Proceedings, The Twelfth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-16)

Mystical Tutor: A Magic: The Gathering Design Assistant via Denoising Sequence-to-Sequence Learning

Adam James Summerville and **Michael Mateas**
Expressive Intelligence Studio
Center for Games and Playable Media
University of California, Santa Cruz
asummerv@ucsc.edu , michaelm@soe.ucsc.edu

Abstract

Procedural Content Generation (PCG) has seen heavy focus on the generation of levels for video games, aesthetic content, and on rule creation, but has seen little use in other domains. Recently, the ready availability of Long Short Term Memory Recurrent Neural Networks (LSTM RNNs) has seen a rise in text based procedural generation, including card designs for Collectible Card Games (CCGs) like *Hearthstone* or *Magic: The Gathering*. In this work we present a mixed-initiative design tool, *Mystical Tutor*, that allows a user to type in a partial specification for a card and receive a full card design. This is achieved by using sequence-to-sequence learning as a denoising sequence autoencoder, allowing *Mystical Tutor* to learn how to translate from partial specifications to full.

Introduction

Procedural Content Generation (PCG) in the context of video games has most commonly focused on level generation or decorative content. Level generation has seen a wide number of approaches from the earliest days of PCG starting with *Beneath Apple Manor* in 1978 and being popularized by *Rogue* in 1980. Decorative content in games has largely come from non-critical components such as texture generation (ALL 2016) or tree generation (SPE 2016). The procedural creation of content other than these has been limited. There have been multiple attempts to generate games (Nelson and Mateas 2007)(Browne and Maire 2010)(Treanor et al. 2012)(Cook and Colton 2014)(Zook and Riedl 2014)(Lim and Harrell 2014), but this has largely been focused on the creation of game rules or mechanics. Perhaps the best example of non-level based PCG in videogames is the user guided evolution of weapons in *Galactic Arms Race* (Hastings, Guha, and Stanley 2009).

Text generation in the context of games has mostly been focused on decorating templates with variations authored by a user a la the work of Compton et al. (Compton, Filstrup, and Mateas 2014). Outside of games, text generation has seen a number of approaches, perhaps most simply with Markov chains. The Markov chain is typically learned from a corpus that learns transition probabilities from word-to-word and then generates chains following these probabilities. A key problem with Markov chain generation is that they have good local coherence (word-to-word probabilities) but tend to have poor global coherence (sentences make no sense). This can be remedied by increasing the historicity of the chain, but this has two main problems:

- The amount of data needed to learn all possible transitions increases exponentially with the length of the history
- The ability to generalize decreases due to learning more and more specific transitions as the transition matrix becomes sparser and sparser.

Long Short Term Memory (LSTM) Recurrent Neural Networks (RNNs) represent the state of the art for sequence learning. They can train with history sizes well beyond what would be feasible in a Markov chain (on the order of tens or hundreds of items instead of twos or threes). They can also generalize and learn higher order rules that are not capable of being learned by the local probabilities of a Markov chain. They have been shown to be able to generate valid C code (or at least syntactically correct) when properly trained (Karpathy 2015). Recent work has focused on Sequence-To-Sequence learning which can "translate" from arbitary sequence to arbitrary sequence. This technique has even been used to "compile" simple Python code, producing program output given the source code of a python program and the program's input, though this has been limited to basic arithmetic and simple for/while loops (Zaremba and Sutskever 2014).

LSTMs have also been used to generate cards for *Magic: The Gathering* (Billzorn 2016). This code even provides the underpinnings of a twitter bot entitled "RoboRosewater" (so named because of Magic's lead designer Mark Rosewater) that generates a new card every day (Milewicz 2016). While it is fun to see the output of the generator, it is not capable of working as a mixed-initative design tool, as it simply generates a sequence of characters one at a time. There are only two possible ways a user could provide input to the generator: **1)** The user has to specify all fields prior to the field they want to set (e.g. if the user wished to set the cost of the card, which is the last field in RoboRosewater's ordering, they would have to fill in all the other fields leaving the system nothing to generate) or **2)** Allow the system to gen-

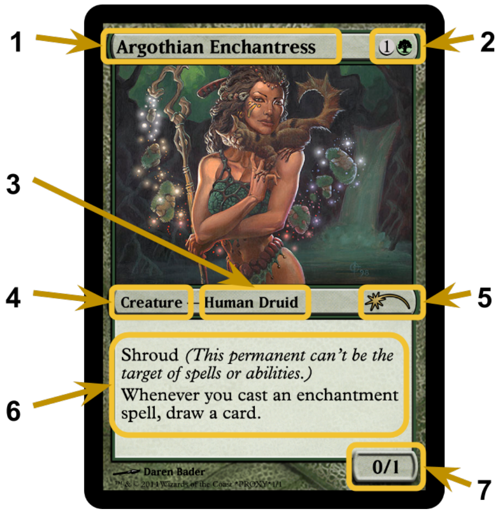
1
2
3
4
5
6
7
Argothian Enchantress
Creature — Human Druid
Shroud (This permanent can't be the target of spells or abilities.)
Whenever you cast an enchantment spell, draw a card.
0/1
Daren Bader

Figure 1: A card from *Magic: The Gathering*. 1) **Name**, 2) **Cost**, 3) **Sub Type**, 4) **Type**, 5) **Set** (shape of the symbol) and **Rarity** (color of the symbol), 6) **Card Text** 7) **Power/Toughness**. **Super Type** would appear to the left of (4) if it were present and **Loyalty** would appear in place of (7) if the card were a Planeswalker.

erate everything other than what they specify, creating the problem that fields which occur earlier in the sequence have no information about later fields (e.g. setting a card cost to be 3 colorless mana, the system does not know this until the end - meaning it very likely will not generate a card for which that cost makes sense). In this work, we obviate the need for these workarounds through the use of Sequence-to-Sequence learning with attention.

A breakdown of a card can be seen in figure 1. Cards can be some number of 8 different types (Land, Creature, Artifact, Enchantment, Sorcery, Instant, Planeswalker, Tribal), although only certain types, such as Artifact and Enchantment, support mixtures of types (e.g. Artifact-Land or Enchantment-Creature). There are also 5 different colors in *Magic* of which a card be any combination (or even have the lack of a color). There are also other, higher-order attributes that a designer might care about, such as whether the card is a certain color (without needing to specify a complete mana cost) or the converted mana cost (a card that costs 5 colorless mana and a card that costs 1 white, 1 red, 1 blue, 1 green, and 1 black both have a converted mana cost of 5). Given all of these meaningful aspects to cards, a designer might wish to be able to specify specific portions of a card, say the type and cost, and let the system fill in the rest.

In this work we introduce *Mystical Tutor* (named after a powerful card that allows players to search their deck for an instant or sorcery card), a mixed-initiative Magic The Gathering card generator that can take a partial card specification as input and generate a valid card. A user can input as much or as little as they want and *Mystical Tutor* will work with those soft constraints to generate a card. *Mystical Tutor* uses a sequence-to-sequence denoising decoder-encoder LSTM to learn how all of the parameters of a card work in conversation with each other as opposed to just learning linear traces through the text of a card. The contribution of this work is a novel denoising sequence-to-sequence neural network as well a novel use of machine learning as a mixed iniative design tool.

Related Work

As mentioned above, the state-of-the-art for machine learned Magic card generation is *RoboRosewater*. *RoboRosewater* uses an LSTM trained at the character level to generate card strings character by character. Figure 2 has 4 examples of cards generated by *RoboRosewater*. The same approach has also been used for the game *Hearthstone* (scfdivine 2015). These systems are good at producing random cards that are generally coherent and interesting, but have no way for a user to specify specific sub-pieces since they generate in a lock-step one character at a time. Any information that a user specifies must come at the beginning of the sequence and there can be no gaps in the information that they specify. LSTMs have also seen use for generating other game content, namely Mario levels in the work by Summerville and Mateas (Summerville and Mateas 2016).

Sequence-to-Sequence learning is a statistical machine translation technique that generates output sequences as a result of input sequences. It has been used to translate from one language to another (Sutskever, Vinyals, and Le 2014), from source code to the output of the code, (Zaremba and Sutskever 2014), or from English sentences to grammatical parsings (Vinyals et al. 2014). They operate by training LSTMs on input sequences and use the hidden state of the encoding neural network as the input to a decoding neural network which then generates the output sequence. Recent work by Ling et al. (Ling et al. 2016) used Magic: The Gathering and Hearthstone cards as the source trace and then learned to generate code for open source implementations of the games given a card specification via Latent Predictor Networks that utilize sequence-to-sequence learning as well as pointer networks to copy from the input to the output (e.g. for instance, copying the name or cost of the card).

Denoising autoencoders are a neural network technique that tries to replicate the input as the output. However, instead of just exactly duplicating the input, noise is added to corrupt the input, typically by setting input fields to 0 at random. This forces the system to learn how different pieces of the input interact to be able to infer the corrupted pieces of the input.

Machine learning has seen little use in mixed-initiative design tools. Bruni, et al. (Bruni et al. 2011) used Bayesian inference to learn how a user goes about planning a UAV mission, and uses this to support future mission planning work.Xia et al. built a system that uses a Bayesian network to learn how to cooperate with a human user during music performance (Xia et al. 2015). Both of these systems learned how people act creatively by observing them in the creative process. In contrast, work by Hoover et al. (Hoover, Togelius, and Yannakakis 2015) only makes use of human output rather than information about the creation process. In this work, they trained a neural network to predict the lowest y-value for specific "voices" of tile types in *Super Mario*

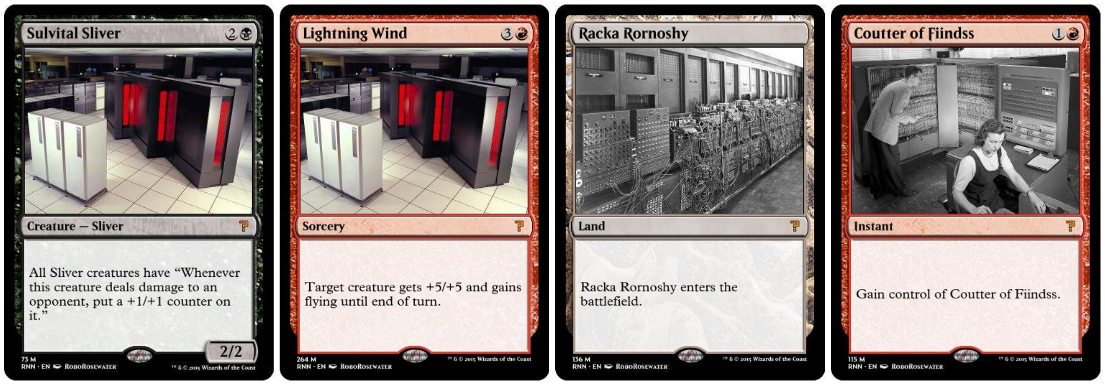
Sulvital Sliver
Creature — Sliver
All Sliver creatures have "Whenever this creature deals damage to an opponent, put a +1/+1 counter on it."
2/2
Lightning Wind
Sorcery
Target creature gets +5/+5 and gains flying until end of turn.
Racka Rornoshy
Land
Racka Rornoshy enters the battlefield.
Coutter of Fiindss
Instant
Gain control of Coutter of Fiindss.

Figure 2: Cards generated by @RoboRosewater. The left two are are good examples of evocative, sensical output, while the right two make sense grammatically but are nonsense from a gameplay perspective.

Bros. (e.g. ground, ?-block, pipe, etc.) based on the other tiles in the same column. This supports a user in drawing in a portion of the level and having the system fill in the rest.

Design Support as Translation

The core idea of using sequence-to-sequence generation is the novel architecture of denoising LSTM autoencoders. Autoencoders are common in the realm of artificical neural networks where they most commonly act as a form of feature compressor by which a given input is fed to a network that then squishes down to a smaller hidden layer (or multiple hidden layers) than the input layer which is then fed to an output layer which tries to recreate the input. A subclass of these are known as denoising autoencoders. A denoising autoencoder operates on "noised" or "corrupted" input (typically some percentage of the input is set to 0 although it could be replaced with a random value) and then tries to reconstruct the original "non-corrupted" input. This forces the neural network to learn not just a discriminative algorithm but rather a generative one. That is, if we have two random variables X and Y, a discriminative algorithm would try to learn:

$P(Y|X)$

where a generative algorithm would try to learn:

$P(Y, X)$

By learning a joint rather than a conditional probability distribution, the generative algorithm learns the relationships between all variables as opposed to just classifying one subset based on the others.

As previously mentioned, LSTMs represent the current state of the art for sequence learning. However, they typically operate under the assumption that the input is the output, not a corrupted representation of the output. We can use the lens of "translation" if we see the approach used to take an input string in the "corrupted" language and generate a string in the "non-corrupted" language and as such act as a denoising LSTM. By randomly permuting and corrupting data, the hope would be that *Mystical Tutor* would learn to generalize from snippets of sequences to full sequences, in this case from partial specifications to full cards. Where RoboRosewater operates on a character-level predicting the next character in a string based on the history, our work processes the entire input string at once and then produces an output via the attentional Sequence-to-Sequence architecture. The contribution of the denoising aspect allows *Mystical Tutor* to seamlessly handle partial input with no regard for ordering of the fields, unlike RoboRosewater.

Data Specification

We used the full set of *Magic: The Gathering* cards, 13,651 cards in total as our data set. The set was duplicated 30 times with each duplication shuffled. Each card had $\frac{1}{3}$ of its components randomly corrupted, which means that each component in a card is expected to be seen 20 times. The simplest way of handling the corruption would be applying dropout on the input layer of the network; However,this was not used as the corruption was wanted at the component level, not the word level. The goal for our denosing task was to handle partial specifications, but it is unlikely that users will specify partial strings. Returning to the Argothian Enchantress example, the word level corruption might look like "Creature – Human Druid – you an spell card" whereas the component level might be "Whenever you cast an enchantment spell, draw a card". The former is unlikely to be entered by a user, while the latter is highly likely. As mentioned above, there are many different components to a card, of which we considered:

- **Cost** - The mana cost of a card. There are 5 different colors, colorless, generic, and "hybrid" mana (half one color, half another) that can be combined in countless ways
- **Type** - Creature, Artifact, Enchantment, Sorcery, Instant, Land, Planeswalker, and Tribal - but again these can be combined (e.g. Artifact Land, Enchantment Creature)
- **Sub Type** - Only applicable to certain card types, but most commonly creatures which are generally a race (e.g. human, elf, goblin, etc.) and a class (e.g. soldier, shaman, warrior, etc.)
- **Super Type** - Modifiers of type with specific rule effects, such as *Legendary* which means that there can only be 1 instance of that card on the battlefield for a given player

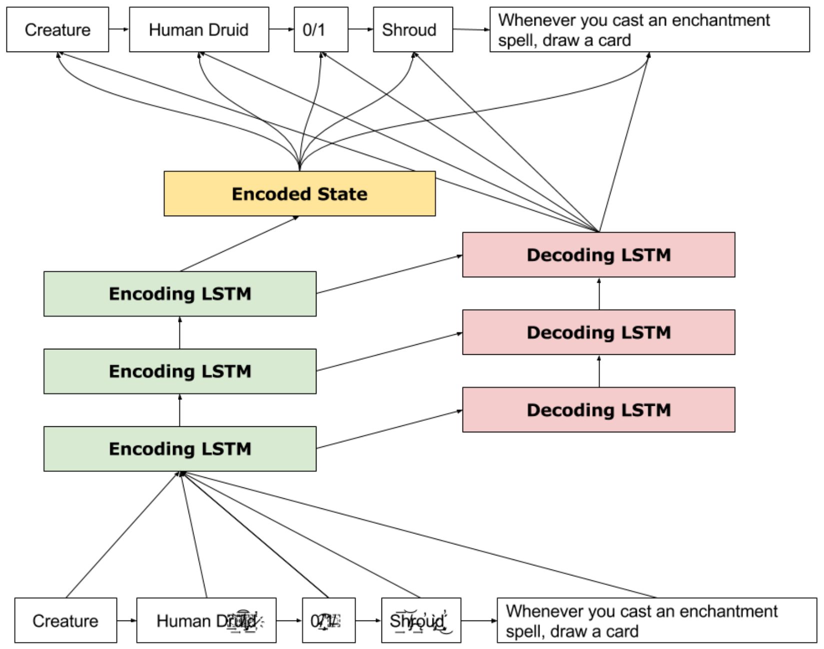

Figure 3: Demonstration of *Mystical Tutor*. Each LSTM layer consists of 256 LSTM cells. The input would not actually be corrupted on a word-to-word basis but instead fields of the card are corrupted. Encoding and decoding happen at the word level.

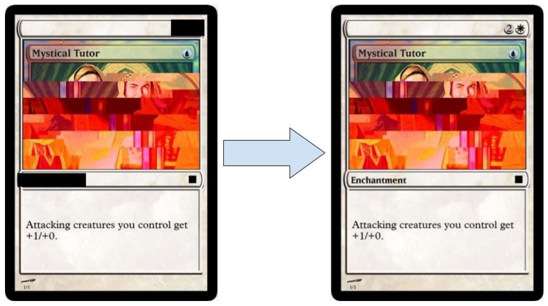
Mystical Tutor
Attacking creatures you control get +1/+0.
Mystical Tutor
Enchantment
Attacking creatures you control get +1/+0.

Figure 4: This card seems entirely plausible. The ability tends towards red, but enchantments that pump your entire team (commonly referred to as *anthems*) are mostly white enchantments. This is probably overcosted by 1 mana, but is a plausible uncommon.

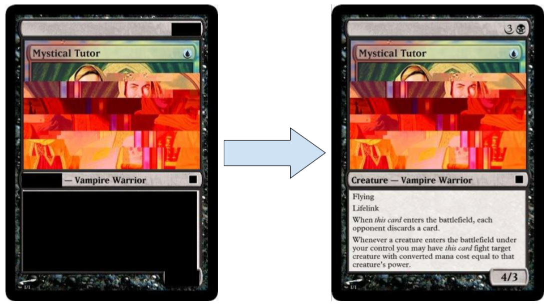
Mystical Tutor
— Vampire Warrior
Mystical Tutor
Creature — Vampire Warrior
Flying
Lifelink
When this card enters the battlefield, each opponent discards a card.
Whenever a creature enters the battlefield under your control you may have this card fight target creature with converted mana cost equal to that creature's power.
4/3

Figure 5: There are a lot of moving parts on this card and in all likelihood would probably have 1 or 2 abilities stripped from it. Flying and Lifelink (which causes a player to gain life when the creature deals damage) are very common in Vampires. Discarding a card less so, but is a black ability. The fighting clause is an oddity that exists nowhere else in the game, but with the proper flavor (a vampire lord that fights enemies based on her minions' strength, perhaps) could work.

- **Power** - A number denoting the offensive capability of a creature
- **Toughness** - Conversely, the defensive capability
- **Card Text** - The core of the game. *Magic* is a game of rules and exceptions. The exceptions come from the card text. This can be simple such as *Flying* (can't be blocked by creatures without flying) or extremely complex modifications upwards of 90 words and 11 lines of text.
- **Loyalty** - A special resource for Planeswalker cards, the rarest, most tactically dense card type

Cards are treated as a series of tokens, with each word on the card being a token along with the special tokens: Start of Card, End of Card, Section Delimiter, Section ID, Card Text New Line, Start and End of Mana Cost, Start of Power/Toughness or Power/Toughness modifier (e.g. -3/+3).

Numbers, whether they are generic mana costs, power, toughness, or power or toughness modifiers, are handled as strings of tokens with length equal to the number e.g. the number 3 is denoted as %%%. Numbers which are represented as English text are left alone. A few special numbers are also treated as special tokens, namely 10, 15, and 20. This brings the total size of the vocabulary up to 637 distinct

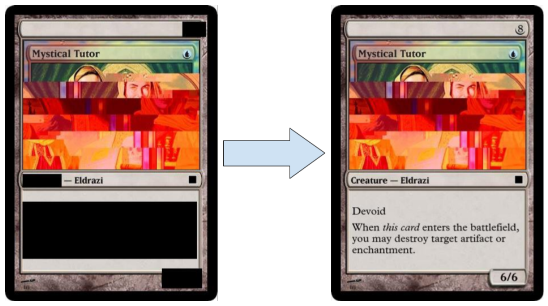
Mystical Tutor
— Eldrazi
Mystical Tutor
Creature — Eldrazi
Devoid
When this card enters the battlefield, you may destroy target artifact or enchantment.
6/6

Figure 6: An entirely plausible card. The only real detraction is the *Devoid*, which is an ability that states that the creature is colorless. In this case it is redundant.

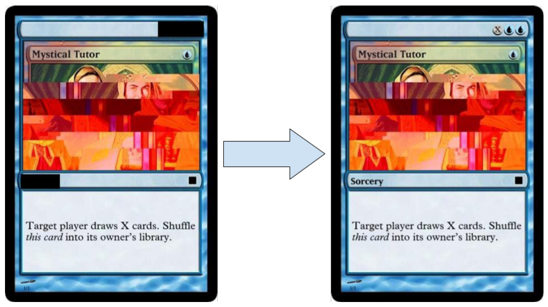
Mystical Tutor
Target player draws X cards. Shuffle this card into its owner's library.
Mystical Tutor
Sorcery
Target player draws X cards. Shuffle this card into its owner's library.

Figure 7: This card text exists as an instant that costs 3 blue mana and X generic mana, so it is interesting that *Mystical Tutor* did not copy that card and instead discounted it a mana by making it less versatile timing wise (shifting from instant to sorcery).

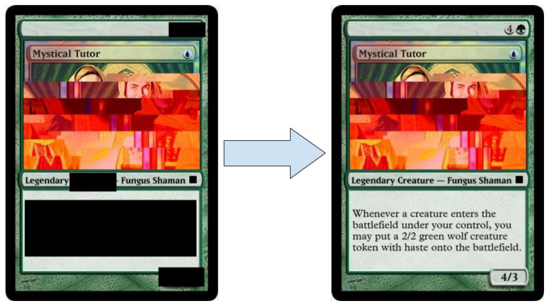
Mystical Tutor
Legendary
Fungus Shaman
Mystical Tutor
Legendary Creature — Fungus Shaman
Whenever a creature enters the battlefield under your control, you may put a 2/2 green wolf creature token with haste onto the battlefield.
4/3

Figure 8: The wolves should probably be replaced by Saproling (a fungus creature in Magic) tokens. The biggest problem is triggering on token creatures causes this to create an unlimited number of creatures ready to attack (since they have haste).

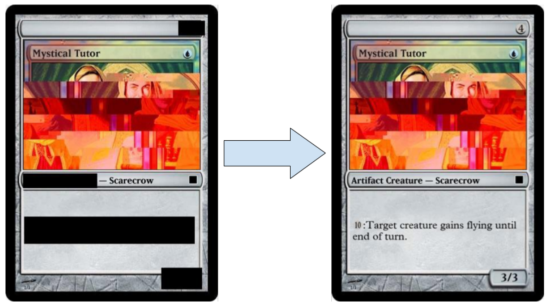
Mystical Tutor
— Scarecrow
Mystical Tutor
Artifact Creature — Scarecrow
10 :Target creature gains flying until end of turn.
3/3

Figure 9: A four mana 3/3 is a workhorse creature that balances limited formats (limited is a format where players are given random packs of cards and they must form decks on the spot). The ability costs a lot of mana, but the reasonable body + ability that can potentially break stalls is a common trope often referred to as "invokers".

tokens.

Two variants of *Mystical Tutor* were trained, the first of which left corrupt information as missing. The second added another token to the encoding vocabulary which acted as a denotation that a section had been corrupted and should normally be present. The intention of the second variant is to act as an indicator to the decoder that a specific section needs to be filled in as opposed to naturally not being present (e.g. A Planeswalker has no power and toughness so that section would never be present, but a Planeswalker missing the mana cost section would be flagged as it should be present).

Each variant consisted of an encoding LSTM and decoding LSTM each of which consisted of 3 layers each with 256 LSTM cells. The encoder operates by learning the input sequence and encoding it to a state vector. The decoder then generates the output sequence using that state vector as well as by using an attentional system allowing it to attend to the encoded state vector. Without attentional system the encoded state vector would simply be used as the starting seed of the decoding sequence, but the attentional system allows the system to look back and move its focus to different parts of the input. This is a key advantage over RoboRosewater, as this allows the system to look at portions of the input as they are relevant in the decoding process (e.g. While decoding the Mana Cost, it can look back at the specific portion of the input that contained the Mana Cost, if present). The architecture can be seen in figure 3. Decoding takes place by greedily taking the tokens with the maximum likelihood.

Evaluation

There exist two evaluations for this work, *Accuracy* and *Aesthetics*. *Accuracy* measures how well *Mystical Tutor* performs the sequence-to-sequence task on unseen data. *Aesthetics* is a more challenging evaluation, as it requires human judgment of expert Magic players, but there are three major criterion that *Mystical Tutor* should be graded on:

- Coherence - How well does this hold up as *Magic* card? Did it do obviously ludicrous things (*Flying* to a Sorcery card) or did it do something that would be allowed by game rules but not by the design rules (e.g. making a traditionally Black creature, a Zombie, White)?
- Novelty - Was the generated card a previously unseen card, or was it simply a previous card regurgitated verbatim? Was the card interesting in an unexpected way?
- Utility - How useful was the generated card for the user? This is perhaps the most difficult to assess as different cards can fulfill different roles, and the usefulness of the

generated card is very contextual (e.g. A user might want to use the generator to try to determine the cost of a mostly designed card or they might want to roll the die and give minimal input).

Method	Training Perplexity	Test Perplexity
Standard	1.07	1.21
Missing Tokens	**1.06**	**1.08**

Table 1: Perplexity measures for both generators for training and test sets.

The accuracy results can be seen in table 1. *Mystical Tutor* was able to achieve a Perplexity of **1.06** on the training data and a Perplexity of **1.08** on held out test data. This test data is still derived from the original set of Magic cards that are again randomly corrupted. Of the corrupted dataset that was previously described there was a 90% - 10% training - test split. Perplexity is a measure most commonly used in natural language processing that measures roughly how many choices a probability distribution has to choose between, e.g. a fair 6-sided die has perplexity of 6, a fair coin a perplexity of 2, and perfect knowledge a perplexity of 1. A perplexity of **1.08** means that it is almost perfectly able to recreate the correct output from the input. The two variants produced training perplexity scores that were very close, but the variant with missing tokens produced a better perplexity on the test set (**1.08** vs **1.21**), so the remaining work showcases the work of the missing token variant.

Due to this high level of accuracy, the coherence of the cards tends to be very high, qualitatively. All of the cards that have been observed were grammatically sound, although there are the occasional ludological hiccups. Figures 4 through 9 represent 6 interactions with *Mystical Tutor*. On the left in each figure is the input that was given to *Mystical Tutor*, and the right represents the output of *Mystical Tutor*. The inputs were chosen to demonstrate *Mystical Tutor* working across different colors, card types, and types of input:

- Figure 4 - This effect, increasing the power of all attacking creatures, could be seen on a number of different cards (e.g. a one time effect on an instant, a "leader" creature, an artifact, or an enchantment) as well as a number of different colors (Red, White, or Artifact).
- Figure 5 - Vampires are predominantly a black creature (although they have also shown up in Red and Blue as sets dictate) and Warriors are most commonly Red or Green showing up roughly half as often in Black, one quarter as often in White, and one tenth as often in Blue. We would expect this card to be black but were interested in seeing if the Warrior pushed it to Red.
- Figure 6 - Eldrazi are an oddity in Magic as they are colorless creatures. Prior to their introduction, the only way for a creature to be colorless was to also be an artifact. They are gigantic Lovecraftian horrors and tend to have large costs, effects, and sizes.
- Figure 7 - This effect has been seen on one card, a blue instant, but similar effects have shown up on a number of blue spells.
- Figure 8 - Legendary creatures represent singular entities and tend to have high power, flavorful abilities. There has only been 1 Legendary Fungus Shaman in Magic, and we wished to see if *Mystical Tutor* had simply memorized it. It did not, and instead produced an entirely novel card.
- Figure 9 - Scarecrows are only ever artifact creatures, tend to cost between 3 and 5 mana, and are usually smallish (3/3 or smaller). There are a few that are anti-flying, some that have abilities that cost colorless mana, and some that have special interactions with -1/-1 counters, but there is no strong cohesion between them as a whole. The generated card is well within the realm of a plausible Scarecrow.

Discussions of *Mystical Tutor*'s output can be seen below each input-output pair.

We also conducted a small user study with 4 designers interested in *Magic* card design pulled from visitors to a *Magic* design blog and from a group of MFA design students' weekly *Magic* group. The users were asked to provide 5 partial card specifications and then were given the results from *Mystical Tutor*, which they then rated on **Sensibleness**, **Novelty**, and **Usefulness** on a 5 point scale. The results can be seen in table 2.

Sensibleness	Novelty	Usefulness
3.70	3.45	3.65

Table 2: User ratings for cards generated by *Mystical Tutor*. All were significantly different from a rating of 3.0 with $p < 0.025$.

Conclusion and Future Work

In this paper we propose the metaphor of translation as a way for a mixed initiative design assistant to help a user design content. To support this, we have presented *Mystical Tutor*, a design assistant for *Magic: The Gathering* that allows a user to input a partial specification and get a full card as output. The core technical contribution is a novel denoising sequence-to-sequence LSTM RNN.

There are two core facets for future work. First, additional technical work. Currently, this work uses a greedy decoding scheme, but using beam search it may be possible to get multiple sequences of higher quality. Similarly, currently *Mystical Tutor* is deterministic at run-time, but a user might like to "re-roll" and see different outputs for the same input. This is possible as-is, but might result in nonsensical results if a low probability word is sampled during the middle of the output sequence. Instead, utilizing a quasi-randomized beam-search should allow for variation that is still robust. The work of Ling et al. (Ling et al. 2016) treated Magic cards as structured input instead of one text string. By treating each section of the card differently, *Mystical Tutor* should be more robust to missing input.

This work could also be easily applied to other structured, text-heavy inputs such as cards for Netrunner or Hearthstone. However, a key disadvantage to these is the much smaller dataset as Netrunner has 700 cards and Hearthstone has 900 which are over an order of magnitude smaller than the size of the Magic dataset.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. IIS-1409992.

References

2016. *Allegorithmic*. https://www.allegorithmic.com/.

Billzorn. 2016. *mtg-rnn*. https://github.com/billzorn/mtg-rnn.

Browne, C., and Maire, F. 2010. Evolutionary game design. *Computational Intelligence and AI in Games, IEEE Transactions on*.

Bruni, S.; Schurr, N.; Cooke, N.; Riordan, B.; and Freeman, J. 2011. Designing a mixed-initiative decision-support system for multi-uas mission planning. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*.

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

Cook, M., and Colton, S. 2014. Ludus ex machina: Building a 3d game designer that competes alongside humans. In *Proceedings of the Fifth International Conference on Computational Creativity*.

Hastings, E. J.; Guha, R. K.; and Stanley, K. O. 2009. Evolving content in the galactic arms race video game. In *Proceedings of the 5th International Conference on Computational Intelligence and Games*, CIG'09, 241–248. Piscataway, NJ, USA: IEEE Press.

Hoover, A. K.; Togelius, J.; and Yannakakis, G. N. 2015. Composing video game levels with music metaphors through functional scaffolding. In *Proceedings of the ICCC Workshop on Computational Creativity and Games*.

Karpathy, A. 2015. *The Unreasonable Effectivenss of Recurrent Neural Networks*. http://karpathy.github.io/2015/05/21/rnn-effectiveness/.

Lim, C.-U., and Harrell, D. 2014. An approach to general videogame evaluation and automatic generation using a description language. In *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*.

Ling, W.; Grefenstette, E.; Hermann, K. M.; Kocisky, T.; Senior, A.; Wang, F.; and Blunsom, P. 2016. Latent predictor networks for code generation.

Milewicz, M. 2016. *RoboRosewater*. https://twitter.com/roborosewater?lang=en.

Nelson, M. J., and Mateas, M. 2007. Towards automated game design. In *AI* IA 2007: Artificial Intelligence and Human-Oriented Computing*. Springer. 626–637.

scfdivine. 2015. *Hearthstone cards as created by a neural network*. https://www.reddit.com/r/ hearthstone/comments/3cyi15/hearthstone_cards_as_created_by_a_neural_network/.

2016. *Speed Tree*. http://www.speedtree.com/.

Summerville, A., and Mateas, M. 2016. Super mario as a string: Platformer level generation via lstms.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to sequence learning with neural networks. In *CoRR*.

Treanor, M.; Blackford, B.; Mateas, M.; and Bogost, I. 2012. Game-o-matic: Generating videogames that represent ideas. In *Procedural Content Generation Workshop at the Foundations of Digital Games Conference*.

Vinyals, O.; Kaiser, L.; Koo, T.; Petrov, S.; Sutskever, I.; and Hinton, G. E. 2014. Grammar as a foreign language. *CoRR*.

Xia, G.; Wang, Y.; Dannenberg, R. B.; and Gordon, G. 2015. Spectral learning for expressive interactive ensemble music performance. In *Proceedings of the 16th International Society for Music Information Retrieval Conference, ISMIR 2015*.

Zaremba, W., and Sutskever, I. 2014. Learning to execute. In *CoRR*.

Zook, A., and Riedl, M. 2014. Automatic game design via mechanic generation. In *AAAI Conference on Artificial Intelligence*.