Spinning Coherent Interactive Fiction through Foundation Model Prompts

Alex Calderwood, Noah Wardrip-Fruin, Michael Mateas
Expressive Intelligence Studio
Computational Media Department
University of California, Santa Cruz
alexcwd,nwardrip,mmateas@ucsc.edu

Abstract

We present *Spindle*, a mixed initiative tool for authoring choice-based interactive fiction that targets *Twine*, a popular framework for text-based storygames. *Twine* artifacts have properties of both stories and games, placing our system at the intersection of Automated Game Design (AGD) and Automated Story Generation (ASG). We construct a generative pipeline that involves condensing narrative context into a compact representation in order to feed to a pretrained language model, which we further fine-tune. We demonstrate that, by maintaining narrative context in the prompt presented to the language model, we can greatly improve over the loss of long-term coherence that still plagues such models. Our story compression technique for representing narrative context uses a handful of freely available natural language processing libraries and models, demonstrating that such interpretive pipelines can be built with limited computational resources and low cost. The resulting tool is capable of producing full-text branching narratives, or of generating individual passages that maintain a high degree of narrative coherence with the prior passages. The framework we design is both language model-agnostic and narrative theory agnostic, allowing future researchers to easily expand on it with new language models and story representations. We release our code under the BSD-4-Clause[1].

Introduction

While large language models have proven capable of producing highly coherent text for such purposes as autocompletion and chat bots, less research effort has gone into exploring the entirely new forms of media they enable. We are interested in ludic spaces that may be created by advancements in language model-based story generation, and present one such tool, a system for authoring interactive fiction which allows for expressive interaction with an AI model. By focusing on the experience of writing as a target for AI augmentation as well as a source of entertainment, this project is influenced by two fields of study: computer mediated writing and automated game design, resulting in a system that is both engaging and encouraging of narrative experimentation.

Our tool allows users to author interactive-fiction narratives with the Twine framework, alternately 'passing the keyboard' between the system's language model and the user at will. When queried, the system utilizes a method of narrative compression to come up with an understanding of the current thrust of the story, and uses that understanding to condition the generation of the next passage. This approach is novel in how it uses the compressed narrative context to improve the coherence of generated narratives through prompt engineering. Prompt engineering is a new and understudied paradigm where pretrained language models are guided towards better responses by providing a succinct (often templated) prompt to the model at the time of prediction (Reynolds and McDonell 2021). This method of iterating on a series of prompts has been successfully used in the computational creativity literature for text to image generation (Liu and Chilton 2021).

Unlike many past story generation techniques, which generate a list of ordered plot events, our system generates fully realized multi-passage branching narratives, with a capability for mixed-initiative use. The generated passages follow Twine syntax and include outgoing links (player decision points) and corresponding passage titles. Generated passages display a level of narrative coherence that allows the model to 'yes-and' the user's apparent authorial intention, while still enabling a degree of defamiliarization that results from the composition of nearly appropriate text, an attribute of AI writing which has been said to be prized by writers including the novelists Robin Sloan and Sigal Samuel, who respectively envision an AI writing assistant as "less Clippy, more séance" and describe feeling "strangely moved" by AI writing (Calderwood et al. 2020).

We fine-tune GPT-3, a model notable for its introduction of meta-learning, or zero-shot learning, to language generation (Brown et al. 2020). Under the zero-shot paradigm, queries consist of a small number of example (*prompt*, *response*) pairs. Further, fine-tuning such a model allows it to capture the grammar and stylistic accompaniments of a structured corpus such as our dataset of Twine stories. The model has notably been used to enable gameplay in *AI Dungeon*[2], a text adventure game that allows arbitrary player input (Hua and Raley 2020). Accessing the model through the

[1] https://github.com/alex-calderwood/spindle

[2] https://play.aidungeon.io/

OpenAI API and utilizing other open-source NLP packages, we fine-tune our GPT-3 based models at a small fraction of the cost of traditional NLP development, and all systems easily run on an internet-connected laptop.

Background

Creating a Twine game is an act of writing as well as interaction design; Twine games are 'played' by reading and read through play, and generating these artifacts requires both story and game generation techniques.

Automated Story Generation

Automated story generation has long been considered a grand challenge of artificial intelligence. Early systems used symbolic reasoning (Meehan 1977; Lebowitz 1987), often utilizing hierarchical generation based on grammar rules which provide model interpretability, clearly identifiable design spaces, easy extensibility, and little or no input data, though they sometimes lack robustness (Black and Wilensky 1979; Riedl and Young 2010). Tracery has found popularity as an author-focused generative text tool with non-deterministic grammar production (Compton, Kybartas, and Mateas 2015). When it comes to full stories, "hand-authored templates are seen as insufficient for large scale narrative generation" (Roemmele 2018), though the combination of generative grammars with language modeling has not been sufficiently explored. A major component of many narrative generation systems includes a planner, a system that searches for a goal story state from the current state to find the series of narrative actions needed to bridge that gap (Porteous 2016; Young et al. 2013). StoryAssembler is a relevant example of a system that uses a planner to generate Twine-like interactive narratives (Garbe et al. 2019).

Utilizing statistical NLP, Chambers and Jurafsky (Chambers and Jurafsky 2008) define the *narrative cloze task* as the prediction of missing narrative events from an event sequence, which they produced from short plot descriptions. Their technique involves running a dependency parser over a collection of plot summaries to produce grammatical relationships, using these to extract a sequence of predicates along with their subject and object noun phrases, and then resolving these into ordered event chains. Pichota and Mooney (Pichotta and Mooney 2016) build on this framework, utilizing an LSTM architecture to extract (s, v, o, m) event tuples from Wikipedia articles (corresponding to subject, verb, object, and modifier, respectively). "John and Mary went to the store," becomes $(john, go, store, \emptyset)$. This event list may be thought of as a high-resolution, but potentially low accuracy, representation of the narratological notion of a fabula, or the chronological ordering of events in a narrative (Bal and Van Boheemen 2009).

Martin et al. follow this approach with a three part decoupled architecture: a system that extracts event tuples from unstructured narratives, an event2event model that predicts the next event predicated on the previous events, and an event2sentence model which expands a sequence of events back into natural language (Martin et al. 2018). To improve the model's performance, they used entity recognition to generalize characters and locations during prediction, which were memorized and later in-filled. This work serves as the closest inspiration for our approach. Our use of modern large-pretrained language models allows us to combine the event prediction and text generation steps into one text completion step, and our narrative interpretation module functions as a memory of important narrative entities without the need for back-filling generalized text.

Many attempts at narrative generation focus just on the production of language, but Mexica (Pérez and Sharples 2001) models creative writing as a cognitive process that consists of engaged and reflective states. In the reflective state, the model evaluates coherence, interestingness, and novelty of the generated sequences for text refinement. Our strategy utilizes reflexive interpretation to increase coherence of the proceeding passages. Like the generation of long-form text, automated interpretation of full stories has proven challenging, not least because it relies on systems such as long distance character co-reference resolution to be of a high-accuracy. BookNLP (Bamman, Underwood, and Smith 2014) is one of the few systems that approaches this tough problem. Increasing the quality of generated stories will likely need to incorporate global measures of coherence, and will therefore likely require the semantic readings that a system like BookNLP can provide.

The Virtual Storyteller System is a plot generation system that used a director model to orchestrate emotional episodic plots (Theune et al. 2003; Theune et al. 2004). The authors imagine a user acting as a character within the narrative, allowing them to influence the course of the story somewhat like a role playing game, not dissimilar to our mixed-initiative writer model.

Relation to Automated Game Design (AGD)

Full-game generation has historically focused on games that can be automatically analyzed with automated game players or a stack of static evaluation criterion (Pell 1992; Browne and Maire 2010; Cook, Colton, and Gow 2016). Such game generators seek to maximize a particular optimization function that guides search, which may be grammar-based, constraint-satisfying, or evolutionary in nature. They often utilize an intermediate Game Description Language (GDL) (Schaul 2013; Summerville et al. 2018; Duplantis et al. 2021), typically a text representation that is designed to have a high expressive range, produce interesting games a high proportion of the time, and be suitable for automatic search. Both Inform 7 and Ceptre can be viewed as narrative GDLs; the former is used in the TextWorld framework (Côté et al. 2018). In our case, Twine's Twee syntax[3] is modeled and generated analogously to these GDL's. From it, the Twee compiler targets interactive HTML.

The automated game players in AGD are intermediate processes that form interpretations of games, similar to the readings our system produces to conduct the narrative flow.

Cook et al. point out that most automatic game generation scholarship has gone towards "objectives, obstacles, and the notion of challenge" (Cook 2015). Game generators have

[3]Examples at: https://dan-q.github.io/twee2/tutorial.html

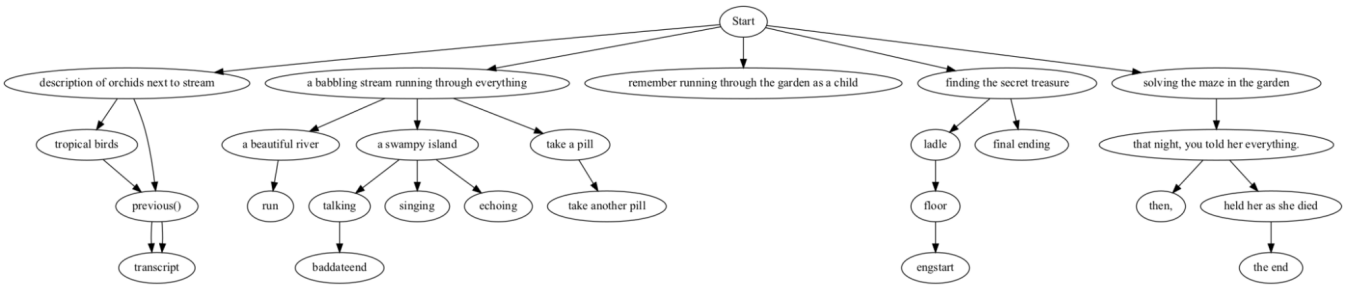

Figure 1: A small Twine tree. The *Start* node (passage and links) was written by hand, further passages and titles were generated by *Spindle*. For space, only passage titles links are shown.

recently begun exploring the hard problem of automatically producing games with semantic, thematic, and cultural import, rather than focusing on fun or challenge. In (Cook 2015), the authors point to narrow readings of the word 'game' as hampering fruitful experimentation in generative systems. Unanswerable questions like "Is this a game?", "Is this a story?", or "Is this art?" often surface at challenging moments in the development of new expressive mediums, defensively boxing-in existing endeavours rather than nurturing interdisciplinary growth. Game-O-Matic is the first game generation system that attempts to reason about the rhetorical relationships between objects that make up the game, utilizing the theory of operational logics to build up an understanding of possible player interpretations (Treanor et al. 2012). Gemini takes this a step further, using answer set programming to build up a proceduralist reading of a game as it is generated, using that reading to influence generation (Summerville et al. 2018). In (Cook 2021) the authors attempted to generate games without scores, focusing on the aesthetic experience of play. It is worth noting that they found it challenging to determine their level of success or generate through a broad expressive range, in part due to the unclear notion of success.

Automatic game generation and games as a whole have seen relatively slow adoption of language models. AI Dungeon is one notable exception (Hua and Raley 2020). A few reasons for this may be that language models are unpredictable, sometimes producing undesirable sexist, racist, and otherwise inappropriate language. Twine games are historically inclusive of explicit and otherwise non-normative content (Harvey 2014), meriting a conversation about what a nuanced treatment of complex social, sexual, and psychological issues looks like in the context of these models.

Additionally, getting these models to understand narrative context is difficult, as we will see. Some work has been done to use language models to evaluate text games (Côté et al. 2018; Kostka et al. 2017a; Kostka et al. 2017b). Fan et al. use neural generation to populate the world of a text-adventure game (Fan et al. 2020), but did not attempt full narrative generation. Earlier games such as Scribblenauts[4] have used human specified corpuses as safer and more reliable tools for injecting language understanding into games. (Barros et al. 2019) used Wikipedia as a corpus to automatically generate murder mystery games populated with characters and appropriately themed puzzles.

Twine

Twine is a platform designed to give non-coders the ability to author branching interactive fiction narratives. Such games are designed and written via a visual authoring tool that positions narrative passages on an interface resembling paper notes connected by strings on a corkboard (Friedhoff 2013). Passages are Twine's "equivalent of a page in a Choose Your Own Adventure book" (Friedhoff 2013), containing markdown-style hyperlinks to other passages. Gameplay consists simply in clicking between passage links, which may simulate making narrative choices for a first or third person character. Passages and links are handwritten, as contrasted with more open-ended commands available in parser games. Collections of Twine games include the Interactive Fiction Database[5] and itch.io[6].

Mixed-Initiative Interface

```
Narrative extraction set to v1.3
enter your story title: After the Storm(s)
by: alex
To Do List: ['Start']
(w) to write Start yourself
(g) to generate Start
(v) to view the written passages.
(f) to generate all remaining passages.
(q) to terminate writing with unwritten passages.
(W/g/f/v/q):
```

Figure 2: Entry into the prototype interface.

Our system models writing a Twine story as constructing a graph of passages connected by their respective links, first soliciting a title and by-line from the author, and then moving into the authoring of the *Start* passage.

Spindle's interface is designed to treat both the human and machine as writers capable of drafting or refining passages

[4]https://en.wikipedia.org/wiki/Scribblenauts

[5]https://ifdb.org/viewlist?id=ax0yq2ksub57ie7o

[6]https://itch.io/games/made-with-twine

in the working story. The system maintains a 'To Do List', or a fringe of outgoing passage titles that have yet to be drafted.

At present, the system's primary interaction loop begins by asking the user to select a passage title from the To Do List. Passage selection is accomplished with arrow keys — building a graphical UI is the focus of the next iteration of the tool. Next, the user is asked to indicate whether they would like to 1.) draft the passage body in a spartan text editor, 2.) indicate that the system should automatically generate the passage body using the narrative context stored for that passage, 3.) view the passages thus far written by the mixed-initiative system, 4.) generate N passages sequentially from the head of the list, or 5.) conclude the writing process, inserting placeholder text for unwritten passages.

The writer may for instance describe a branch point in an adventure game, say, a description of a cave with many tunnels for the player to explore. The tool is designed not to impose stylistic restrictions on the writer, so they may author link text in declarative or imperative styles ("You head into the entrance covered in stalactites" vs "Enter the slimy tunnel") or any other text ("An unusual door"). Additionally, the writer can override the link text with specific titles for the linked passages so that "enter the cave" corresponds to the more conventionally formatted title "The Cave".

The author may want to immediately write some passages and generate placeholder text for the others. Alternatively, they may decide to have the system generate all narrative branches recursively to create unplanned stories to play themselves. Using the tool in this manner allows it to produce 'games that the developer wants to play', which is commonly cited as a reason game makers get into development.

After a passage has been written by human or machine, it is parsed by the system to ensure it is well-formed Twine, and passed back to the writing protocol if not. In practice, the models we train rarely produce malformed syntax.

When all passages have been written or writing is manually concluded, the system uses the Twee compiler to convert the passages into playable HTML, saves those files, and then launches the Twine game in the default web browser as shown in Figure 3. At present, the games do not make use of custom visual styling, though we imagine future work will include conditionally generating stylesheets on the basis of a game's thematic content.

Formulating the Generation Problem

Twine stories can be thought of as directed graphs composed of *passages* $p_i := (p_i^{title}, p_i^{body})$ connected to others by outgoing links. Each story contains a predefined $Start$ node. Any text in the body may be formatted as a link to a specified passage[7].

The model should ideally be aware of all preceding text in order to generate the next passage, according to the practice of automated feature engineering in which deep learning models themselves learn what information is relevant to their predictions. In practice, it has been observed that longer prompts decrease the coherence of completions in large language models and many strategies for increasing long form coherence have recently been proposed (Guan et al. 2021; Cho et al. 2018). Prompt engineering therefore necessitates a trade-off between including more information that might be useful to the prediction, and not 'overwhelming' the model with too much text. State of the art text completion models typically allow no more than around 1024 words shared between the input (prompt) and output (completion). So we define two auxiliary functions whose purpose is to condense preceding passages to a length that can be ingested without a loss of coherence due to to this phenomenon.

These abstract functions are implemented according to authorial goals and narratological propositions. In our experimentation, we repeatedly iterated over their practical implementation, each addition seeming to enhance narrative coherence along some dimension.

A *narrative reading* $R(p_i) = r_i$ is a representation of the features most salient to the given passage. These representations can be arbitrarily complex, holding on to important details such as characters and plot events, and stylistic features such as tone or point of view, throwing away details determined to be irrelevant.

Additionally, an *explanation* function $X(p_i) = X(r_0, ..., r_{i-1}) = \chi_{p_i}$ maps chains of narrative readings to a plain text description χ_{p_i} of the narrative context leading up to passage p_i. A sequence of individual readings $r_0, ..., r_{i-1}$ forms a context chain for a passage at depth i

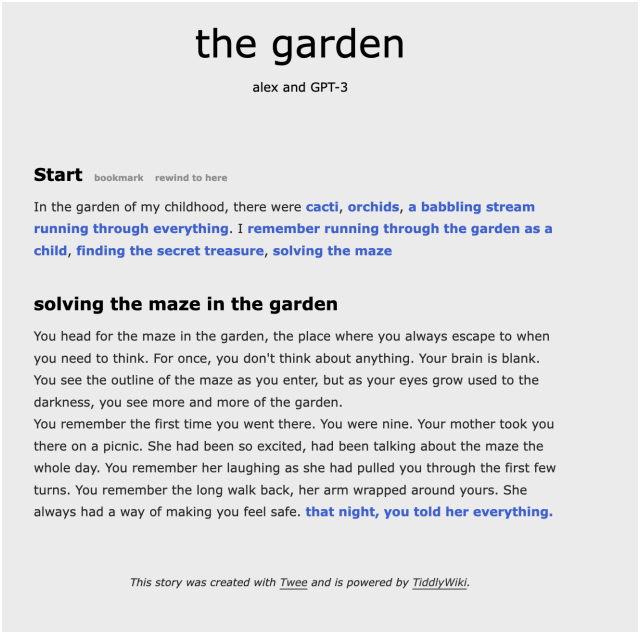

Figure 3: A screenshot of a playable Twine game produced by the system, corresponding to the tree in Figure 1

[7]Advanced Twine authors sometimes make use of macros or custom Javascript code to allow conditional or stateful storytelling. While language models have shown an ability to produce complex code snippets, for simplicity we have excluded passages containing these features.

along the minimum spanning tree path between the passage and *Start*, p_0. The necessity of a plain text description falls out of the paradigm of transfer learning: using an out of domain model on a sub—or alternate—domain[8]. By transforming our latent story representations into English, we can plug in any large pre-trained model into the generation pipeline, and ensure our code base is largely model agnostic. Additionally, the narrative descriptions produced by this intermediate stage of the system are highly interpretable, as they are in plain English.

Text Preprocessing and Model Training

We use these X and R (implementations detailed in the next section) to approximate a distribution $\mathcal{L}$ over the narratives we aim to generate. We would like to provide the narrative context to the model, alongside the title of the passage to be generated: $\chi_p|p^{title}$ in order to produce a p^{body}. Fan et al. (Fan, Lewis, and Dauphin 2019) and others have shown that preprocessing text—injecting external knowledge, adding start and end tokens, and generalizing named entities—aids in generating coherent stories.

Data Processing

To begin, we convert Twine binaries into *Twee 2*, to represent the interactive stories textually as in Figure 4. Gathering sufficient Twee-formatted stories required modifying the Twee source[9] to handle decompilation from games natively authored in the graphical Twine interface.

We split our collection of Twine stories into passages, excluding passages with macros and non-unicode characters. Next, we run our narrative reader and explanation functions on our corpus to produce textual descriptions of the story up until this point. Finally, we build $(prompt, response)$ pairs for each passage by appending to the readings unique start and end tokens unlikely to appear in the corpus (e.g. $start$:=<‖start‖>):

$$prompt(p) = start|\chi_p|p^{title}|begin$$
$$response(p) = p^{body}|end$$

Of the 512 Twine stories downloaded from *itch.io*, only 82 were Twee decompilable, producing 10,784 passages with a total of 11,098 embedded links.

Training

For each of the following experiments, we feed the processed training examples to the OpenAI *completion* endpoint with the default meta-parameters: *top_p*, *temperature*, and *best_of* all set to 1. Experimenting with a different *top_p* would retrieve multiple responses from the model, which we could pick from based on some retrospective evaluation. This is left for future work, as is providing the user with temperature (stochasticity) control. Each experiment uses GPT-3's largest available 175B parameter *davinci* model, resulting in three separate fine-tuned models. Fine-tuning involves a limited retraining of a neural network's weights (until recently, this typically meant a selective retraining of only the latter levels of a feed-forward network) (Howard and Ruder 2018; Lester, Al-Rfou, and Constant 2021). Fine-tuning through the API abstracts the typical machine learning workflow, which typically requires splitting data into training and test sets, selecting an optimization function, and tuning hyperparameters.

Running a fine-tune job is currently free and took under three hours for each tune. The total cost incurred from using the models did not exceed $40, demonstrating that using open-source language toolkits on top of fine-tuning existing large language models is a viable way to cheaply develop creative language applications.

[8]https://ruder.io/state-of-transfer-learning-in-nlp/

[9]https://github.com/tweecode/twee

Narrative Reader Development

In this section we present a series of increasingly sophisticated implementations of the two interpretation functions and example passages generated by each from the same *Start* passage (Figure 4).

```
:: Start
A tornado devastated town. Now you, Robert,
    and Maya are alone.
[[Head outside|take them outside]]
[[Continue to shelter at home.|tell them to
    stay inside]]
```

Figure 4: Our example input passage setting up a story about a tornado. Bracketed text indicates link text followed by the passage linked. Double colons :: represent a passage title.

0. Naive Method

The first question we wanted to address was the capability of a fine-tuned *davinci* model to generate grammatical Twine stories, regardless of their narrative coherence. For this baseline, $X(p)$ returns no narrative context and passage bodies are generated solely based on their corresponding title.

We found that fine-tuning the *davinci* model with this narrative reader generated Twine stories with correct Twine syntax with high reliability despite the medium size of our tuning dataset. This freed us to refocus on the quality of the generated narratives in subsequent definitions of R and X. The passages are sometimes formatted with rich text that we were surprised to see render appropriately through Twee. Unsurprisingly, the passages followed English grammar and storytelling conventions.

The passages generated naively were disjointed; Figure 5 shows subsequent passages completely diverging from the tornado narrative. The only times any narrative continuity was observed between passages was when the titles were extremely explicit. Additionally, passages switch perspective and verb tense, introduce new characters and do not refer back to previous characters.

```
:: tell them to stay inside
 <div id="day">
Day 49
</div>
You are in your car. You're on your way to
    pick up your daughter from daycare.
You're still not ready to take this step.
    The thought of leaving your child alone
    for a whole day makes you cold.
You have a lot of work to do to finish your
    project. You need to finish the sound
    engine, implement a pathfinding system,
    design the world generation...
Your free time is running short.
You need to finish your project. You need to
     do it soon.
[[ok now im ready]]

:: ok now im ready
 You look at your opponent. His eyes are
     locked on you, a mixture of fear and
     anticipation on his face. You draw your
      sword and run towards him.
You both charge at each other, swinging your
     swords wildly. You manage to land a few
     hits, but he parries most of them.
    Suddenly, he jumps back and cowers in
    fear. "You can't beat me", he cries. You
     look down and see the cause of his
    panic. You did a horrible job of
    cleaning the blood of your opponent off
    your sword. Horrible. You've been
    defeated by a sword-swinging mop.
[[You're done.|engstart]]
```

Figure 5: Two passages generated via the naive method did not show continuity with the initial tornado passage (Figure 4) or each other, but demonstrate adherence to Twine syntax.

1. Zero-Shot Summarization

One concept for our narrative reader function was to use GPT-3 as a zero-shot summarizer, defining X and R according to Equations 1 and 2.

$$R(p_i) = ZeroShot(p_i) \tag{1}$$

$$X(r_0, ..., r_{i-1}) = r_0|...|r_{i-1} \tag{2}$$

Using *davinci*, our *ZeroShot* summarizer produced accurate single sentence summaries of complex multi-character passages using the following prompt scaffold:

```
My job is to summarize short stories
    into a single sentence. Here is an
    example of a longer story:
"""
<Passage Text>
"""
This is my one-sentence summary:
"""
```

While this prompt engineering approach to summarization was promising with *davinci*, we did not have the resources to perform the summarization to produce $(prompt, response)$ pairs on our full dataset. *Davinci* is 10x the cost of the next largest *curie* model, which tended to simply produce the first sentence of the passage rather than an accurate summarization. In order to fine-tune the model with the ZeroShot narrative reading function, we would need to generate a summary for all 10,784 passages in our tuning dataset. Since OpenAI's pricing model charges per generated token, this was prohibitively expensive with the *davinci* model.

2. Entity Recognition

```
:: tell them to stay inside
 You go back out to the hallway, where your
     best friends are waiting.
"Are you crazy?" asks Robert. "You said it
    was a trap!"
"It probably is a trap," you admit. "But I
    can't just leave them in there."
Maya just looks at you with big, sad eyes.
"Please," she says. "just this once."
You sigh. "I don't know, Robert. I'm not
    going to be able to sleep if I know you
    two are in danger."
"Like you sleep anyway," says Robert. "What
    if something happens to us and you don't
     even know?"
"Can't we at least try to alert the [[unggoy
    ]] about the danger first?" you ask.

:: unggoy
 "Ain't you the one who told me to 'keep my
     head up lowbrows' and all that?"
[[Well, I'm just looking out for you.|
    another1]]
[[I have a solution.|another1]]
```

Figure 6: Passages generated via the entity recognition based reader from the same input passage as before.

For our next experiment, we define our narrative reading as

$$R(p) := (q_p, c_p, l_p) \tag{3}$$

where c was the set of characters referenced in the passage, l the set of locations, and q, the set of pronouns ("they", "you", "she", etc.). These were extracted from passage text using the python *spacy* library for part of speech tagging and a pretrained BERT model[10] for state of the art entity recognition (Devlin et al. 2019). The explanation function X counted the characters and locations in the context chain and wrote the most frequent (top 8) of each to short sentences of the form: "Locations mentioned: ..., Characters mentioned: ... Pronouns referenced: ...".

This model proved capable of generating passages that consistently referred back to previously mentioned locations and characters (Figure 6 mentions 'Maya' and 'Robert' from the previous passage), adding a sense of social and spatial

[10]https://huggingface.co/dslim/bert-base-NER

continuity to the stories. However, the passages generated still do not often follow the narrative, or if so, only loosely or by chance.

3. Event Extraction / Fabula

For the final experiment we report, we utilize the event extraction methodology of (Martin et al. 2018).

```
:: tell them to stay inside
 "It's dangerous out there!" you scream. You
      feel weak, but you manage to stand.
Maya and Robert look at you in shock. "You
    can't be serious!"
"I have to do something!" You say. "I have
    to protect everyone!"
You turn and rush back outside. You ignore
    the searing pain in your back and [[seek
     out the children]]

:: seek out the children
 You find a house with no roof and only
     walls left. What was once a house is
     now just a pile of rubble. You find a
     group of around 20 children huddled
     together. They are crying and shaking.
     What do you do?
[[you try to talk to them]]
[[you leave them alone]]
```

Figure 7: Passages generated via the fabula reader.

We expand on our previous narrative reader (Equation 3) with the following:

$$R(p) := (q_p, c_p, l_p, e_p) \tag{4}$$

where e_p is an ordered list of (s, v, o) triples extracted from the passage.

X is defined similarly to the previous section, with the addition of a bulleted list of events, written in plain text as in Figure 8.

Thanks to modern NLP libraries, this reader architecture is cheap to run even on a CPU. Here, it does not directly predict event $e_{i+1}$from e_i, as in (Martin et al. 2018), who use a further LSTM model to expand predicted events into sentences. Rather, the pretrained language completion model, when fine-tuned on a corpus of $(prompt, response)$ pairs that include this event list, is expected to jointly learn next-event likelihood alongside Twine grammar.

To reduce the prompt length (necessitated by the discussion in the section *Formulating the Generation Problem*), we apply the following crude reduction algorithm to produce a new event list for X. We chose 32 as a constant following initial tests and intuition.

```
func reduce(events):
  while length(events) > 32:
    events = events[::2] // every other event
```

The effects of this necessary reduction step means that important events may be omitted from the fabula presented as context to the model. It is an active area of NLP research to find mechanisms to reduce the length of the prompt given to a text generation model while discarding only less important details. One may for example imagine a system for sifting events based on perceived relevance to the current passage.

```
<|begin|>Pronouns referenced: you, yourself,
     anyone, and everyone. Mentioned
    Locations: None. Mentioned People: Katie
    . Preceding Events:
* you packed bag
* you asked mom
* you tried not to look at anyone in the
    eyes
* you shove bag
* you use body weight
* you hear sound
* it gives way
* plane begins to move
* you feel hand<|title|>:: offering support
    <|start|>
```

Figure 8: An example prompt produced through the fabula reader.

However, the passages generated by this version of the model seem to appropriately introduce new characters and back-reference existing ones ('Maya', 'Robert' in Figure 6). It tends to stay on topic, with the deeper passage in Figure 6 appropriately following the tornado theme without overtly mentioning it. This example also demonstrates the model's tendency to follow the Twine convention of presenting branch points that represent straightforward character actions at the end of a passage, often weighted by considerations of exploration and morality.

Discussion

Generating Twine Stories

The project was originally motivated by asking if Twine *syntax* could be modeled in a straightforward way, and when GPT-3 was brought on board to bootstrap the process, it became clear that syntax was a trivial problem, inter-passage *fluency* was very high, and intra-passage *coherence* was the problem to solve. This required an exploration of the story generation literature. In iterating on our narrative reader formulation, we demonstrated that multiple theories of narrative can be substituted into this generative framework and that pairing the interpretive system with prompt-engineering is a fruitful methodology for language model based interactive story generation.

The Authoring Experience

Using the tool to write interactive fiction is itself a fun, game-like experience. The feeling of having primed the model to generate something interesting and coherent within

the given narrative context is uniquely pleasurable, somewhat like having succeeded in having an interesting conversation with a stranger, resulting in the discovery of new opportunities for directions to take your writing. High coherence is not the only goal in building the model however; unexpectedness is also key (Table 1). The hope is that a tool like this might be useful for professional writers as well as novices looking for inspiration, writing assistance, and for those who would rather experience a custom Twine story than write one.

Passage Coherence	User Response
Incoherent; random	Annoyance, Confusion
Mildly Coherent; unexpected	Defamiliarization, Curiosity
Coherent; unexpected	Amusement, Joy, Ideation
Coherent; expected	Boredom

Table 1: Based on informal testing with 6 users, we identified a few common user responses to the various levels of coherence the models produce.

Limitations and Future Work

As we've established a method for using automated narrative readings to guide narrative text generation, it is clear that there are many additional theories of narrative that could be used instead of the fabula + entity method we arrived at. Such possible readings include formalist theories such as the SIG (Elson 2012), reader-response models (Castricato et al. 2021), or frame-based computational readings (Peng et al. 2021). Our earlier experimentation with other unsupervised summarization methods did not yield promising results, but a reviewer points out that this should be formally evaluated, and recent non-GPT abstractive summarization techniques such as those found in (Alomari et al. 2022) may suffice [11]). The fabula-based event structure we arrived at does not encapsulate a total understanding of narrative and we look forward to experimentation with many other representation formats.

Mawhorter et, al. has introduced the theory of *choice poetics*, "a formalist framework for understanding the impact of narrative choices on the player experience via their options, their outcomes, and how those relate to player goals" (Mawhorter et al. 2018). Its application to this work seems clear; we might model the sequence of choices that led to the present passage as an additional component of the reading.

Recent work has shown complex world state may be learned from a story via semantic role labeling, which can be used to populate knowledge graphs about the story world, and then stories can be generated such that they reduce differences between the current story world graph and a given goal graph (Peng et al. 2021). This approach is an extremely promising approach to story generation. Integrating a similar knowledge graph parsing approach into our architecture is an obvious next step, as is integrating a symbolic planner to reason over inferred narrative semantics.

Additionally, we are working towards a front-end interface that will allow for a more seamless revision-iteration loop for more lively or dynamic interaction with the written text. This will enable us to conduct a user study to assess the quality of the written work and the experience of working with the tool.

Finally, the evaluation of story text coherence beyond qualitative analysis needs to be addressed. Story coherence is generally assessed with human evaluation, though automated analysis of new character introduction or scene/object permanence may be possible. Without these evaluations, we are unable to make objective statements about the increase in narrative coherence we see from the baseline narrative reader to the fabula approach.

[11] such as RefSum (Liu, Dou, and Liu 2021) or gensim's TextRank summariser (`tinyurl.com/2s35hcr4`

References

[Alomari et al. 2022] Alomari, A.; Idris, N.; Sabri, A. Q. M.; and Alsmadi, I. 2022. Deep reinforcement and transfer learning for abstractive text summarization: A review. *Computer Speech Language* 71:101276.

[Bal and Van Boheemen 2009] Bal, M., and Van Boheemen, C. 2009. *Narratology: Introduction to the theory of narrative*. University of Toronto Press.

[Bamman, Underwood, and Smith 2014] Bamman, D.; Underwood, T.; and Smith, N. A. 2014. A bayesian mixed effects model of literary character. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 370–379.

[Barros et al. 2019] Barros, G. A. B.; Green, M. C.; Liapis, A.; and Togelius, J. 2019. Who killed albert einstein? from open data to murder mystery games. *IEEE Transactions on Games* 11(1):79–89.

[Black and Wilensky 1979] Black, J. B., and Wilensky, R. 1979. An evaluation of story grammars. *Cognitive science* 3(3):213–229.

[Brown et al. 2020] Brown, T. B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

[Browne and Maire 2010] Browne, C., and Maire, F. 2010. Evolutionary game design. *IEEE Transactions on Computational Intelligence and AI in Games* 2(1):1–16.

[Calderwood et al. 2020] Calderwood, A.; Qiu, V.; Gero, K. I.; and Chilton, L. B. 2020. How novelists use generative language models: An exploratory user study. In *HAI-GEN+ user2agent@ IUI*.

[Castricato et al. 2021] Castricato, L.; Biderman, S.; Cardona-Rivera, R. E.; and Thue, D. 2021. Towards a formal model of narratives.

[Chambers and Jurafsky 2008] Chambers, N., and Jurafsky, D. 2008. Unsupervised learning of narrative event chains. In *Proceedings of ACL-08: HLT*, 789–797.

[Cho et al. 2018] Cho, W. S.; Zhang, P.; Zhang, Y.; Li, X.; Galley, M.; Brockett, C.; Wang, M.; and Gao, J. 2018. Towards coherent and cohesive long-form text generation. *arXiv preprint arXiv:1811.00511*.

[Compton, Kybartas, and Mateas 2015] Compton, K.; Kybartas, B.; and Mateas, M. 2015. Tracery: An author-focused generative text tool. In Schoenau-Fog, H.; Bruni, L. E.; Louchart, S.; and Baceviciute, S., eds., *Interactive Storytelling*, 154–161. Cham: Springer International Publishing.

[Cook, Colton, and Gow 2016] Cook, M.; Colton, S.; and Gow, J. 2016. The angelina videogame design system—part i. *IEEE Transactions on Computational Intelligence and AI in Games* 9(2):192–203.

[Cook 2015] Cook, M. 2015. Formalizing non-formalism: Breaking the rules of automated game design. *FDG*.

[Cook 2021] Cook, M. 2021. The road less travelled: Trying and failing to generate walking simulators. *arXiv preprint arXiv:2104.10789*.

[Côté et al. 2018] Côté, M.-A.; Kádár, A.; Yuan, X.; Kybartas, B.; Barnes, T.; Fine, E.; Moore, J.; Hausknecht, M.; Asri, L. E.; Adada, M.; et al. 2018. Textworld: A learning environment for text-based games. In *Workshop on Computer Games*, 41–75. Springer.

[Devlin et al. 2019] Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding.

[Duplantis et al. 2021] Duplantis, T.; Karth, I.; Kreminski, M.; Smith, A. M.; and Mateas, M. 2021. A genre-specific game description language for game boy rpgs. In *2021 IEEE Conference on Games (CoG)*, 1–8.

[Elson 2012] Elson, D. K. 2012. Detecting story analogies from annotations of time, action and agency. In *Proceedings of the LREC 2012 Workshop on Computational Models of Narrative, Istanbul, Turkey*, 91–99.

[Fan et al. 2020] Fan, A.; Urbanek, J.; Ringshia, P.; Dinan, E.; Qian, E.; Karamcheti, S.; Prabhumoye, S.; Kiela, D.; Rocktaschel, T.; Szlam, A.; et al. 2020. Generating interactive worlds with text. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 1693–1700.

[Fan, Lewis, and Dauphin 2019] Fan, A.; Lewis, M.; and Dauphin, Y. 2019. Strategies for structuring story generation. *arXiv preprint arXiv:1902.01109*.

[Friedhoff 2013] Friedhoff, J. 2013. Untangling twine: A platform study. In *DiGRA conference*.

[Garbe et al. 2019] Garbe, J.; Kreminski, M.; Samuel, B.; Wardrip-Fruin, N.; and Mateas, M. 2019. Storyassembler: an engine for generating dynamic choice-driven narratives. In *Proceedings of the 14th International Conference on the Foundations of Digital Games*, 1–10.

[Guan et al. 2021] Guan, J.; Mao, X.; Fan, C.; Liu, Z.; Ding, W.; and Huang, M. 2021. Long text generation by modeling sentence-level and discourse-level coherence. *arXiv preprint arXiv:2105.08963*.

[Harvey 2014] Harvey, A. 2014. Twine's revolution: Democratization, depoliticization, and the queering of game design. *G— A— M— E Games as Art, Media, Entertainment* 1(3).

[Howard and Ruder 2018] Howard, J., and Ruder, S. 2018. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*.

[Hua and Raley 2020] Hua, M., and Raley, R. 2020. Playing with unicorns: Ai dungeon and citizen nlp. *DHQ: Digital Humanities Quarterly* 14(4).

[Kostka et al. 2017a] Kostka, B.; Kwiecieli, J.; Kowalski, J.; and Rychlikowski, P. 2017a. Text-based adventures of the golovin ai agent. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, 181–188.

[Kostka et al. 2017b] Kostka, B.; Kwiecieli, J.; Kowalski, J.; and Rychlikowski, P. 2017b. Text-based adventures of the golovin ai agent. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, 181–188. IEEE.

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

[Lester, Al-Rfou, and Constant 2021] Lester, B.; Al-Rfou, R.; and Constant, N. 2021. The power of scale for parameter-efficient prompt tuning.

[Liu and Chilton 2021] Liu, V., and Chilton, L. B. 2021. Design guidelines for prompt engineering text-to-image generative models.

[Liu, Dou, and Liu 2021] Liu, Y.; Dou, Z.-Y.; and Liu, P. 2021. Refsum: Refactoring neural summarization.

[Martin et al. 2018] Martin, L.; Ammanabrolu, P.; Wang, X.; Hancock, W.; Singh, S.; Harrison, B.; and Riedl, M. 2018. Event representations for automated story generation with deep neural nets. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

[Mawhorter et al. 2018] Mawhorter, P.; Zegura, C.; Gray, A.; Jhala, A.; Mateas, M.; and Wardrip-Fruin, N. 2018. Choice poetics by example. *Arts* 7(3).

[Meehan 1977] Meehan, J. R. 1977. Tale-spin, an interactive program that writes stories. In *Ijcai*, volume 77, 9198.

[Pell 1992] Pell, B. 1992. Metagame: A new challenge for games and learning. *Heuristic programming in artificial intelligence*.

[Peng et al. 2021] Peng, X.; Xie, K.; Alabdulkarim, A.; Kayam, H.; Dani, S.; and Riedl, M. O. 2021. Guiding neural story generation with reader models.

[Pérez and Sharples 2001] Pérez, R., and Sharples, M. 2001. Mexica: A computer model of a cognitive account of creative writing. *Journal of Experimental & Theoretical Artificial Intelligence* 13:119 – 139.

[Pichotta and Mooney 2016] Pichotta, K., and Mooney, R. 2016. Learning statistical scripts with lstm recurrent neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30.

[Porteous 2016] Porteous, J. 2016. Planning technologies for interactive storytelling. In *Handbook of Digital Games and Entertainment Technologies*. Springer.

[Reynolds and McDonell 2021] Reynolds, L., and McDonell, K. 2021. Prompt programming for large language models: Beyond the few-shot paradigm. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, 1–7.

[Riedl and Young 2010] Riedl, M. O., and Young, R. M. 2010. Narrative planning: Balancing plot and character. *Journal of Artificial Intelligence Research* 39:217–268.

[Roemmele 2018] Roemmele, M. 2018. *Neural Networks for Narrative Continuation*. Ph.D. Dissertation, University of Southern California.

[Schaul 2013] Schaul, T. 2013. A video game description language for model-based or interactive learning. In *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, 1–8. IEEE.

[Summerville et al. 2018] Summerville, A.; Martens, C.; Samuel, B.; Osborn, J.; Wardrip-Fruin, N.; and Mateas, M. 2018. Gemini: bidirectional generation and analysis of games via asp. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 14, 123–129.

[Theune et al. 2003] Theune, M.; Faas, S.; Nijholt, A.; and Heylen, D. 2003. The virtual storyteller: Story creation by intelligent agents. In *Proceedings of the Technologies for Interactive Digital Storytelling and Entertainment (TIDSE) Conference*, volume 204215, 116.

[Theune et al. 2004] Theune, M.; Rensen, S.; op den Akker, R.; Heylen, D.; and Nijholt, A. 2004. Emotional characters for automatic plot creation. In *International Conference on Technologies for Interactive Digital Storytelling and Entertainment*, 95–100. Springer.

[Treanor et al. 2012] Treanor, M.; Blackford, B.; Mateas, M.; and Bogost, I. 2012. Game-o-matic: Generating videogames that represent ideas. In *Proceedings of the The Third Workshop on Procedural Content Generation in Games*, PCG'12, 1–8. New York, NY, USA: Association for Computing Machinery.

[Young et al. 2013] Young, R. M.; Ware, S. G.; Cassell, B. A.; and Robertson, J. 2013. Plans and planning in narrative generation: a review of plan-based approaches to the generation of story, discourse and interactivity in narratives. *Sprache und Datenverarbeitung, Special Issue on Formal and Computational Models of Narrative* 37(1-2):41–64.