Narrative Generation in the Wild: Methods from NaNoGenMo

Judith van Stegeren
Human Media Interaction
University of Twente
Enschede, The Netherlands
j.e.vanstegeren@utwente.nl

Mariët Theune
Human Media Interaction
University of Twente
Enschede, The Netherlands
m.theune@utwente.nl

Abstract

In text generation, generating long stories is still a challenge. Coherence tends to decrease rapidly as the output length increases. Especially for generated stories, coherence of the narrative is an important quality aspect of the output text. In this paper we examine how narrative coherence is attained in the submissions of NaNoGenMo 2018, an online text generation event where participants are challenged to generate a 50,000 word novel. We list the main approaches that were used to generate coherent narratives and link them to scientific literature. Finally, we give recommendations on when to use which approach.

1 Introduction

Coherence is generally considered to be a property of a good story. For a story to be coherent, "all the parts of the story must be structured so that the entire sequence of events is interrelated in a meaningful way" (Shapiro and Hudson, 1991), p.960. For generated stories, in particular those generated using neural models, coherence tends to decrease rapidly as the output length increases. For this reason, generating long stories is still a challenge (Kiddon et al., 2016).

To gain more insight in how generation of long stories is done 'in the wild', we review a collection of story generation projects that were created as part of the online challenge NaNoGenMo.

NaNoGenMo, or National Novel Generation Month[1], is a yearly online event that challenges participants to create a novel using text generation. Participants have one month (November) to develop a text generator and use it to procedurally generate the text of their novel. GitHub is used to register for the event and share participants' progress throughout the month. To qualify as a NaNoGenMo winner, participants have to publish their code and share a generated text of at least 50,000 words.

Since NaNoGenMo takes place online, we can use it to study practical approaches to text generation and story generation. Participants do not necessarily use state-of-the-art techniques from story generation research. Instead, the NaNoGenMo entries offer us a look into practical novel generation methods used in a (mostly) non-academic context. NaNoGenMo provides an accessible repository of story generation projects (including both code and output) that is incomparable to any academic generation challenge in terms of diversity and scale. What makes NaNoGenMo extra interesting is that it focuses on the generation of texts with a much longer length than addressed in most scientific research.

We analysed the work of participants from NaNoGenMo 2018[2], see Section 3. We start with categorising the projects by their output type, focusing on projects that generate text with a novel-like structure. We then list the main methods for text generation used by participants in Section 4, since text generation methods influence the coherence of the output text. In Section 5, we discuss projects that generate text with a coherent narrative structure. We list the different approaches that were used to achieve this narrative structure, and link them to scientific literature. Finally, we provide some recommendations on when to use which approach.

2 Related work

2.1 NaNoGenMo

NaNoGenMo was invented in 2013 by Darius Kazemi. His inspiration was NaNoWriMo, or National Novel Writing Month, another online event

[1] https://www.github.com/nanogenmo

[2] https://github.com/NaNoGenMo/2018

Proceedings of the Second Storytelling Workshop, pages 65–74
Florence, Italy, August 1, 2019.

in November where participants are challenged to write a 50,000 word novel.

The first attempt to create a survey of text generation methods used by NaNoGenMo participants was a blog post[3] in Russian. The author discussed projects of NaNoGenMo 2013-2015, and categorised them by generation technique, such as Markov chains, recycling existing works of fiction, simulation, high-level plot generation, and neural networks. Inspired by this blog post, the NaNoGenMo community conducted their own survey[4] of methods (2016) and programming languages (2014–2017) as part of the event.

There is some cross-pollination between the NaNoGenMo community and academia. Participants sometimes refer to research articles, either for their own projects or to help other participants. Additionally, NaNoGenMo has been mentioned in scientific literature in fields that have a close connection to the goal of the event: procedural generation for games (Karth, 2018), story generation (Montfort, 2014; Horswill, 2016) and computational creativity (McGovern and Scott, 2016; Cook and Colton, 2018; Compton et al., 2015).

Cook and Colton (2018) discuss the NaNoGenMo community in detail in their paper on online communities in computational creativity. Although they review some of the projects from NaNoGenMo 2016, the focus of their article was not the methods or quality of the projects, but rather the community of NaNoGenMo itself. Montfort (2014) developed a novel generator called World Clock, as entry for NaNoGenMo 2013. Interestingly, most of the researchers citing NaNoGenMo have participated themselves in the past.

2.2 Story generation

Story generation is the procedural creation of stories. Mostafazadeh et al. (2016) define a *story* or *narrative* as "anything which is told in the form of a causally (logically) linked set of events involving some shared characters." Yao et al. (2019) split story generation into two distinct tasks: on the one hand generating the sequence of events, and on the other hand generating the surface text, i.e. the actual text that makes up the story. This is reminiscent of the classic NLG pipeline (see, e.g., (Reiter and Dale, 1997), in which planning stages precede surface realisation. Story generation is sometimes limited to the first aspect, generating the underlying sequence of events, or *fabula*, with generation of the surface text of the story out of scope (Martin et al., 2018; Porteous and Cavazza, 2009; Lebowitz, 1987). Some story generation systems focus on generating the surface text, given a fabula as input. For example, Storybook (Callaway and Lester, 2002), the Narrator (Theune et al., 2007) and Curveship (Montfort, 2009) all generate story text from an underlying fabula. Other research focused on aspects of the *telling* of the story, for example stylistic variation (Montfort, 2007), affective language (Strong et al., 2007) and personality (Lukin et al., 2014; Walker et al., 2011).

2.3 Narrative coherence

The generation of long stories, such as the 50,000 word novels of NaNoGenMo, places strong demands on coherence: the set of events in the story need to be linked, and preferably also fit into some overarching dramatic structure.

One way of achieving coherence in generated stories is by imposing a specific structure on the output text. Researchers have investigated the structure inherent in existing stories to find out how humans do this. Propp's model of the structure of Russian folktales has been used in various story generation systems (Gervás, 2013). Alternative narrative structures that have been used to guide story generation are Booker's seven basic plots (Hall et al., 2017), the Hero's journey or Monomyth (García-Ortega et al., 2016) and the Fool's journey from tarot cards (Sullivan et al., 2018).

In neural text generation, it is less easy to impose a narrative structure on the generated texts – unless the task is split into two steps, like in the work of Yao et al. (2019). An alternative way improve the global coherence in texts generated with recurring neural networks was proposed by Holtzman et al. (2018), who used a set of discriminators to encode various aspects of proper writing.

Another way of achieving coherence is through emergent narrative (Aylett, 1999). This is a type of narrative (at the fabula level) that emerges from simulating simple behaviours that, when interacting, create a complex whole. The simulation gives rise to a sequence of causally linked events which give coherence to the story. The coherence in emergent narrative tends to be mostly local in nature:

[3] https://habr.com/en/post/313862/

[4] The programming language surveys can be found by searching for issues labeled 'admin' in the GitHub repositories for those respective years.

although the events are linked through their immediate causes and consequences, it is difficult to impose a global dramatic arc on them. Examples of generation systems that use the emergent narrative approach are FearNot! (Aylett et al., 2005), the Virtual Storyteller (Swartjes and Theune, 2008) and the simulation framework from Talk of the Town (Ryan et al., 2016).

Simulation-based narratives are particularly suitable for game-based story generation, since games often already have a world-state, characters, objects and a set of rules that describe valid changes to the game state. The rule system of role-playing game Dungeons & Dragons is the most well-known of its kind. Various story and quest generation systems (Martens, 2015; Tapscott et al., 2018; Kybartas and Verbrugge, 2014) have been built upon this and other related rule systems.

3 Data

NaNoGenMo uses GitHub's built-in issue tracker to keep track of all user submissions. Every issue corresponds to one NaNoGenMo project. In the issue thread, participants can post comments, interact with other users, share their development process and publish previews of the generated novels.

We downloaded all issues from the NaNoGenMo 2018 repository as JSON data using the GitHub API. We took issues into account that were opened between the start of NaNoGenMo 2018 and March 2019, that were labels as 'completed' and not labeled as 'admin'. The label 'completed' means that both the generator code and a 50,000 word output are publicly available. All 61 issues[5] were manually reviewed by the first author, by looking at the programming code, the output, the tools and used datasets. For practical reasons, we ignored projects in other languages than English.

NaNoGenMo uses a loose definition of 'novel': any text of more than 50,000 words qualifies as an acceptable output. There are no rules dictating the format, style, grammaticality, subject or content of the text. As a result, the outputs vary greatly from one another. See Figure 1 for a categorisation of NaNoGenMo projects according to their output type. Most projects generate a novel-like text, with a form that resembles sentences, paragraphs and chapters. One participant (project 72) created a generator for an Interactive Fiction game. Other projects generated word art, e.g. repetitions of one word, ASCII art or text without meaning, poems, graphs or lists. In the rest of this paper, we will limit our discussion to the 35 projects that generate novel-like text.

[5]Throughout this paper we will reference each project by its issue number on GitHub. The details of each project can be found on the corresponding issue page on GitHub, i.e. https://github.com/NaNoGenMo/2018/issues/{issuenumber}.

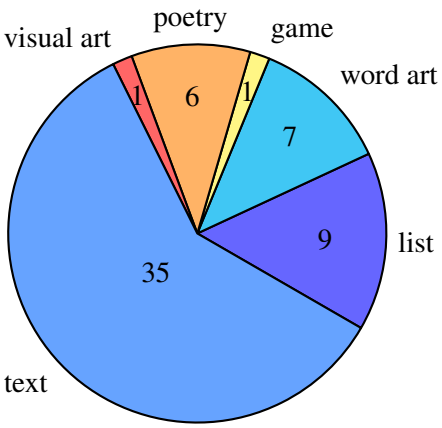

Figure 1: Output type of completed NaNoGenMo 2018 projects.

Language	Projects
Python	19
Javascript	8
Lua	3
Bash	3
C	2
Samovar	1
Ruby	1
Perl	1
PHP	1
ML	1
Julia	1
Java	1

Figure 2: Programming languages used in NaNoGenMo projects that generate novel-like text. Projects that use more than one programming language are counted multiple times.

For an overview of the programming languages used in the projects, see Figure 2. Some projects used multiple languages. The availability of good NLP and NLG resources in a particular language has probably contributed to people choosing those languages. Consequently, the choice for a particular programming language may have influenced the chosen text generation and narrative generation approach, and vice versa. Both Python and Javascript,

the two most popular programming languages with NaNoGenMo participants, have accessible libraries for text processing and text generation. Participants that programmed in Python mainly used Markovify, SpaCy and NLTK; Javascript projects used mostly Tracery (Compton et al., 2015), a Javascript library for text generation with context-free grammars. The developers of Tracery specifically mention the NaNoGenMo community as the target audience for Tracery, which could explain the wide adoption of Tracery within the NaNoGenMo community, as well as the large number of projects in Javascript, a programming language that is not typically used for text generation or text processing.

In addition to NLP libraries and tools, most participants use externally sourced text data. Public domain books from Project Gutenberg[6] and The Internet Archive[7] were very popular with NaNoGenMo participants, as was Darius Kazemi's Corpora[8] repository, which is a collection of word lists organized by subject, such as games, medicine and religion. Some participants created their own corpus from online resources, such as subtitles, marathon reports, horror stories and reports of personal experiences with psycho-active drugs.

4 Text generation methods

The 35 novel generation projects of NaNoGenMo 2018 use a variety of text generation methods to create the surface text of their novel. In this section, we provide a survey of the various approaches we have seen.

4.1 Templating

More than 10 projects use some form of *templating*. Libraries like Tracery offer a fast way to implement this in Javascript and Python. Most text templates were hard-coded in the generator, which is time-consuming and requires manual effort. An alternative approach used in some projects (projects 64, 101 and 104) was to create templates automatically, e.g. by running all sentences from a corpus through a part-of-speech (POS) tagger and creating sentence templates from the POS-tags.

The popularity of templating is not surprising, as templates offer a strong form of control over the surface text. However, using templates does not guarantee a good quality output. If templates are filled with randomly chosen phrases, as was done in some projects, the quality of the generated text may be worse than that of a text generated with Markov chains (discussed next).

4.2 Markov chains

At least 8 projects used *Markov chains* for text generation. Markov chains are statistical language models, which can be created fully automatically from corpora. They can be used for text generation by choosing a start token and using the probabilities in the model to choose the next token. Using Markov chains is an accessible approach to text generation, as it does not require coding the content of the output. Markovify[9], a Python library for working with Markov chains, was used by the majority of users that used Markov chains for generation. We believe that Markovify has contributed to the popularity of the Markov chain approach under NaNoGenMo participants.

Not your average ultra (project 89) creatively mixes the outputs of two Markov chains. One Markov chain was trained on a collection of marathon reports, the other on a dataset of reports of personal experiences with psychoactive drugs. As the generator produced more text, the influence of the second Markov chain on the generator grew stronger, which resulted in output in the form of a race journal that becomes progressively delirious over time.

Although the outputs from a Markov chain are often less coherent than those produced by templates, the advantage of Markov chains is that they often yield surprising or interesting results. For participants that value creativity over coherence, Markov chains are a suitable technique for text generation. As we will see in Section 5, the lack of coherence is not always a problem.

4.3 Remixing

Remixing external sources, such as text from existing novels, was also a popular approach with participants. More than half of the projects use some form of remixing to create their output. One example of remixing is creating a new text by taking a source text and substituting words from the text according to specific rules. A hilarious example of this is *Textillating* (project 96), where Dickens' Great Expectations is 'improved' by increasing the number of exclamation marks and substituting each

[6] www.gutenberg.org
[7] www.archive.org
[8] https://github.com/dariusk/corpora
[9] https://github.com/jsvine/markovify

adjective in the text with its most extreme synonym.

Some participants collected external sources and composed their novel by cutting-and-pasting sentences from these. For example, *Doctor, doctor!* (project 86) used the corpus of Yahoo! health questions and answers to generate a dialogue between a doctor and a patient. Another participant scraped sentences from GoogleBooks about a set of topic words, and created an original text by cutting-and-pasting snippets from Google Books preview files. In some cases, remixing was paired with statistical modeling. The author of *Angela's claustrum* (project 28) transformed an old NaNoWriMo novel draft into an outline and remixed this into a new novel by using a stochastic model of Gutenberg texts.

With this category of methods, either the output text is very similar to the source text (and similarly coherent), or the output is completely new but loses some coherence in the process, often because developers chose to introduce random words into existing sentences in their word substitution.

4.4 Machine Translation

There were various generators that used *machine translation* techniques for creating a novel. Project 22 created a new text by mapping every sentence of Northanger Abbey by Jane Austen to a sentence written by Sir Arthur Conan Doyle, using sentence embeddings.

Project 61 used machine translation to transform the text of one of L. Frank Baum's Oz books. All dialogue from the book was translated to the "language" of The Muppets' Swedish Chef, and all other text was translated to Valleyspeak.[10]

One participant (project 33) used a public domain movie as the basis for their novel. They turned the movie into a collection of screenshots and fed this to Microsoft Cognitive services to generate captions for the screenshots. The captions were then transformed into novel text. This can be seen as a form of machine translation. Instead of translating between different languages, this project translates between different modalities (video to image, image to text).

Machine translation within NaNoGenMo can be seen as a form of remixing, and the drawbacks are indeed very similar. Either the output text shows a strong resemblance to the original text, or it is more creative but ends up incoherent.

[10] Valleyspeak is an American social dialect that originates from the San Fernando Valley in Southern California.

4.5 Deep learning

Finally, there were three projects that used deep learning to create their novel. Two projects, project 73 and project 76, used Torch[11] to create an LSTM architecture trained on an external dataset. Project 73 trained the LSTM on a crowdsourced collection[12] of Dungeons & Dragons character biographies, and project 76 used user-written horror stories scraped from CreepyPasta[13]. Both projects have output that is neither coherent nor grammatical. However, the LSTM does manage to convey the typical style of RPG biographies and horror stories. Finally, project 99 used machine learning to see whether a neural network trained on the text of Moby Dick could succesfully reconstruct the original text, by predicting the sequence of sentences.

5 Methods for narrative coherence

NaNoGenMo output is at least 50,000 words, or roughly 75 pages of text. This is a much greater length than is usually produced by story generation systems. In computational creativity and creative NLG, typical outputs range from tweets (140-280 characters) to stories of one or two pages, with exceptions such as Curveship (Montfort, 2009), UNIVERSE (Lebowitz, 1987) and World clock (Montfort, 2014).

To see how NaNoGenMo participants generate coherent novel-length narratives, the first author performed an informal analysis of the outputs of the 35 text generation projects, specifically focusing on coherence and the presence of narrative structure. Out of the 61 projects of NaNoGenMo, only 14 projects had a narrative structure, that is, they exhibited coherence as discussed in Section 2.3. Below we give an overview of the approaches used to achieve this. We can categorise the approaches for generating this narrative as follows.

5.1 High-level specification

Some projects achieve coherence by hard-coding a narrative structure in their input. *The League of Extraordinarily Dull Gentlemen* (project 6) defines that narrative structure in a specification written in Samovar, a PROLOG-like domain-specific language for world-modeling using propositions. The

[11] http://torch.ch/

[12] https://github.com/janelleshane/DnD_bios

[13] https://www.creepypasta.com/

specification is a high-level description of the story, with its representation level a mix of a fabula and surface text: it is not just a sequence of events, but also includes dialogue and narrative exposition. The surface text for its output was generated by running the specification through Samovar's assertion-retraction engine[14], taking the resulting sequence of events and realising those into sentences with a Python script. This approach is similar to that of other story generation systems that use logic programming to generate stories or fabulas, such as (Martens, 2015), (Robertson and Young, 2015) and (García-Ortega et al., 2016).

Hard-coding a narrative arc in a specification can be seen as high-level templating. It also has similar advantages as templating: because the author specifies the narrative arc by hand, they have tight control over the surface text, which results in an output that looks like it was written by a human. However, this approach places an authorial burden on the developer of the generator. The story of project 6 of 50,000 words was generated in 930 lines of Samovar. We expect that the effort of writing this specification could be further reduced with code generation. Another disadvantage is that one story specification defines exactly one surface text. The surface text of project 6 includes little variation. The book consists of scenes where multiple characters perform the same action in sequence. Repeating patterns are clearly visible in the output text, making for a dull read – hence the title of the project. However, the output of project 6 sets itself apart from other generated novels by having grammatical surface text and maintaining a clear traditional narrative arc throughout the entire story with a beginning, an incident, a climax and a problem resolution.

For authors that want to generate a story out of a high-level story description, using a domain specific language like Samovar might be a suitable solution. The code for this NaNoGenMo project is very readable and could serve as an introduction to this approach. As this approach requires the user to write part of the story, it is less suitable for projects where the author also wants the generator to create the contents of the fabula, or requires a lower cost in terms of writing the code and specification.

[14] https://catseye.tc/article/Languages.md#samovar

5.2 Hard-coded narrative elements

Instead of hard-coding the narrative structure of the entire story in the generator, it can be hard-coded only in specific places. An example of this approach from outside NaNoGenMo is described in Reed (2012), where the author used a grammar to generate 'satellite sentences' that can be inserted in a larger human-authored narrative for an interactive fiction game. Satellite sentences are sentences that "moderate pacing and reestablish context within dialogue scenes" (Reed, 2012), such as "She coughed", "The clock was ticking" and "It was getting late".

There were a few NaNoGenMo projects where the generated text itself had no structure at all, but where the developer still created a narrative by providing a hard-coded section at the beginning and/or ending of the book. Having a fixed beginning and ending can tie otherwise incoherent pieces of generated text together, as it gives readers a context in which they can interpret the generated text. Even text generation techniques that normally do not lead to coherent output, such as Markov chains and random generation, can still be 'saved' by using this technique.

An example is *Not your average ultra* (project 89), which succesfully frames the (in)coherence of Markov chains by naming the specific setting of the novel at the beginning and end: an ultramarathon.

Similarly, *The Defeat at Procyon V* (project 83) contains 50,000 words of dialogue between a science fiction Fleet Commander and their Super Admiral. The lines of dialogue are randomly generated from a grammar of science fiction technobabble, occasionally interspersed with exposition sentences, similar to the satellite sentences from Reed (2012). Because the beginning and ending of the novel are fixed, the reader has a context in which to interpret the conversation: the conversation is about the various weapons and technologies that were deployed in the defense of Procyon V.

With this approach, the problem of generating a coherent narrative is transformed into writing narrative elements that frame the generated text in such a way that the reader perceives a narrative in the entire text. It particularly useful in instances where developers prefer straight-forward text generation techniques over narrative generation techniques, and for developers that want to write as few lines of code as possible.

5.3 Simulation

There were various projects (projects 11, 18, 39, 60 and 100) that used simulation as the basis for their narrative. The projects with simulation-based narratives had two things in common.

Firstly, most projects used rule systems that are similar to those of well-known role-playing games. For example, *The Longest Corridor* (project 18) uses a combat system that closely resembles that of Dungeons & Dragons. The project generates stories about a mythical corridor filled with monsters and treasure. For each chapter of the novel, the system generates a hero who has to fight their way through the corridor. If the hero is defeated by the inhabitants, the hero's remains will stay in the corridor for later heroes to find. If the hero reaches the end of the corridor and finds the treasure, they install themselves as the new master of the corridor, waiting for new adventurers to come and challenge them. This continuity, where characters of previous chapters (old world state) can interact with characters from current chapters (current world state), is what moves this project from a straight-forward simulation into the realm of narrative. Similarly, *Of Ork, Fae, Elf and Goblin* (project 39) generates a fabula of a group of creatures that fight each other with procedurally generated weapons in different rooms.

Another roleplaying-game inspired project is *High Fantasy with Language Generator* (project 60). Instead of having one global text-level simulation that tracks the world state and governs the entire narrative, it uses multiple low-level simulations that each govern one type of event. The project follows a group of adventurers on their quest. During their travels, the characters encounter monsters, visit local taverns and play dice games with strangers. For each of these scenes, the generator uses a separate simulation.

A second property of simulation-based novels is that they often have a journal-like format. The world state of the simulation gives rise to the surface text of the story. Since the world state is updated with each clock tick, it is intuitive to let the novel's sections (chapters, paragraphs) correspond to one clock tick. Consequently, simulation-based narratives are particularly suitable for generating journals or logbooks, in which each section corresponds to one unit of time. *The Pilgramage* (project 11) is an example of a project that follows a journal format.

A weakness of some of the simulation-based projects in NaNoGenMo is that they generate events that are not linked to each other. An example is *Wheel of Fortune* (project 100), which simulates characters who slowly grow old and die, all the while experiencing events that are generated from randomly drawn tarot cards. The resulting sequence of events looks like a fabula. However, the events are not related to each other and do not influence each other: the characters' actions happen completely in a vacuum. This does invite the reader to imagine their own narrative, but this requires a lot of effort on part of the reader. Still, symbolism from tarot cards can be used successfully to shape a narrative when combined with other methods, such as high-level specification of narrative structure (see Section 5.1). A story generator from outside NaNoGenMo that also used the tropes from tarot was developed by Sullivan et al. (2018). However, Sullivan et al. (2018) used the tarot cards to generate movie-like story synopses, with a plot structure based on Booker's seven basic plots and screenwriting principles.

5.4 Evoking a narrative

Some of the project outputs *evoke* a narrative, even though there is no narrative structure explicitly present in the text. This can even be the case for output texts that are not grammatical. Incoherent texts that still have a recognizable novel form force the reader to guess the meaning of the author. This subjective interpretation might still evoke a narrative in the reader.

As Veale (2016) notes in his paper on poetry generation, form can be more important than content. Veale calls this effect 'charity of interpretation': if humans see a text in a well-known form (or *container*), they are disposed to attribute more meaning to the text than it actually has. We saw two distinct ways of achieving this.

If the text of the novel is limited to a specific subject, readers will try to fill in the gaps in the structure with their own knowledge and expectations. An example of a project that limits its topic to instill a sense of coherence is *Doctor, doctor!* (project 86). The output text has the form of a dialogue, consisting of randomly chosen questions and answers from a dataset of Yahoo! questions from the health domain. The questions and answers have no logical connection whatsoever, but the vocabulary and writing style will be recognizable to

readers who are familiar with medical discussions on the internet. Even though the answers of the doctor make no sense in the context of the respective questions, readers will infer that this novel is about a dialogue between a doctor and their hypochondriac patient.

Another technique for evoking a narrative is by connecting unrelated random elements with each other to improve the perceived coherence. *Out of Nowhere* (project 57) simulates an interaction between its characters by connecting interactions at the word level, which we explain below. Out of Nowhere produces the script for a play, based on lines of English text from public-domain phrase books. The characters represent different nationalities, and their dialogue lines are based on the text of phrase books for their respective languages. The character dialogue is generated by choosing lines from each character's phrase book. Most dialogue lines are chosen randomly, but the generator increases the coherence of the output with a few tricks. Both the location and the interactions are influenced by the words that occur in previous lines. For example, if the previous line contains the word 'waiter', the generator will include a restaurant or cafe in the scene. Similarly, if one of the previous lines contains a question mark and an interrogative word ("what", "who", etc.), the generator will assign a higher probability to lines that would constitute a logical answer. For example, if previous lines contain the phrase "Where is ...?" the generator favors sentences like "In Timbuktu" or "At my house". This is a similar approach as is used in Reed (2012), where the text generator takes different types of context into account, such as dialogue progression, location and time of day. The difference is that Reed tagged his text with locations for the satellite sentences, whereas the generator of project 6 generates all sentences and their connections on the fly. The result of project 57 is a script that has similar quality as the generators that use the simulation approach, even though there is no underlying world state for this play. All the coherence comes from word-level choices.

Besides limiting the topic of a text, using the right style can increase the perceived coherence of a text as well. If a reader recognizes a particular style from a particular type of narrative, the reader might infer meaning where there is none. A project that adapts this idea in an original way is *Velvet black skies* (project 65), which uses statistical modeling to find the most cliche sentences in a corpus of science fiction writing. The developers defined cliches as "n-grams that occur in the texts of more than 30 authors." The generator creates a new text from these cliches by clustering them by topic and by remixing them into a chapter for each topic. Readers of science fiction classics will immediately recognize the particular style of vintage science fiction.

The above techniques ask something extra of the reader during the interpretation of the text. As such, they are suitable for situations where the writer wants to highlight the subjective experience of the reader in ascribing meaning to a text. Additionally, these techniques could be used for collaborative creation in text generation (authoring aids), i.e. applications where a computer generates a first draft of a story and the human finishes it. In the latter case, the human author can take the concepts created by the generator and polish them before publication.

6 Conclusion

We discussed the most prevalent text generation methods from NaNoGenMo 2018 and their respective advantages and disadvantages. We discussed four different approaches that were used to achieve coherence (or the semblance of it) in novel-length texts, highlighting some of the most creative projects.

If there is already a high-level story arc thought out for the surface text, using a high-level specification to define this story arc is a good approach. Hard-coding the high-level narrative arc in a specification can reduce the authorial burden of manually writing the full text significantly. However, the approach is not suitable for projects where the generator should generate the fabula in addition to the surface text.

If the generator is also in charge of generating the events that underlie the surface text, simulation-based approaches are a good choice. It has been applied in various story generation systems already, most notably for the game domain, because of the overlap in functionality between simulations for narratives and rule systems for games. A weakness of simulation approaches is that, if the generated events are not interrelated, the sequence of events generated by a simulation lacks narrative coherence.

However, even text generation methods that do

not create coherent text can be turned into a narrative, either by hardcoding narrative elements, such as a contextualising beginning or ending, or by evoking a narrative by exploiting readers' charity of interpretation.

In this paper we could only give a high-level overview of the different approaches, and briefly discuss a few example projects. Those who want to see more examples and study the different approaches in detail can refer to the NaNoGenMo repository on GitHub. We have made the data for our analysis in this paper available online.[15]

7 Acknowledgments

This research is supported by the Netherlands Organisation for Scientific Research (NWO) via the DATA2GAME project (project number 055.16.114). We would like to thank the reviewers for their useful remarks.

References

Ruth Aylett. 1999. Narrative in virtual environments-towards emergent narrative. In *Proceedings of the AAAI fall symposium on narrative intelligence*, pages 83–86.

Ruth S Aylett, Sandy Louchart, Joao Dias, Ana Paiva, and Marco Vala. 2005. Fearnot!–an experiment in emergent narrative. In *International Workshop on Intelligent Virtual Agents*, pages 305–316. Springer.

Charles B Callaway and James C Lester. 2002. Narrative prose generation. *Artificial Intelligence*, 139(2):213–252.

Kate Compton, Ben Kybartas, and Michael Mateas. 2015. Tracery: an author-focused generative text tool. In *International Conference on Interactive Digital Storytelling*, pages 154–161. Springer.

Michael Cook and Simon Colton. 2018. Neighbouring communities: Interaction, lessons and opportunities. In *International Conference on Computational Creativity*.

Rubén H García-Ortega, Pablo García-Sánchez, Juan J Merelo, Aránzazu San-Ginés, and Ángel Fernández-Cabezas. 2016. The story of their lives: Massive procedural generation of heroes' journeys using evolved agent-based models and logical reasoning. In *European Conference on the Applications of Evolutionary Computation*, pages 604–619. Springer.

Pablo Gervás. 2013. Propp's morphology of the folk tale as a grammar for generation. In *Proceedings of the 2013 Workshop on Computational Models of Narrative*, volume 32. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

Jason Andrew Hall, Benjamin Williams, and Christopher J Headleand. 2017. Artificial folklore for simulated religions. In *2017 International Conference on Cyberworlds (CW)*, pages 229–232. IEEE.

Ari Holtzman, Jan Buys, Maxwell Forbes, Antoine Bosselut, David Golub, and Yejin Choi. 2018. Learning to write with cooperative discriminators. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1638–1649, Melbourne, Australia. Association for Computational Linguistics.

Ian D Horswill. 2016. Dear leader's happy story time: A party game based on automated story generation. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Isaac Karth. 2018. Preliminary poetics of procedural generation in games. *Proc. Digital Games Research Association*.

Chloé Kiddon, Luke Zettlemoyer, and Yejin Choi. 2016. Globally coherent text generation with neural checklist models. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 329–339.

Ben Kybartas and Clark Verbrugge. 2014. Analysis of ReGEN as a graph-rewriting system for quest generation. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(2):228–242.

Michael Lebowitz. 1987. Planning stories. In *Proceedings of the 9th annual conference of the cognitive science society*, pages 234–242.

Stephanie M. Lukin, James O. Ryan, and Marilyn A. Walker. 2014. Automating direct speech variations in stories and games. In *Tenth Artificial Intelligence and Interactive Digital Entertainment Conference*.

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

Lara J Martin, Prithviraj Ammanabrolu, Xinyu Wang, William Hancock, Shruti Singh, Brent Harrison, and Mark O Riedl. 2018. Event representations for automated story generation with deep neural nets. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Jeffrey D McGovern and Gavin Scott. 2016. Eloquentrobot: A tool for automatic poetry generation. In *Proceedings of the Seventh ACM Conference on Bioinformatics, Computational Biology, and Health Informatics*.

Nick Montfort. 2007. *Generating narrative variation in interactive fiction*. Ph.D. thesis, University of Pennsylvania.

[15] https://github.com/jd7h/narrative-gen-nanogenmo18

Nick Montfort. 2009. Curveship: An interactive fiction system for interactive narrating. In *Proceedings of the Workshop on Computational Approaches to Linguistic Creativity*, pages 55–62, Boulder, Colorado. Association for Computational Linguistics.

Nick Montfort. 2014. New novel machines: Nanowatt and World clock. Trope Tank Technical Report TROPE-13–03, July 2014.

Nasrin Mostafazadeh, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James Allen. 2016. A corpus and cloze evaluation for deeper understanding of commonsense stories. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 839–849, San Diego, California. Association for Computational Linguistics.

Julie Porteous and Marc Cavazza. 2009. Controlling narrative generation with planning trajectories: the role of constraints. In *Joint International Conference on Interactive Digital Storytelling*, pages 234–245. Springer.

Aaron A Reed. 2012. Sharing authoring with algorithms: Procedural generation of satellite sentences in text-based interactive stories. In *Proceedings of The third workshop on Procedural Content Generation in Games*. ACM.

Ehud Reiter and Robert Dale. 1997. Building applied natural language generation systems. *Natural Language Engineering*, 3(1):57–87.

Justus Robertson and R Michael Young. 2015. Automated gameplay generation from declarative world representations. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*.

James Ryan, Michael Mateas, and Noah Wardrip-Fruin. 2016. Characters who speak their minds: Dialogue generation in talk of the town. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Lauren R. Shapiro and Judith A. Hudson. 1991. Tell me a make-believe story: Coherence and cohesion in young children's picture-elicited narratives. *Developmental Psychology*, 27(6):960–974.

Christina R. Strong, Manish Mehta, Kinshuk Mishra, Alistair Jones, and Ashwin Ram. 2007. Emotionally driven natural language generation for personality rich characters in interactive games. In *Proceedings of the Third Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, pages 98–100.

Anne Sullivan, Mirjam Palosaari Eladhari, and Michael Cook. 2018. Tarot-based narrative generation. In *Proceedings of the 13th International Conference on the Foundations of Digital Games*, page 54. ACM.

Ivo Swartjes and Mariët Theune. 2008. The virtual storyteller: Story generation by simulation. In *Proceedings of the 20th Belgian-Netherlands Conference on Artificial Intelligence (BNAIC)*, pages 257–264.

Alan Tapscott, Carlos León, and Pablo Gervás. 2018. Generating stories using role-playing games and simulated human-like conversations. In *Proceedings of the 3rd Workshop on Computational Creativity in Natural Language Generation (CC-NLG 2018)*, pages 34–42.

Mariet Theune, Nanda Slabbers, and Feikje Hielkema. 2007. The Narrator: NLG for digital storytelling. In *Proceedings of the Eleventh European Workshop on Natural Language Generation*, pages 109–112, Saarbrücken, Germany.

Tony Veale. 2016. The shape of tweets to come: Automating language play in social networks. *Multiple Perspectives on Language Play*, 1:73–92.

Marilyn A. Walker, Ricky Grant, Jennifer Sawyer, Grace I. Lin, Noah Wardrip-Fruin, and Michael Buell. 2011. Perceived or not perceived: Film character models for expressive NLG. In *International Conference on Interactive Digital Storytelling*, pages 109–121. Springer.

Lili Yao, Nanyun Peng, Weischedel Ralph, Kevin Knight, Dongyan Zhao, and Rui Yan. 2019. Plan-and-write: Towards better automatic storytelling. In *Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)*.