
- **GraphicEntityModule** – Facilitates the implementation of graphical representations of the game. It can incorporate simple geometric shapes, load graphics from files, and apply transformations to images.

The entire framework is implemented in Java, with Maven utilized for managing various packages. Graphics are handled by the JavaScript framework PixieJS, under the control of the **GraphicEntityModule** class. Additionally, CodinGame employs the concept of a stub generator, which, when provided with a simple pseudocode, can generate an initial bot for the game.

Although the framework documentation [4] may be lacking, the authors provide a game skeleton [6] and several examples on their GitHub [3].

1.4.2 Technical limitation

Players must communicate using standard streams with the **Referee**. Each player has access to one CPU core and 768MB of RAM, without access to GPU. In multi-player games, up to four players can participate in a single match. All games have a hard limit of 30 seconds for the entire match duration.

Chapter 2

Legends of Code and Magic

Legends of Code and Magic (LoCM) is a CCG, designed to provide an easy-to-work-with environment for research into CCG AI. The game “The Elder Scrolls: Legends” has been used as a starting point while designing the rules.

CCGs frequently feature complex effects that pose challenges during bot creation and complicate accurate simulation by AI. For instance, certain cards in some games not only clear the entire board but also fill it back with random cards. With effects like this, any attempt at simulation would be rendered completely futile.

Because of that, effects in LoCM are kept on the simpler side, with no randomness in the battle phase. Only a few complex rules are implemented, such as splitting the board into two halves (added in version 1.2), runes (removed in 1.5), cards summoning multiple copies of a *Creature* (added in version 1.5) and *Items* affecting multiple *Creatures* (added in version 1.5). The rest of the rules and effects were designed with simplicity in mind. They are discussed in more detail in Chapter 2.3.

All three versions are available for direct access as contests on CodinGame and as a repository for download on GitHub. All of the necessary links can be found on the projects page [10].

2.1 History

Version 1.0 of Legends of Code and Magic was initially introduced in 2018. It debuted on CodinGame through the Sprint contest, which lasted for 4 hours and attracted over 700 participants. Following this, it was featured in the Marathon contest, which spanned 4 weeks and garnered participation from over 2100 contestants. Subsequently, it was made available for future users under the category of Bot Programming [8].

During the initial Marathon contest, the predominant approach for the draft

phase involved the utilization of a predefined ordering of cards. Although some participants experimented with alternative strategies, simpler static ordering strategies proved more effective. In the Battle Phase, search-based techniques dominated, with neural networks being a less popular alternative.

In 2019, version 1.2 [9] was introduced, notable for splitting the game board into two lanes with no interaction between *Creatures* on different lanes.

This version was utilized in several contests during the Congress on Evolutionary Computation (CEC) and the Conference on Games (COG), with prizes sponsored by the IEEE Computational Intelligence Society. Across five editions of version 1.2, a total of 17 unique submissions were received, with some bots being upgraded between contests.

Most participants continued to employ static ordering in the Draft Phase, with only a minority of participants attempting dynamic adjustments to their drafting strategies based on previously selected cards. In contrast, strategies for the Battle Phase exhibited greater diversity, with search-based approaches being the most popular.

The subsequent version, released in 2022, made its debut during COG 2022, attracting 6 participants. Notably, neural network-based strategies emerged as the most favored approach, adopted by 4 of the participants.

All this history has been summarized in more detail in [12] written by the authors of the original version. In it, they review the competitions where LoCM has been utilized, focusing on the perspective of the organizers.

2.2 Research

Legends of Code and Magic have already been used as the subject of several interesting papers.

First, in the work [18], researchers adopted a Reinforcement Learning paradigm to the whole game. They developed separate policy networks for draft and battle phases, training them concurrently via self-play. The resulting agents were assessed through comparison with state-of-art agents available at the time as well as by submitting it to the Strategy Card Game AI Competition.

Then, the authors continued this research in [19]. This time, they concentrated only on the drafting phase. Authors tested three approaches to the history of previous picks — passing them directly, using LSTM or completely disregarding history. Again, their agents were tested by competing against each other and bots from previous competitions as well. What is interesting, in their experiments, the third approach achieved the best results in all the experiments, followed by the second one.

Next, authors of [14] experimented with Online Evolutionary Planning. The evaluating function was created as a linear combination of the health of the opponent, the value of the opponent's and own cards on board, *Items* in hand, and the number of cards to be drawn. The weights of those parameters were tuned using the N-Tuple Bandit Evolutionary algorithm.

On the other hand, authors of [22] decided to use genetic algorithms in their approach. They chose three different scoring functions and used GA to find the best parameters for each of them.

The new version 1.5 was already used in some research. Authors of [21] decided to use end-to-end policy trained using deep reinforced learning coupled with optimistic smooth fictitious play. This approach was quite successful, as their submission won the contest.

Authors of the original game used it for their research as well. One example can be [13] in which they experiment with an evolving function for the evaluation of cards and game state. The studied aspects are different representations of genomes and categories of opponents to train against.

Another example is [11], where authors test the usage of evolution in the draft phase. More specifically, their concentration is on active genes and the assessment of their ideas in terms of the cost of training.

2.3 Rules

Each player starts with 30 Health Points, and their main goal is to reduce the health of their opponent to 0. They can achieve their goal using cards they draw from their deck.

Each player has a deck of 30 cards, from which they can draw them to their hands. One can hold up to 8 cards in hand. Each player has a guaranteed draw of one card per round, but this number can be increased by runes or playing cards with *cardDraw* property.

A player can use some of his cards to summon friendly *Creatures* on the board. This board is split into two lines, and *Creatures* from different lines can not interact with each other. That means that the *Creature* can attack the opponent, or *Creatures* of the opponent, only if they are in the same lane, as shown in the picture below – the incorrect target is marked with a red arrow.

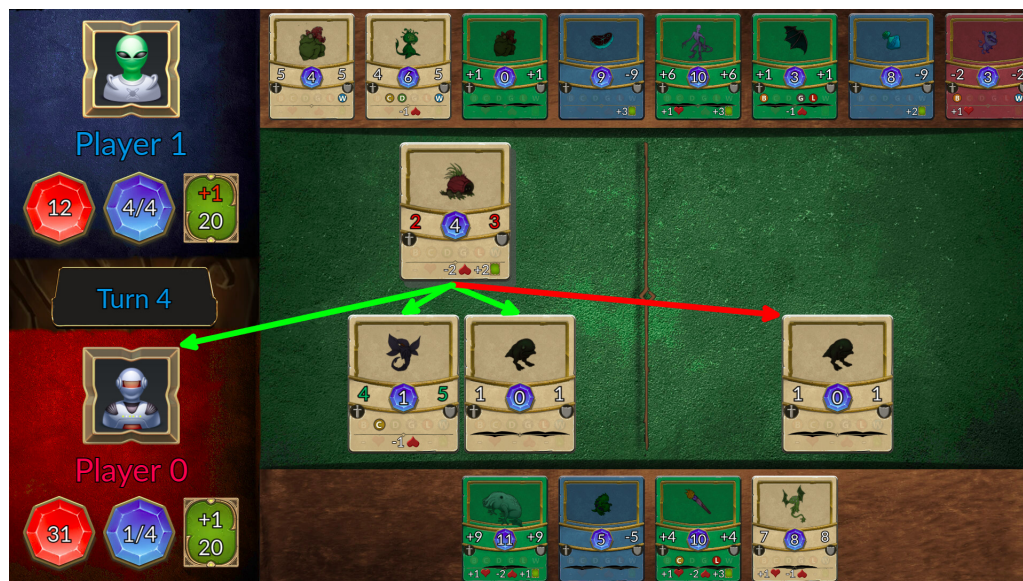


Figure 2.1: Possible attack targets

The game consists of two phases: *Draft Phase* and *Battle Phase*.

2.3.1 Cards

There are two types of cards: *Creatures* and *Items*. Playing any card costs a certain amount of mana, specific to that card.

Each *Creature* has two statistics: attack and defense. The attack represents how much damage that *Creature* can deal to the opponent or to their cards. Defense corresponds to the amount of damage that the *Creature* can withstand.

Figure 2.2: Example *Creature* card

A *Creature* can have special abilities called *Keywords*. There are six of them:

- *Breakthrough*: When a *Creature* with this keyword overkills another *Creature*, excess damage is dealt to the opponent.

- *Charge*: *Creatures* with *Charge* can attack in the same round they were summoned.
- *Drain*: Each time a *Creature* with *Drain* attacks, it heals its owner by the amount of damage dealt to the target.
- *Guard*: Enemy *Creatures* on the same lane as this *Creature* can not deal damage to the player nor other *Creature* without *Guard*. An example can be seen in the Figure 2.3.
- *Lethal*: When a *Creatures* with *Lethal* deals damage to another one, it instantly kills it.
- *Ward*: *Creatures* is completely protected from the next incoming damage.



Figure 2.3: Possible attack targets with Guard

Finally, each card can have properties *cardDraw*, *myHealthChange* and *opponentHealthChange*. The first of them allows the player to draw a certain amount of additional cards in the next turn. The second and third ones respectively give health to the player and damage the opponent.

Next, we have *Items*. There are three subtypes of them:

- *Green Items*: a player can use them on their *Creatures* on board and increase their attack and defense, as well as add keywords to them.
- *Red Items*: they can be used to decrease the attack and defense of opponent *Creatures*, as well as remove keywords from them.
- *Blue Items*: a player can use them to deal damage directly to the opponent or to enemy *Creatures*.



Figure 2.4: Example *Item* cards

2.3.2 Mana

During Battle Phase, a player requires mana to use any card. The first player starts with max mana equal to one, while the second one starts with max mana amounting to two. Then, each round, their max mana is increased by one, up to the max of twelve. Next, their mana is restored to this new max. The second player starts with a higher max mana thanks to the one-time bonus mana, which is considered used after they spend all their mana in a single turn.

2.3.3 Runes

Each player starts a game with five runes. They correspond to 25, 20, 15, 10, and 5 health points. When the player first time hits those thresholds, they lose their respective rune. Additionally, their draw for the next turn is increased by one.

Besides that, runes are used to help to solve the stalemates. When a player does not have any cards left in the deck, then the rune corresponding to the highest threshold explodes, setting their health points to that amount.

This mechanic comes from the game *The Elder Scrolls: Legends*. But, in the original, the additional draft was immediate, and if the drawn card had a keyword

Prophecy, it could have been played instantly. However, here, it was not possible to implement it fully because of the technical limitations of the platform. The decision to play the drawn card would require pausing the current player's turn and giving a chance to the other player to make a choice. CodinGame does not have such an ability. In theory, it could have been done, but would require an overly complicated workaround and would be confusing for a user.

The absence of this keyword significantly diminished the influence of this mechanic compared to its original form, while it still was increasing complication of the game.

2.3.4 Draft Phase

During the Draft Phase, players have to build their decks for the game. It consists of 30 turns, in each of them the game presents two identical sets of three cards to both competitors. Each player chooses one card from each set, without knowledge about the content of future sets. Then, each of the players obtains their own copy of that card. Consequently, they can both take the same card. Those 30 chosen cards make the decks of the players to use during Battle Phase.

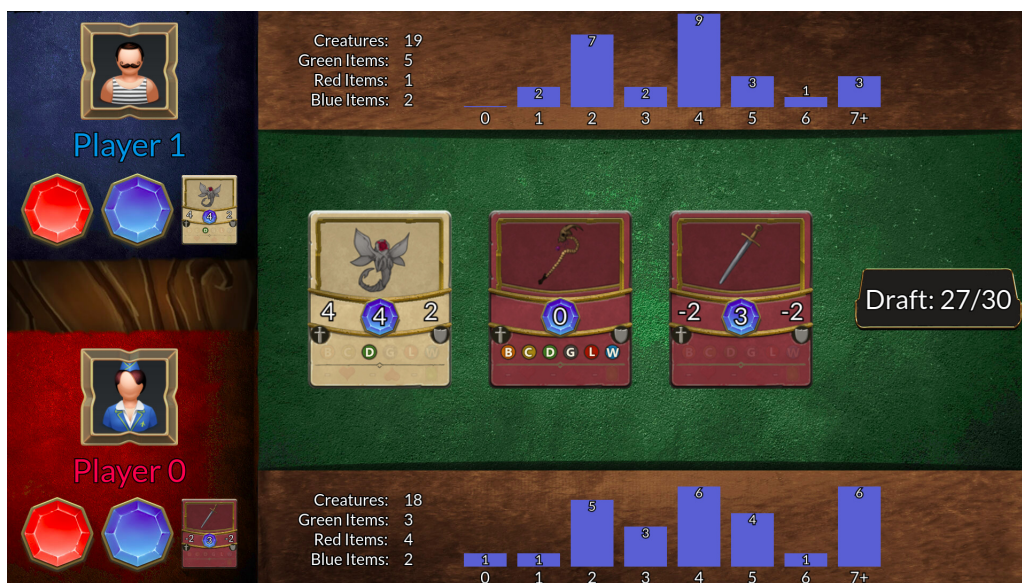


Figure 2.5: Draft Phase

2.3.5 Battle Phase

Through Battle Phase, a player can use three types of actions. The first of them is *Summon*, which puts a *Creature* represented by the chosen card onto the board. The second action is *Cast*, which uses a spell-type card and changes the statistics of a *Creature* on the board, deals damage to the opponent, or heals the player. Last but not least, there is *Attack* action, which allows a player to use one of their *Creatures* to attack one of the opponent's *Creatures* or the opponent himself.

The first two types of actions require a certain amount of mana, specific to the used card. It is represented as the *Cost* on the card.

2.4 Communication

Next, we will discuss communication between the main program and players. For this purpose, we use standard streams. In each turn, the referee sends each player several lines of information about the state of the game. The first part of this state comprises the health of both players, their mana, the number of cards in decks and to draw in the next turn. It also contains information about all the cards on the board and in the current player's hand. Then the player responds with a series of actions separated by a semicolon.

Each action is described by its type, followed by potentially several numerical arguments and then by comment, for example "SUMMON 10 1 **summoning** 10". In this case, it would be the ID of the chosen card equal to 10, the lane to which it should be summoned being 1, and the accompanying comment being "summoning 10".

If a player returns an incorrect action, there are two possible reactions. First, if the type of action is incorrect or there is an incorrect number of arguments, the game is ceased, and the other player is declared a winner. Second, if the value in the argument is incorrect, the player receives only a warning.

Chapter 3

Changes

Version 1.2 of the game performed satisfactorily without major issues. However, it necessitated a few design alterations to serve as a more effective testbed for researching Collectible Card Games.

3.1 Problems

The primary concern revolved around the Draft phase, which failed to properly reward players experimenting with more intricate deck-picking strategies. As evidenced by the Post Mortem [1] for Marathon on CodinGame and in statics for contests during CEC and COG as mentioned in Chapter 2.1, the leading strategy in the Draft phase involved offline ordering of cards. Many participants opted to settle in advance an order of cards and then during gameplay, select the highest-ranked card according to this order. While some attempted more dynamic approaches, the lack of success led them to revert to offline ordering. Notably, all contests for version 1.2 were won by bots employing offline ordering strategies.

This issue partly stemmed from the absence of complex card effects, resulting in limited interactions between cards. In most CCGs, various categories of *Creature* equivalents exist, with cards specifically interacting with these categories. For instance, in Hearthstone, the card "Scavenging Hyena" possesses the effect "Whenever a friendly Beast dies, gain +2/+1," encouraging players to prioritize acquiring more Beasts and cards that interact with them. As a further example, in The Elder Scrolls: Legends, the card "Soul Split" allows players to "Sacrifice a creature to summon 3/2 Sundered Shade in each lane," offering strategic value in utilizing heavily wounded or weak *Creatures*.

Regrettably, effects of this nature may not smoothly integrate into a game primarily designed for AI research in CCGs, prioritizing accessibility and ease of experimentation.



Figure 3.1: Example cards from other games

Unfortunately, simplifying rules in LoCM led to a lack of such intricate interactions. The only incentive to build a more complicated drafting bot was roles of cards like Tank or Remover as well as mana curve. But, as mentioned earlier, more complicated drafting approaches worked similarly or even worse than a pre-defined order.

The final problem relates to Runes, which were initially designed with a specific card effect in mind – *Prophecy*, as discussed in section 2.3.3. However, without the presence of this ability, this mechanic became significantly less impactful compared to its original conception, while still contributing to the overall complexity of the game.

3.2 Constructed Mode

The most significant and important change implemented in the project was the introduction of *Constructed Mode* designed to replace Draft. In this mode, players construct their entire deck at once, as opposed to gradually assembling it through numerous small decisions. The aim is to incentivize the creation of more complex decks and stimulate the development of more elaborate strategies.

In Constructed Mode, both players are presented with the entire pool of 120 cards generated for this game, utilizing the generation process described in the chapter Chapter 4. From this pool, players must construct their decks. Each player has the option to choose a single card twice, resulting in two unique copies with distinct IDs. The deck size remains unchanged, at 30 cards.

The visualization of this phase is depicted in the picture below. Numbers displayed in the corners of the cards indicate how many times this card has been chosen by the player with the corresponding color. Due to space limitations and the extensive number of cards, the list of available cards is divided into three frames to accommodate them all. Consequently, supplementary and purely visual turns were introduced to manage this aspect seamlessly. However, the mana curve visualization positioned above and below the main board remains unchanged.



Figure 3.2: Constructed Phase

3.3 Area

There are some popular effects present in many CCGs. Among them, one allows one to summon multiple *Creatures* at the same time using a single card, while another involves *Items* that affect several targets simultaneously. Examples from TESL are depicted in the picture 3.3 below. Notably, the design of the Scouting Patrol card revolves around the mechanic of multiple lines, specific to TESL. Cards like those served as inspiration for the introduction of the new effect in version 1.5: *Area*. This effect, while simple, has a significant impact on gameplay and addresses the second part of the main problem of previous versions: the lack of more diverse and interesting effects. Furthermore, it provided a valuable type of card: boardclear which comprises *Items* capable of removing multiple weak *Creatures* simultaneously.



Figure 3.3: Inspirations for Area

The Area effect can have three possible values: *Target*, *Lane1*, *Lane2*. Examples of cards with these effects are shown in the graphic 3.4 below, with the value of the Area represented by the black wave in the bottom part of the card.

Figure 3.4: Example cards with Area keyword. From left: *Target*, *Lane1*, *Lane2*

Cards with *Target* function exactly as before summoning and affecting only one *Creature*.

For *Creature* cards with the *Lane1* and *Lane2* effects, multiple copies are summoned when used. *Lane1* creates two copies on the specified lane, while *Lane2* puts one copy on each lane (with the argument corresponding to the target lane ignored). Needless to say, the limit on the number of *Creatures* in each lane still applies, so an additional copy is summoned only if there is space available. The possible effects of *Creature* cards with the Area effect are illustrated in the picture 3.5 below, with green arrows symbolizing *Lane1* and blue arrows symbolizing *Lane2*.



Figure 3.5: Effects of *Lane1* and *Lane2* on *Creature* card

For the *Items*, this effect functions a little differently. A spell with *Lane1* affects all *Creatures* in the same lane as its original target, while *Items* with *Lane2* affect all *Creatures* on all lanes, (with the argument corresponding to the lane ignored). Naturally, in both cases, *Items* only impact friendly or enemy *Creatures*, depending on the type of the spell. Below, in picture 3.6, we can see which targets would be affected by *Items* with *Lane1* and *Lane2* effect, with green arrows representing *Lane1* and blue arrows representing *Lane2*.

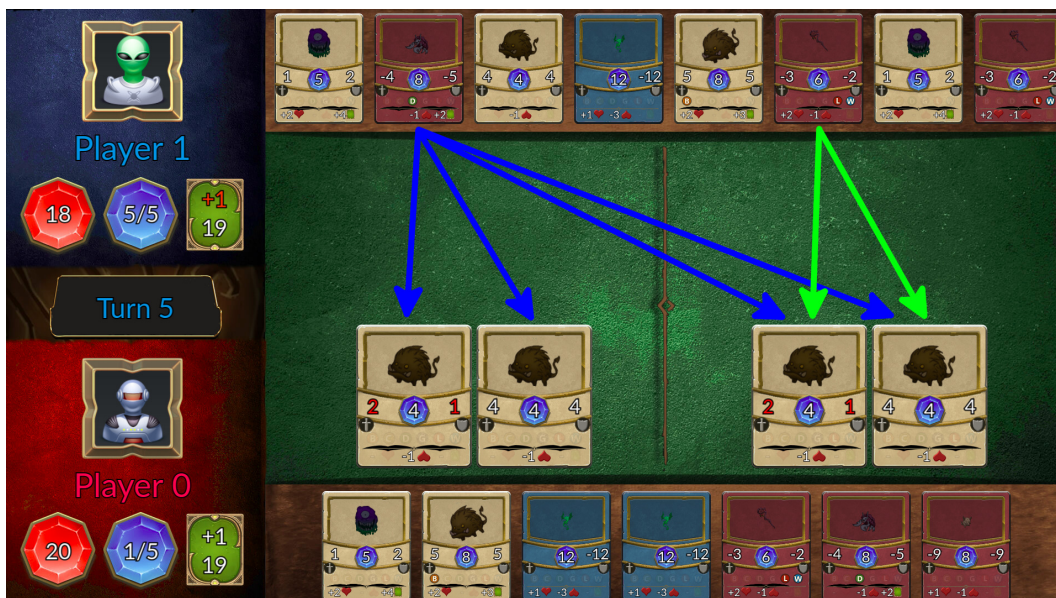


Figure 3.6: Effects of *Lane1* and *Lane2* on *Item* card

Additionally, effects like *cardDraw*, *myHealthChange*, and *opponentHealthChange* activate once for each time this card is triggered. Consequently, for *Creatures*, these effects activate once for each summoned copy, while for *Items* they activate for each impacted *Creature*. This further enhances the value of the Area effect.

3.4 Runes

Another change involves the removal of runes. As stated in the section 3.1, they did not provide enough for the game. Their removal may initially appear straightforward, but it requires a series of smaller changes across various aspects of the game.

First, of course, is the removal of the most obvious use of Runes. That is the logic behind managing the triggering and detonating of a rune, as well as providing a player with an additional card.

Additionally, runes were previously employed to resolve stalemates. When a player drew their whole deck, consecutive runes would explode in successive turns and set that player's health to the corresponding value. To compensate for their removal, we have replaced this mechanic with constant damage of 10 health points per card the player is supposed to draw. This modification offers strategic depth, particularly benefiting players who construct decks with emphasis on Drain, accumulating this way significant amounts of health.

Another associated change involves the format of the input data. Input sent to each player contained information about the number of still active runes for both them and their opponent. Consequently, adjustments had to be made to all the exemplary bots including the boss agent, to accommodate this revised input format.

Lastly, the representation of runes in the user interface (UI) had to be removed to align with their removal from the game mechanics.

Chapter 4

Generation

A significant change was connected with cards utilized within the game. It has been decided to move away from a predefined list of cards employed consistently across all games. Instead, a new approach was adopted – generating a unique set of cards for each game. This process encompassed not only generating values for the cards but also creating their corresponding images. This change was introduced to reinforce the necessity of evaluating cards online instead of relying solely on offline ordering.

4.1 Card Generation

Our card generator mechanism utilizes three types of Random Generators:

- `WeightedGenerator` – this generator employs a typical uniform distribution to select one object from a set of possibilities, each with a weight assigned
- `NormalGenerator` – uses the normal distribution to produce a random number
- `ShuffleGenerator` – it randomizes the order of a provided list through shuffling

The process of generating a card takes place in several fazes. To start with, we determine the mana cost of the new card using the `WeightedGenerator` with possible values ranging from 0 to 12. By default, all weights are equal. Next, we employ a similar generator to choose the card type.

In the subsequent step, we use `ShuffleGenerator` to shuffle all the remaining properties, apart from attack and defense. Afterward, we attempt to add them until we exhaust the pool of candidates or reach the budget limit for the new card. Following this, we allocate the remaining budget to determine the attack and defense values. This is accomplished using the `NormalGenerator` with a mean of 1 and a standard deviation of 2, shifted by the remaining mana.

As a final, tweaking step, we align the values of attack and defense with the type of card, following several rules:

- if either attack or defense is negative, it is set to 0,
- for *Creature* cards, the defense must be at least 1,
- for *Blue Items*, attack is set to 0 as it is disregarded,
- *Red* and *Blue Items* use exclusively negative numbers, hence we multiply their attack and defense values by -1.

Each property is associated with two costs – additive and multiplicative. By default, each of the generators utilizes only one of them, while the second one is set to neutral value – 0 and 1 respectively.

Keywords are treated differently in their generation process. It is done in two stages – first, we determine the number of *Keywords* to add, similar to the mana selection process. Next, we shuffle the *Keywords* and systematically attempt to add them to the new card. If budget constraints prevent the addition of all *Keywords*, we conclude this phase without filling all available slots.

Although the default mode for selecting the set of cards for a game is now to generate them dynamically, sometimes it is useful to have the ability to use the card set created in advance. For this reason, there has been added possibility of specifying a file with such a set to use.

4.2 Example of Generation

In the picture 4.1 we can see an example card generated using our generator. We will now use it to go through the generation process:

1. Initially, `typeGenerator` provided us with a *Red Item* (probability for *Creature*:40%, every color of *Item*:20%).
2. Then, the `manaGenerator` selected a cost of 10 mana (with uniform probability for values [0-12]).
3. Following this, the `ShuffleGenerator` within the `propertyOrderGenerator` provided us with the order of properties [area, oppHealthChange, Keywords, draw, myHealthChange].
4. The `areaGenerator` choose an *Area* value of *Lane1*, reducing the budget to 7 mana (with probabilities *Target*:50%, *Lane1*:25%, *Lane2*:25%).
5. The `oppHealthChangeGenerator` choose 0, leaving the budget unaffected at 7 mana (probability for 0:50%, -1:25%, -2:12.5%, -3:12.5%).

6. The `keywordsGenerator` chose 4 as maximum number of *Keywords* to add. Utilizing the `shuffleGenerator`, it determined that the initial 4 *Keywords* to consider were *Ward*, *Lethal*, *Charge*, and *Guard*. These *Keywords* collectively consumed 4 mana, leaving us with 3 mana.
7. The `drawGenerator` provided us with +2 draw, depleting the budget to 2 mana (with probability for 0:48.8%, +1:24.4%, +2:12.2%, +3:12.2%, +4:2.4%).
8. The `myHealthChangeGenerator` selected a value of +3, reducing the remaining mana to 1 (probability for 0:50%, +1:25%, +2:12.5%, +3:12.5%).
9. The `attackGenerator` and `defenseGenerator` yielded values of 2.2077343103493865 and 1.6360940146843843 respectively (from normal distribution with standard deviation 2 and mean 1). Following the adjustment by the remaining budget and rounding down, we obtained values of 3 and 2. Since the new card is categorized as a *Red Item*, these values were multiplied by -1, resulting in final values of -3 and -2.



Figure 4.1: Example card generated using new generation method

In picture 4.2, we observe an example set generated through the described procedure. As we can see, it has produced a varied array of distinct cards.



Figure 4.2: Example card set generated using new generation method

4.3 Generation Configuration

Naturally, hardcoding all the values dictating the card generation process would complicate potential experiments with specific types of card sets. To enhance accessibility, all the steering values are extracted from a designated file `cardWeights.json`, allowing for easy modification. In the picture 4.3 we can see part of the default configurations for different generators.

```
"typeProbabilities": {
  "creature": 0.4,
  "itemGreen": 0.2,
  "itemRed": 0.2,
  "itemBlue": 0.2
},
"areaProbabilities": [{
  "name": "target",
  "weight": 50,
  "multCost": 1,
  "addCost": 0
}, {
  "name": "lane1",
  "weight": 25,
  "multCost": 0.7,
  "addCost": 0
}, {
  "name": "lane2",
  "weight": 25,
  "multCost": 0.6,
  "addCost": 0
}
],
"bonusAttackDistribution": {
  "mean": 1.0,
  "std": 2.0
}
```

Figure 4.3: Fragment of default `cardWeights.json`

4.4 Card Images

The next part of the game to be changed was the images of the cards. Before, each card had its own picture, assigned forever.

But that approach works well only with the original card set. Especially, adding new cards was awfully complicated, as each new card would require a new image. It

got even worse when we changed from a pre-defined set of cards to generate a new set for each game.

Because of this, we had to change the way of assigning pictures to cards. Now, we will reassign images to cards at the beginning of each game. We will do so by looking at the cards chosen for Constructed Mode and assigning different images for each of them based on their ID and random offset.

However, then in each game, we would use exactly the same picture for completely different cards. We solve this problem by altering its appearance. To achieve this, we add a colorful mask, rotation, and scaling. But we do not want to make those adjustments entirely random, so we will base them on the statistics of the card and its ID:

- We add three colored components for three main values of the cards: red for the attack, green for the defense, and blue for the cost. The value of each mask corresponds to the respective statistic. One can view an example of colors in figure 4.4.
- Stronger cards should be bigger, so we scale the picture based on the sum of the attack and defense of the card.
- We further alter the appearance of the card by adding a small rotation to the picture. For this purpose, we calculate the angle as a hash of the statistic of the card. Then, we choose the direction of rotation using the parity of its ID.

Together, those three adjustments give us quite a wide spectrum of diverse images, as could have been seen in the picture 4.2 above.

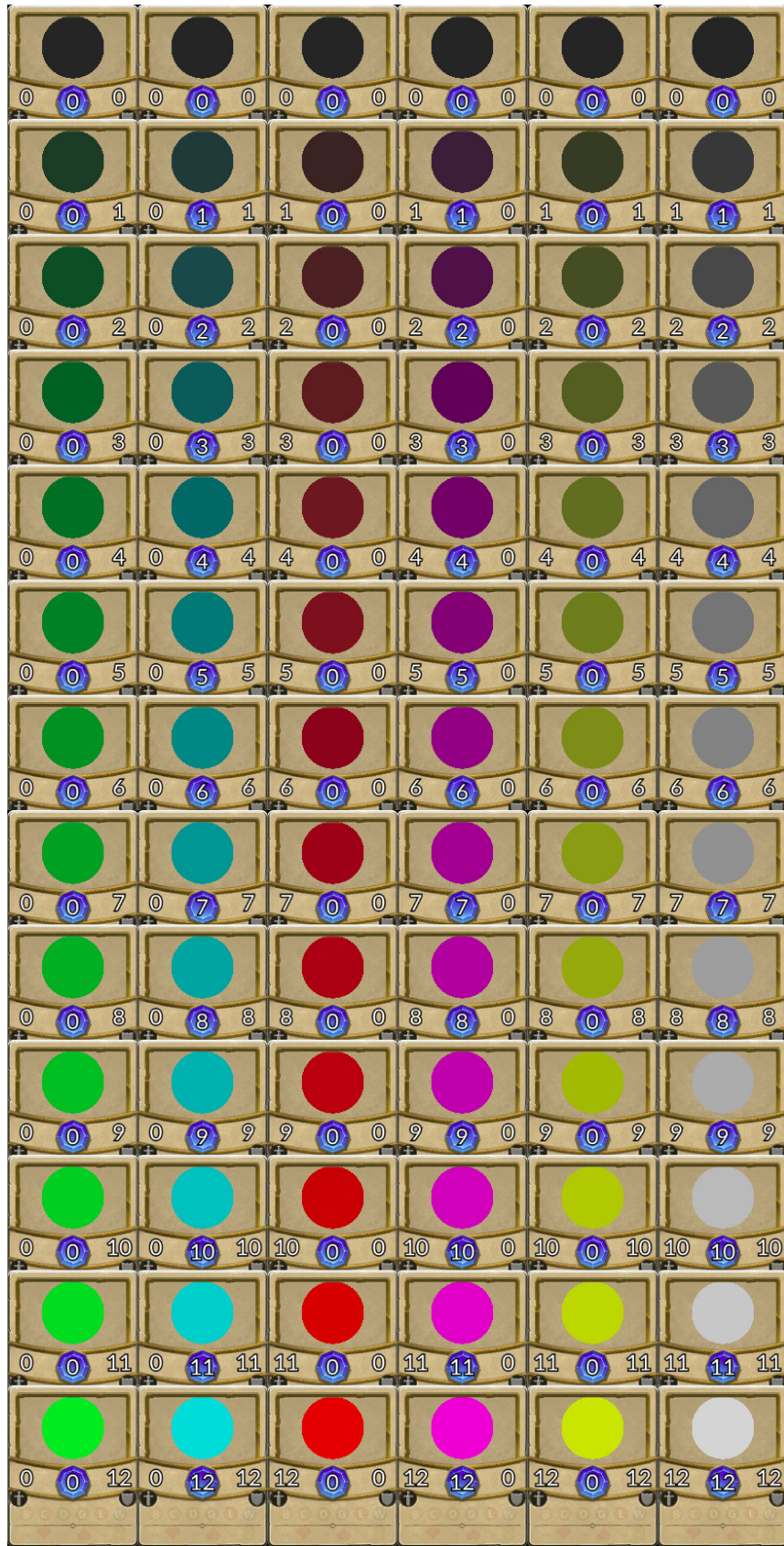


Figure 4.4: Example color masks

Chapter 5

Summary

Changes made for new version 1.5 have been overall successful.

The new version has been already used during the contest for COG 2022 without major problems. The competition has been won by team ByteRL, with NeteaseOPD in second and Inspirai in third place.

Most of the planned work has been accomplished, and there are only two pieces missing. First, copies summoned by *Creatures* with the keyword *Area* have an ID generated as `biggest_ID_yet+1`. It is unintuitive and requires the user to follow what IDs have been used. It will be changed, so cards in decks will be given evenly spaced out IDs, and additional copies will use numbers in between.

Second, cards generated by the generator are not well balanced. For example, a spell with the *Lane2* keyword costs 1.6 times more than one with the *Target*. But it almost always will have several correct targets, making its value a lot higher. Because of problems like that, weights and costs in the generator will have to be rebalanced.

Bibliography

- [1] Legends of code and magic: version 1.0 - feedback and strategies from contestants”. <https://www.codingame.com/forum/t/legends-of-code-magic-cc05-feedback-strategies/>. Accessed: 2024-03-29.
- [2] M. Campbell, A. Hoane, and F. Hsiung Hsu. Deep blue. *Artificial Intelligence*, 134(1):57–83, 2002.
- [3] CodinGame. Example game projects on github. <https://github.com/CodinGame>. Accessed: 2024-03-29.
- [4] CodinGame. Framework documentation. <https://www.codingame.com/playgrounds/25775/codingame-sdk-documentation/introduction>. Accessed: 2024-03-29.
- [5] CodinGame. Repository with framework engine. <https://github.com/CodinGame/codingame-game-engine>. Accessed: 2024-03-29.
- [6] CodinGame. Repository with game skeleton. <https://github.com/CodinGame/game-skeleton>. Accessed: 2024-03-29.
- [7] A. K. Hoover, J. Togelius, S. Lee, and F. de Mesentier Silva. The many ai challenges of hearthstone. *KI - Künstliche Intelligenz*, 34(1):33–43, Mar 2020.
- [8] J. Kowalski and R. Miernik. Legends of code and magic. version 1.0. <https://www.codingame.com/multiplayer/bot-programming/legends-of-code-magic>. Accessed: 2024-03-29.
- [9] J. Kowalski and R. Miernik. Legends of code and magic. version 1.2. <https://www.codingame.com/contribute/view/162759566f5a132f64b4de78ed637a2f309a>. Accessed: 2024-03-29.
- [10] J. Kowalski and R. Miernik. Main page of legends of code and magic. <https://legendsofcodeandmagic.com/>. Accessed: 2024-03-29.
- [11] J. Kowalski and R. Miernik. Evolutionary approach to collectible card game arena deckbuilding using active genes. In *IEEE Congress on Evolutionary Computation*, pages 1–8, 2020.

- [12] J. Kowalski and R. Miernik. Summarizing strategy card game ai competition. In *IEEE Conference on Games*, pages 1–8, 2023.
- [13] R. Miernik and J. Kowalski. Evolving evaluation functions for collectible card game ai. In *International Conference on Agents and Artificial Intelligence*, volume 3, pages 253–260, 2022.
- [14] R. Montoliu, R. Gaina, D. Perez Liebana, D. Delgado, and S. Lucas. *Efficient Heuristic Policy Optimisation for a Challenging Strategic Card Game*, pages 403–418. 04 2020.
- [15] S. Ontañón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss. A survey of real-time strategy game ai research and competition in starcraft. *IEEE Transactions on Computational Intelligence and AI in Games*, 5(4):293–311, 2013.
- [16] T. Romstad, M. Costalba, J. Kiiski, and et al. Stockfish: A strong open source chess engine. <https://web.archive.org/web/20220804024025/https://stockfishchess.org/>. Accessed: 2022-08-08.
- [17] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.
- [18] 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*, pages 611–614, 2019.
- [19] 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)*, pages 54–61, 2020.
- [20] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, Nov 2019.
- [21] W. Xi, Y. Zhang, C. Xiao, X. Huang, S. Deng, H. Liang, J. Chen, and P. Sun. Mastering strategy card game (legends of code and magic) via end-to-end policy and optimistic smooth fictitious play. *arXiv preprint arXiv:2303.04096*, 2023.

- [22] Y. Yang, T. Yeh, and T. Chiang. Deck building in collectible card games using genetic algorithms: A case study of legends of code and magic. In *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 01–07, 2021.