Proceedings of the Sixteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-20)

Chaos Cards: Creating Novel Digital Card Games through Grammatical Content Generation and Meta-Based Card Evaluation

Tiannan Chen, Stephen J. Guy
Department of Computer Science & Engineering, University of Minnesota
{chen2814, sjguy}@umn.edu

Abstract

We present a novel framework to procedurally generate executable cards for *Hearthstone*-like digital card games, along with an approach for evaluating card strengths using an evolved match environment. Here we introduce *Chaos Cards*, a digital card game in the style of *Hearthstone*, but designed to support the procedural generation of cards, including their diverse effects, via a grammatical model. To understand the potential performance of procedurally generated cards in actual games, we integrate a simulation-based approach to evaluate card strengths, and train a neural network model for fast card strength prediction. Because the strength of a card is most meaningful when considered in the context of the pool of competitive decks (know as the meta) it plays in and against, we propose an evolutionary evaluation approach which simultaneously evaluates card strength and refines the environment in which cards are tested. We showcase some example cards generated by our framework, along with their strength evaluations. Additionally, we conduct tests between evaluations from meta game environments and random game environments to show the importance of the environment in evaluating card strengths. Lastly, we show our neural network is able to learn the strength of important cards in a meta environments with largely positive correlation.

Introduction

Procedural content generation (PCG), the algorithmic creation of content with limited or indirect user input (Togelius et al. 2011), has been adopted in commercial games such as *Diablo III* (Blizzard) and *No Man's Sky* (Hello Games), and studied in academic context including generating furniture layouts (Germer and Schwarz 2009) and textures (Dong et al. 2019). It is also found in domains beyond entertainment, such as automatically generating code as test cases for software verification (Claessen and Hughes 2011) or randomly generating math quizzes (Tomás and Leal 2013). As a powerful tool for creating large amounts of diverse data and continuously introducing new experiences, PCG is often considered part of a mixed-initiative co-creativity (Yannakakis, Liapis, and Alexopoulos 2014).

Collectible card games (CCGs) are a popular type of turn-based strategy games, where players use decks built from the cards they collected to compete against other decks. CCGs date back to as early as the physical form of *Magic the Gathering* (Wizards of the Coast) in 1993, but now appear more often in digital forms, such as *Hearthstone* (Blizzard) and *Shadowverse* (Cygames). CCGs have been an interesting problem domain for AI, including deck optimization (García-Sánchez et al. 2016), gameplay strategies (Świechowski, Tajmajer, and Janusz 2018), and game balancing (Jin 2019), due to diverse card effects and complex interactions between cards in an adversarial setting. These properties of CCGs also make them interesting for studying the PCG of cards, which has potential applications in assisting the creation of new cards and introducing new game modes where cards are generated on-demand.

PCGs for cards in CCGs give rise to three important challenges. The first challenge is how to model the complex card effects for procedural generation. Effects can be of various types (e.g. damaging, or healing) and magnitudes (e.g. how much damage it deals). Effects can be triggered by different events (e.g. battlecries and deathrattles are triggered when minions are played and destroyed respectively), or may require certain conditions (e.g. having leader's health below 15). Certain effects may even recursively attach effects to other cards or spawn other cards.

The second challenge is in understanding the impact of generated cards on game balance, mainly the strength of generated cards. Balancing is an important aspect of games, heavily impacting player experience, and drawing large amounts of designer effort. Automatic evaluation is especially important for procedurally generated content because of their amount and stochasticity. Due to complexity of the interactions and being in an adversarial setting, card strength is not easily defined on a card in isolation, but should be tested in the context of (simulated) matches. More concretely, the strength of a card depends heavily on the existence of other cards and decks, known as the environment of the game. In popular CCG games, the environment evolves quickly to one that contains predominately competitive decks, called the/a meta or meta game, which means defining card strengths with regard to the meta is highly de-

sired. Additionally, as game simulations are generally costly as an evaluation method, fast prediction on strengths of new cards requires training an efficient model, such as a neural network model, based on simulation results, which has its own challenge due to the complexity of the cards.

The third challenge is how to integrate card generation with their in-game execution, i.e., how to generate executable cards instead of only card descriptors. Card execution is necessary for cards to participate in actual games, and is also needed by the simulation-based evaluation scheme. Generating executable cards is important, because it saves us from the implementation and execution of a parser/interpreter, and avoids unnecessary ambiguity issues when using certain descriptor types (e.g. pure text description would be hard for machine interpretation).

To address these challenges, we propose a framework combining the procedural card generation and card strength evaluation. First, we create a prototype digital card game, Chaos Cards, using a recent grammatical PCG tool, Grammatical Item Generation Language (GIGL) (Chen and Guy 2018), which helps seamlessly integrate the card generator with game mechanisms, enabling the procedural generation of executable cards. Second, we propose a meta-based card evaluation scheme, where we evaluate card strengths by simulating matches between decks that are created with an evolutionary scheme. This scheme enables the environment to converge to a meta, where card evaluations are likely to be more meaningful. Third, we train a recursive neural network (Socher et al. 2011) model for fast prediction of card strengths. Recursive neural networks capture grammatical or hierarchical structures well, and are suitable for content that is generated through such models.

Our work here presents three main contributions:

- *Chaos Cards*: We create Chaos Cards, a novel prototype digital card game with procedurally generated executable cards, as a platform for studying PCG for adversarial content generation with complex grammatical structures.
- *Evolutionary Simulation-based Evaluation*: We propose a evaluation method for card strengths that simulates matches while evolving for the meta, which provides a new perspective of evaluating combinatorial elements (cards) in adversarial settings.
- *Card Strength Predictor*: We train a deep learning model based on recursive neural networks, providing an efficient way of estimating card strengths.

Related Work

Here we briefly survey related work in grammatical structures in PCG and studies on CCG related topics.

Grammatical PCGs

Grammar-based models have been widely used in PCG studies. Grammars have been used for generating Mario Levels (Smith et al. 2009; Shaker et al. 2012), environments in an endless-run type game (Toto and Vessio 2014), instances of MMORPG games (Merrick et al. 2013), and 2D maze or puzzle games (Khalifa and Togelius 2020). They are also used in procedural modeling of 3D scenes (Krecklau and Kobbelt 2012), cities (Parish and Müller 2001; Talton et al. 2011), villages (Emilien et al. 2012), and complex buildings and landscapes (Merrell and Manocha 2008). Grammatical generation is also used with natural languages to procedurally generate sentences (Kempen and Hoenkamp 1987) and stories (Ammanabrolu et al. 2019).

There have also been work on providing tools to facilitate grammatical PCGs, many in the form of domain-specific languages (DSLs). Recent examples include *HyPED* (Osborn, Lambrigger, and Mateas 2017) which targets action games, *Tracery* (Compton, Filstrup, and Mateas 2014) which support grammar-based procedural story authoring, and *Ceptre* (Martens 2015) which encodes rules for interactive narratives and strategy games. Other closely related works include *Expressionist* (Ryan et al. 2016), which is an in-game text generator authoring tool based on probabilistic context-free grammars, and *CatSat* (Horswill 2018), a logic language for PCG tasks based on answer set programming. Our implementation is able to procedurally generate executable cards by using *GIGL* (Chen and Guy 2018), a DSL supporting the implementation of generic probabilistic grammars with compile time integration with C++ code.

CCG Studies

CCGs, especially *Hearthstone*, have been used as the problem domain for many AI studies. For a recent review on CCG studies, we refer to an article by Hover et al. (2020). Most works on CCGs study deck building strategies (García-Sánchez et al. 2016) or game playing strategies (Grad 2017; Świechowski, Tajmajer, and Janusz 2018), and Zook et. al. (2019) takes this a step further by using trained gameplay AIs to discover design flaws. A few studies (Jin 2019; de Mesentier Silva et al. 2019) work on balancing the game by analysing or adjusting numerical attributes on existing cards. Close to our aim is the work of Ling et al. (2016) which studies the generation of program code given text description of cards. In contrast though, we rely on authored grammar rules to directly generate cards so as to guarantee the results are executable. We then further work to understand the impact of the resulting cards on gameplay.

Chaos Cards

Chaos Cards is a prototype digital CCG we create as a problem domain for this work, with executable cards generated via a grammatical model. Players make decks of 20 cards, where they take turns spending a limited pool of mana to play cards. The game rules are similar to the game *Hearthstone* where both players try to defeat the opponent leader by attacking with minions or playing spell cards. As compared to *Hearthstone*, some aspects have been simplified such as the weapon system is replaced with an attack value attached to the leader (see tinyurl.com/chaoscards-rule). Chaos Cards is implemented in GIGL/C++, with text-based gameplay.

Grammatical Card Generation

Cards in Chaos Cards and other CCGs have a structured format of cost, stats, effects, etc. Card effects are diverse with

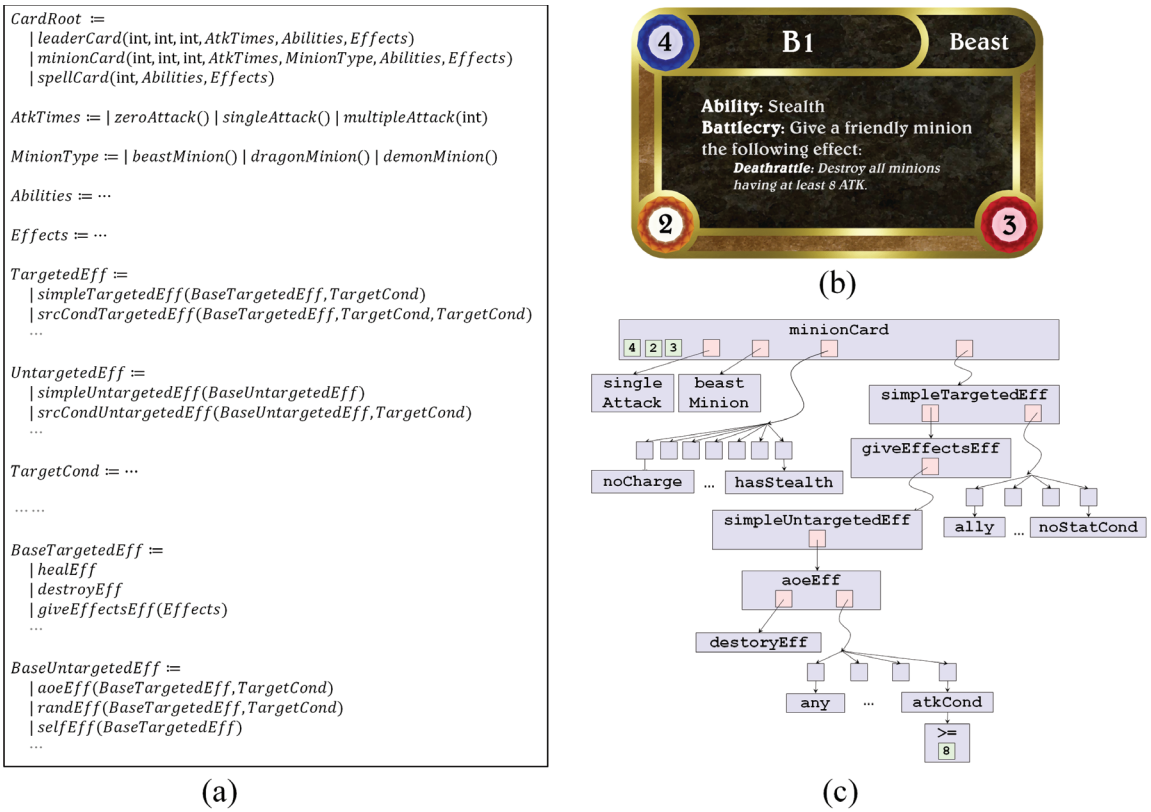


Figure 1: Illustration of structured card generation. (a) Item grammar for card generation ("..." indicates omissions). (b) rendering of an example generated card. (c) Tree representing the internal structure of the example card (curving arrow indicates omitted paths and branches). Card arts and names (e.g. B1) in this paper are added manually for showcasing. The full grammar has 33 nonterminal types and 132 rules (see Additional Materials).

various combinations of triggers, conditions, and impacts which can be well represented in a grammatical structure. In addition, the possibility of recursive effects, such as ones giving effects to other cards/characters (seen in the example in Figure 1) makes probabilistic grammars a good choice for the procedural card generation. We do not select a simpler model, such as a sequence model, because such representations either have difficulties in being executable (e.g. card text or tokens) or heavily limit the diversity of cards. The card generator is implemented in GIGL (Chen and Guy 2018), which is designed for encoding grammatical PCGs and provides compile time integration with C++ code, accommodating the need for diverse executable cards.

The probabilistic grammar to generate content is called an *item grammar* in GIGL, which has named expansion rules. Figure 1a shows a key part of the item grammar for cards in Chaos Cards. The grammar contains rules specifying how a nonterminal node can be further expanded to nonterminal nodes (to apply additional rules) or terminal nodes (i.e., specific values, including empty). The card generation process starts from a starting node (of *CardRoot* type) and stochastically and recursively selects from rules to expand nonterminals until all branches reach terminals. For example, a *CardRoot* node can be expanded by stochastically choosing from one of the three rules, *leaderCard*, *minionCard*, or *spellCard*. If selecting *minionCard*, then three integer terminals needs to be determined for cost, attack, health respectively, and four other nonterminals of type *AtkTimes*, *MinionType*, *Abilities*, *Effects* need to be further expanded. The *giveEffectsEff* rule (give extra effects to cards) is an example of recursion, as it goes back to a nonterminal type (*Effects*) the expansion already passed through. GIGL allows constraints to be placed to limit the nesting depth of such effect to prevent over-complicated cards.

Each time a rule is selected, corresponding human-readable text (text part in Figure 1b) is also generated. The end result is a card with a clear text description of its effect, along with an executable C++ object preserving the tree structure (Figure 1c) of the generated card.

Card Strength Evaluation

Understanding the strength of cards is crucial for balancing. We adopt a simulation-based evaluation for card strengths, where cards are evaluated with simulated matches to provide an objective evaluation of strength. Additionally, these simulated matches provide an environment (cards and decks), as the context needed for strength evaluation. Cards are best thought of not as weak or strong on their own, but rather as effective or ineffective within the context of other cards

shared in its deck or played against it in the opponent's deck.

Simulation AI

Automated simulations of matches requires a gameplay AI. Very strong AI can be slow to compute, which reduces the number of simulated matches used in evaluation. To balance effectiveness and performance considerations, we adopt a one turn look-ahead search-based AI using a heuristic on game state. The AI does not cheat, but rather perform several attempts with random cards filling in for unknown cards (e.g ones in its opponent's hand). Similar to Monte Carlo Tree Search (Chaslot et al. 2008), actions are chosen stochastically in a way that balances on exploring less tested actions and retrying actions which were tested promising. The heuristic reward at the leaf of the tree is the based on the end of turn game state, primarily the relative difference on the total attack plus leader health between the two sides.

Card Strength Evaluation Metric

Evaluating card strength numerically requires defining a metric based on the data from the simulated matches. In general, a card participating in more winning matches should have a higher strength rating. However, directly considering win rate may be too noisy, especially for cards which were rarely drawn or played. Therefore, we use a weighted win rate as an evaluation metric:

$$s_c = \frac{\sum_k w_{c,k} r_{c,k}}{\sum_k w_{c,k}} \tag{1}$$

where s_c is the strength evaluation for card c, k goes over each participating deck in each match (iterate twice for each match to account for both sides), $w_{c,k}$ is a weight denoting the contribution of the card in the deck for the match, and $r_{c,k}$ denotes the result (1 for win, 0 for loss, 0.5 for draw) of the corresponding match-side pair denoted by index k. The contribution $w_{c,k}$ is decided by how many times the card is drawn, comes on to the field and has an effect on it being triggered, and normalized to a sum of 1 for all cards in the deck. This metric essentially means we weight the result of each match by the participation of the card in the match.

Evolving for the Meta

As mentioned, card strengths are most meaningful when considered in a meta game, i.e. an environment with most of the decks being highly competitive. In order to meaningfully evaluate them then, we must also establish a meaningful environment for the cards to participate in, i.e. the meta. We create this environment through an evolutionary approach that refines the decks in the environment while performing simulated matches to evaluate card strengths. Genetic evolution components are involved in the process, including cross-over and mutation, as they are effective ways to explore combinatorial spaces of decks. At each round, decks with relatively competitive performance are probabilistically kept for the next round, and are subsequently modified to explore new potentially better decks.

Our approach to evolving the deck pool used in testing follows an iterative approach with the following three steps:

1. **Modification Proposition** The deck pool may be modified in one of three ways: creating a new deck by selecting individual cards, performing a cross-over of random two decks from the deck pool which generates two offspring decks, or performing a mutation on a random deck from the deck pool by replacing one random card in it. Considering exploration versus exploitation balance, cards with higher ratings or less testings are more likely to be chosen for creating a new deck or mutating.
2. **Evaluation Update** A set of matches are performed matching each of the new decks against each deck in current deck pool. Each pair in the evolution process is tested with repeated matches and with alternation in playing order. A deck's strength rating is based on its win rate.
3. **Deck Pool Update Decision** The reference deck(s) which new deck(s) are compared to depends on the type of modification chosen in step 1. A new deck modification attempts to replace the worst rated deck in the deck pool, a cross-over attempts to replace the parents selected for cross-over, and a mutation attempts to replace the deck it mutates from. This choice of replacement targets is made so that we either maximize the potential improvement, or improve decks by replacing instead of populating decks of similar styles, which prevents the deck pool from being too homogeneous. Decks which improve upon the reference deck are always selected, while decks which come close but do not improve on the reference deck may still be chosen so as to ensure sufficient exploration of space of decks using the Metropolis-Hasting Markov Chain Monte Carlo (Hastings 1970) style rejection scheme:

$$P_m = min\Big(1, exp\Big(\frac{\Delta E_m}{T}\Big)\Big), \tag{2}$$

where P_m is the probability to accept the modification m, ΔE_m is the change on deck ratings with the modification, and T is a parameter called temperature to control the exploration versus exploitation balance.

The three steps above form an evolution round. The evolution process is run over a fixed number of rounds. As the evolution progresses, the process moves from from global exploration to local refinement over time: smaller modifications are preferred in step 1; more repeated matches are tested in step 2; and the parameter T is decreased in step 3 (as in simulated-annealing). The whole evolution process is initiated by randomly sampling a set of decks from the card pool of generated cards as the initial deck pool. After the target number of rounds has been reached, an additional batch of pair-wise matches are simulated between decks in the final deck pool as a final refinement. Card strengths and deck strengths are updated throughout the whole process to reflect their evaluations in the context of the actively evolved environment, which converges to a meta towards the end.

Deep Learning Model for Card Strengths

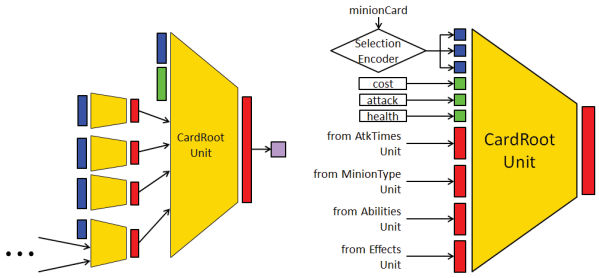
Although automatic simulation of games is much faster than actual players playing games, the performance cost is still significant for evaluating card strengths. Therefore, we wish

to explore the option of using a deep learning model to provide quick estimates on card strengths.

Recursive Card Strength Predictor

There are two important technical challenges in designing neural network models for card strength predictors. First, each instance from the grammar (i.e., each card) is of a different size, so that a fixed input network model cannot be used. Second, grammars have recursive, hierarchical structures which should ideally be reflected in the predictions. For these reasons, we adopted a network structure based on recursive neural networks (Socher et al. 2011). Recursive neural networks can be considered as a generalization of recurrent neural networks (RNNs) that support not only recursion, but also branching structures.

Our network architecture design is illustrated in Figure 2. We conceptualize the strength predictor as a series of information compression operations finally resulting in a single number. We build the recursive neural network with basic building blocks called nonterminal units. Each nonterminal unit encodes the compression of information when passing through the rules expanding a type of nonterminal, but in the reverse direction (i.e. merging the info from its children as the encoding for the parent node). The input to a nonterminal unit includes the following (when applicable): the choice of rule for the expansion (one-hot encoded), the terminal information (normalized, e.g. the mana cost mapped linearly to $[-1, 1]$), and the outputs from nonterminal units, depending on the specification in the grammar (see Figure 2b for an example). Different instances of rule application through the same nonterminal type share weights. The implementation of a nonterminal unit is generally a fully connected network layer with a sine activation. The output from the final *CardRoot* unit is followed by three fully connected network layers (not shown in the figure) with first two using sine activations and the final one using a sigmoid to produce the prediction of card strengths in the range of $(0, 1)$.



(a) Card Strength Predictor (b) CardRoot Unit

Figure 2: Recursive neural network card strength predictor architecture. Nonterminal units (yellow) compute output vector (red) based on rule selections (blue inputs), terminal statistics such as cost (green inputs), and/or outputs from other nonterminal units. (a) The overall card strength predictor. The input recursively passes through the tree structure of nonterminal units corresponding to that of the card, and the output of the root unit derives the final network output (purple). (b) The CardRoot unit, an example nonterminal unit.

Experiments and Discussions

To show our framework's capability of understanding generated cards, we perform experiments in the following two main parts. First, we examine the meta-based card strength evaluation method. Second, we test the neural network method using the evaluations from the first part as the training data. Each experiment is repeated with four independent runs, and any statistic shown is averaged across the runs unless specified. Experiments on this paper are executed on a machine with a 2.3 GHz Intel Xeon CPU and a 64 GB RAM. The run times for card generation ($< 10\mu s$) is trivial.

Meta-based Card Strength Evaluation

We use the evolutionary method mentioned in the Card Strength Evaluation section to evaluate cards strengths. Each experiment starts with a randomized card pool of 1000 cards and 30 initial random active decks and evolves a meta environment with 1000 evolution rounds. In order to verify the importance of a meta environment in evaluating cards, we also create a baseline environment with decks created and matched in a purely random way (called the random environment) for comparison. The random environment starts with the same card pool for each independent run and randomly creates 3000 decks and performs matches between randomly selected decks. The number of matches is around 180k ~ 190k (not fixed due to stochastic choice of modifications) in the meta environment and 200k in the random environment. Each match on average takes around 0.05 ~ 0.1 seconds depending on match lengths in the environment. The validity of the evolution is confirmed via simulated matches between the final deck pool of the meta environment and the top rated decks in the random environment (which shows an overwhelming win for meta environment; see Additional Materials); we expect the competitive environment to provide more faithful card evaluations than the random one.

Figure 3 shows the top cards rated by the meta environment and their ranks on both the meta and the random environment. Both environments agree on the #1 rated card (L1), which is a overpowered leader card. As playing a leader card in Chaos Cards replaces leader with the card (including the stats) and trigger its battlecries if any, the card is strong from multiple aspects, buffing/healing leaders, board clearing etc. More interestingly though, is the #2 rated card in the meta environment (S1), which draws cards with extreme cost efficiency. The random environment vastly underestimates it because of not having competitive decks of cards for the card drawing effect to synergize with. In fact, *Shadowverse* has the exact same card (*Inferno Surge*) that is only available as a token card spawned by playing a high cost legendary card.

Card Strength Predictor Training

We use the recursive neural network model mentioned in the Deep Learning Model section to train a card strength predictor. The model contains around 40,000 parameters, and are trained for 50 epochs, with a 800 to 200 training versus validation split on the card strength evaluation data from the

(a) Meta: #1; Random: #1 (b) Meta: #2; Random: #206

Figure 3: Cards rated top-2 in the meta environment simulation in one independent run, with their ranks (out of 1000) in both the meta and the random environment shown.

(a) Rank: #34 (b) Rank: #277

Figure 4: Ranks (out of 1000) of the predicted strengths for two of the zero cost spells in the test set.

first part of experiment in the meta version of environment. The loss function is log-cosh (a blend of MSE and MAE) weighted by the participation (defined in the Card Strength Evaluation Metric section) of the card. These weights are applied, because we value more on the prediction of important cards, which are tested more and with less uncertainly, and because the weighted average of the labels is an invariant of 0.5. Dropouts are adopted to mitigate over-fitting. Each training epoch takes less than 3s. Two statistics besides the loss function are examined on the validation sets, the root mean squared error and the linear correlation coefficient. We consider it is important to also incorporate the weight (participation) for those statistic for similar reasons in choosing the loss function. In fact, the results in Table 1 show that while the statistics without weighting are underwhelming, we can achieve a small amount of error and largely positive correlation with weighting, which means the predictor performs well on important cards in the meta environment.

To justify the use of the recursive network network, we has done a comparison against sequence models including RNN, GRU and LSTM with approximately the same number of parameters. The comparison shows our recursive network network learns faster and gives a lower validation loss.

Novel Card Generation. The trained network can be used to estimate strengths of new cards instantly without the need for the slow simulation-based evaluation process. For example, we can use GIGL to generate 1000 random cards (as the test set) and evaluate all 1000 using the network in less than 5 seconds. Figure 4 shows the prediction ranks for two of the zero cost spells. The network successfully learns dealing damage to enemies is better than doing so to allies (the former one is generally beneficial while the latter one is only useful in special cases like triggering deathrattles).

	Non-weighted	Weighted
RMSE	0.150	0.052
Correlation (r)	0.152	0.770

Table 1: The non-weighted and weighted versions of root mean squared error (RMSE) and linear correlation coefficient (r) on the validation sets.

Conclusion and Future Work

While our approach to PCG with Chaos Cards has demonstrated the ability to procedurally generate executable cards with a variety of interesting effects and provides ways for evaluating generated cards, there are still important limitations to our approach. First, the card generation is constrained by the authoring of the grammar, including what aspects of the card are created, and what effects are possible and how they can combine together. Additionally, the deep learning model performs less well on weak cards, mainly due to the complexity of cards and insufficient data.

Looking into the future, we would like to explore learning aspects of cards that are difficult to be authored into formal grammars, such as card names. Further, we would like to consider creating not just individual cards, but rather decks or sets of cards containing interesting interactions, to further refine our problem space. Doing so would be a key step towards understanding game balance in a more holistic fashion. Finally, we are interested in understanding other more human oriented aspects of PCG applied to Chaos Cards such as how fun or engaging the generated cards might be to play.

Acknowledgments and Additional Materials

We would like to acknowledge Zachary Chavis for artistic contributions on cards shown in this paper.

We refer readers to tinyurl.com/chaoscards-grammar for the full card grammar, and to tinyurl.com/chaoscards-supmat for results not included in the paper, including more card evaluation/prediction examples, deck match tests for validating the evolution, neural network comparisons etc.

References

Ammanabrolu, P.; Tien, E.; Cheung, W.; Luo, Z.; Ma, W.; Martin, L.; and Riedl, M. 2019. Guided neural language generation for automated storytelling. In *Proceedings of the Second Workshop on Storytelling*, 46–55.

Chaslot, G.; Bakkes, S.; Szita, I.; and Spronck, P. 2008. Monte-carlo tree search: A new framework for game ai. In *AIIDE*.

Chen, T., and Guy, S. J. 2018. GIGL: A domain specific language for procedural content generation with grammatical representations. In *Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Claessen, K., and Hughes, J. 2011. Quickcheck: a lightweight tool for random testing of haskell programs. *Acm sigplan notices* 46(4):53–64.

Compton, K.; Filstrup, B.; and Mateas, M. 2014. Tracery: Approachable story grammar authoring for casual users. In *Seventh Intelligent Narrative Technologies Workshop*.

de Mesentier Silva, F.; Canaan, R.; Lee, S.; Fontaine, M. C.; Togelius, J.; and Hoover, A. K. 2019. Evolving the hearthstone meta. In *2019 IEEE Conference on Games (CoG)*, 1–8. IEEE.

Dong, J.; Wang, L.; Liu, J.; Gao, Y.; Qi, L.; and Sun, X. 2019. A procedural texture generation framework based on semantic descriptions. *Knowledge-Based Systems* 163:898–906.

Emilien, A.; Bernhardt, A.; Peytavie, A.; Cani, M.-P.; and Galin, E. 2012. Procedural generation of villages on arbitrary terrains. *The Visual Computer* 28(6-8):809–818.

García-Sánchez, P.; Tonda, A.; Squillero, G.; Mora, A.; and Merelo, J. J. 2016. Evolutionary deckbuilding in hearthstone. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, 1–8. IEEE.

Germer, T., and Schwarz, M. 2009. Procedural arrangement of furniture for real-time walkthroughs. In *Computer Graphics Forum*, volume 28, 2068–2078. Wiley Online Library.

Grad, Ł. 2017. Helping ai to play hearthstone using neural networks. In *2017 federated conference on computer science and information systems (FedCSIS)*, 131–134. IEEE.

Hastings, W. K. 1970. Monte carlo sampling methods using markov chains and their applications. *Biometrika* 57(1):97–109.

Hoover, A. K.; Togelius, J.; Lee, S.; and de Mesentier Silva, F. 2020. The many ai challenges of hearthstone. *KI-Künstliche Intelligenz* 34(1):33–43.

Horswill, I. D. 2018. Catsat: A practical, embedded, sat language for runtime pcg. In *Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Jin, Y. 2019. Proposed balance model for card deck measurement in hearthstone. *The Computer Games Journal* 8(1):25–40.

Kempen, G., and Hoenkamp, E. 1987. An incremental procedural grammar for sentence formulation. *Cognitive science* 11(2):201–258.

Khalifa, A., and Togelius, J. 2020. Multi-objective level generator generation with marahel. *arXiv preprint arXiv:2005.08368*.

Krecklau, L., and Kobbelt, L. 2012. Interactive modeling by procedural high-level primitives. *Computers & Graphics* 36(5):376–386.

Ling, W.; Grefenstette, E.; Hermann, K. M.; Kočiskỳ, T.; Senior, A.; Wang, F.; and Blunsom, P. 2016. Latent predictor networks for code generation. *arXiv preprint arXiv:1603.06744*.

Martens, C. 2015. Ceptre: A language for modeling generative interactive systems. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*.

Merrell, P., and Manocha, D. 2008. Continuous model synthesis. In *ACM transactions on graphics (TOG)*, volume 27, 158. ACM.

Merrick, K. E.; Isaacs, A.; Barlow, M.; and Gu, N. 2013. A shape grammar approach to computational creativity and procedural content generation in massively multiplayer online role playing games. *Entertainment Computing* 4(2):115–130.

Osborn, J. C.; Lambrigger, B.; and Mateas, M. 2017. Hyped: Modeling and analyzing action games as hybrid systems. In *Thirteenth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Parish, Y. I., and Müller, P. 2001. Procedural modeling of cities. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 301–308. ACM.

Ryan, J.; Seither, E.; Mateas, M.; and Wardrip-Fruin, N. 2016. Expressionist: An authoring tool for in-game text generation. In *International Conference on Interactive Digital Storytelling*, 221–233. Springer.

Shaker, N.; Nicolau, M.; Yannakakis, G. N.; Togelius, J.; and O'neill, M. 2012. Evolving levels for super mario bros using grammatical evolution. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, 304–311. IEEE.

Smith, G.; Treanor, M.; Whitehead, J.; and Mateas, M. 2009. Rhythm-based level generation for 2d platformers. In *Proceedings of the 4th International Conference on Foundations of Digital Games*, 175–182. ACM.

Socher, R.; Lin, C. C.; Manning, C.; and Ng, A. Y. 2011. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, 129–136.

Świechowski, M.; Tajmajer, T.; and Janusz, A. 2018. Improving hearthstone ai by combining mcts and supervised learning algorithms. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, 1–8. IEEE.

Talton, J. O.; Lou, Y.; Lesser, S.; Duke, J.; Měch, R.; and Koltun, V. 2011. Metropolis procedural modeling. *ACM Transactions on Graphics (TOG)* 30(2):11.

Togelius, J.; Kastbjerg, E.; Schedl, D.; and Yannakakis, G. N. 2011. What is procedural content generation?: Mario on the borderline. In *Proceedings of the 2nd international workshop on procedural content generation in games*, 3. ACM.

Tomás, A. P., and Leal, J. P. 2013. Automatic generation and delivery of multiple-choice math quizzes. In *International Conference on Principles and Practice of Constraint Programming*, 848–863. Springer.

Toto, F. S. G., and Vessio, G. 2014. A probabilistic grammar for procedural content generation. In *Sixth Workshop on Non-Classical Models of Automata and Applications (NCMA 2014)*, 31.

Yannakakis, G. N.; Liapis, A.; and Alexopoulos, C. 2014. Mixed-initiative co-creativity.

Zook, A.; Harrison, B.; and Riedl, M. O. 2019. Monte-carlo tree search for simulation-based strategy analysis. *arXiv preprint arXiv:1908.01423*.