Procedural Generation of Interactive Stories using Language Models

Jonas Freiknecht
University of Mannheim
Department of Computer Science IV
Mannheim, Germany

Wolfgang Effelsberg
University of Mannheim
Department of Computer Science IV
Mannheim, Germany

ABSTRACT

In this paper we introduce an architecture, an implementation and an evaluation of a system for the automatic creation of interactive stories for games. Our goal is to algorithmically create a branched story for the entire game; in each game run a different variant is generated. The architecture uses natural language processing (NLP) to generate meaningful stories. For NLP we use a statistical language model based on a neural network (Generative Pretrained Transformer, GPT-2). The basic architecture generates stories with too many characters which tend to get incoherent for longer texts, so we add a component restricting the number of persons and improving the consistency. The system is initialized with a handwritten game introduction that defines the main characters and the inventory; it also sets the goals for the game. From that text the remainder of the game story is generated algorithmically. We have fully implemented our system, and we report initial, encouraging experimental results.

CCS CONCEPTS

• **Information systems → Multimedia content creation**; • **Software and its engineering → Interactive games**.

KEYWORDS

procedural content generation, interactive stories, language models, natural language processing

ACM Reference Format:
Jonas Freiknecht and Wolfgang Effelsberg. 2018. Procedural Generation of Interactive Stories using Language Models. In *Proceedings of the International Conference on the Foundations of Digital Games (FDG '20).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

The creation of story generating systems is a long-standing field in the domain of procedural content generation for games (PCG-G). A story generating system is designed to generate coherent, credible, and dramatically meaningful narratives [1]. This is a task that even human authors fail to accomplish from time to time, often due to budget and workload. These last two stumbling blocks are a common reason to explore procedural storytelling, even if the temporal aspect is always under discussion [23]. It is not only useful for the main plot of a game, but also for subplots. In this way the authors retain sovereignty over the main storyline so that the course of play remains known for the entire development team.

Another good argument to deal with procedural storytelling is the creativity that an algorithm generates. People often write about their personal experience or modify stories that they have read themselves. In contrast, an algorithm based on the vast set of available literature and language models draws from a nearly infinite pool of rules, possibilities, phrasing and connections. So, instead of reflecting the opinions, prejudices, tastes or moods of a single author, the algorithm returns a mixture of its training data. However, even these can be biased and violate ethical principles.

The use of stories in the domain of games, may they be handwritten or generated, is manifold: Some games only aim to have a basic story to convey mood and setting [4], so that the player can put herself in the position of the protagonist. Others live from diverse, branched stories that take the decisions of the players into account and might even have different endings. In total, we identified five different classes of games, characterized by the nature of their story:

- games without story (e.g., *Tetris*, *Rollercoaster Tycoon*),
- games with a simple background story, to convey mood and setting (e.g., *Need for Speed*, *Street Fighter II*),
- games with complex but fixed stories (e.g., *Final Fantasy VII*, *Legend of Zelda*),
- games with interactive stories that convey the feeling that the player can influence the course of the game (e.g., *Borderlands*, *Mass Effect*),
- games with interactive stories whose course and ending can be actively influenced by the player (e.g., *Life is Strange*, *Detroit: Become Human*).

This work focuses on the last category, namely on the generation of interactive stories that the player can influence during gameplay. Our motivation is that text adventures are currently experiencing a revival through the generation via language models. Similar to Nick Walton's *AI Dungeon 2* [26] and Nathan Witmore's *GPT-2 Adventure* [27] we make use of statistical language models to generate a unique story with each new run of the generator.

We distinguish ourselves from the approaches mentioned above by providing a reasonable set of actions from which the player can choose to continue the story. Presenting three methods of action generation, we compare the results in regard of creativity, controllability, coherence, mood, and variety.

In addition, we evaluate two processes of story generation in terms of feasibility and performance: one in which the generator

FDG '20, September 15–18, 2020, Malta

ACM ISBN unknown…$15.00
https://doi.org/10.1145/1122445.1122456

reacts spontaneously to the previously executed action, and the other in which all actions and their consequences are precalculated. We examine how to control the development of the stories by managing the state, progress and inventory outside the generator and by placing prefabricated text modules in the generated paragraphs to keep objects, places and people in the storyline.

2 RELATED WORK

Yao et al. describe the story generation process as a two-steps process, planning a sequence of events (story planning) and formulating the actual text that the player sees (surface realization) [30]. In fact, the complexity of both steps is divided into many sub-processes, especially with adaptive approaches. *Façade*, as an example, is a system that builds dramatic tension by concatenating events and by reacting carefully on the player's actions [15].

One of the most obvious approaches is to use prefabricated grammars, often in combination with manually generated building blocks (templates). A famous representative is *Tracery* by Compton et al. [6]. Its core element is a list of production rules, which create sentences by substitution of symbols. The concatenation of those sentences results in a story. The advantage of this approach is that the computer has maximum control over the text to be generated. The disadvantage is also obvious: Creating templates and complex rules is time-consuming. Furthermore, simulated creativity can result in a creativity that differs from what a human would expect [5].

Grammars can (but do not have to) go hand in hand with planning algorithms. Those generate stories under the assumption that stories consist of a sequence of actions that work towards a predefined goal (see Chapter 7 in [22]). A disadvantage of using planners (often based on the Planning Domain Definition Language (PDDL) standard) is that a planner works strictly towards the given goal. An interactive story, on the other hand, can also be worth playing if it contains detours and setbacks or if the protagonist has to completely reorient herself. All these events speak against the nature of a planner designed for optimization.

Neural network based story generators were frequently discussed in the last years and also operate in the domains *story planning* and *surface realization*. L. J. Martin et al. trained two neural networks, *event2event* and *event2sentence*, to generate chains of events maintaining the semantic meaning of the story and translating the event chain to readable and understandable texts [14].

Yao et al. propose a hierarchical generation framework that is not limited to any domain. Generating the storyline first, it extends this storyline in a second step to a story. The interesting part is that they compare a full storyline planning and a subsequent story generation to a partial storyline planning with an interlacing story generation [30].

Fan, Lewis, and Dauphin show a four-step coarse-to-fine model which returns a structured action plan to an arbitrary story prompt. The action plan containing placeholders for entities is assembled to a story and placeholders are replaced by specific references [8].

A holistic approach to narrative continuation is presented by Roemmele. Besides traditional NLP techniques, she proposes to use neural networks to solve various tasks including story ending prediction and she presents an automated evaluation of stories based on specific linguistic metrics [21].

Cychosz et al. created DINE (Data-driven Interactive Narrative Engine), an authoring platform for interactive fiction primarily addressing the authors' tasks, instead of the players'. Their approach resembles our method but instead of generating new passages, DINE classifies the user input and maps it to professionally, pre-authored texts [7].

Angela Fan et al. explore the generation of game environments by machine learning using crowd-sourced data from the multiplayer text adventure game environment *LIGHT* [25]. They show how to construct cohesive arrangements of locations, characters, and objects and emphasize the diversity and variety of their results [9].

In this paper we would like to address possible shortcomings regarding creativity in grammars and planning algorithms by statistical language models. Such models provide information about the probability of the correlations of word sequences. In recent years language models have appeared, such as Google's *Bidirectional Encoder Representations from Transformers* (BERT), Facebook's *Cross-lingual Language Model Pretraining* (XLM), or OpenAI's *Generative Pretrained Transformer* (GPT and GPT-2).

Based on *GPT-2*, created by Radford et al. [19], and trained and published by Hugging Face [29], Whitmore recently released a text adventure using GPT-2's strength to predict the next token(s) in a sentence to continue a story [27]. The fact that previously played passages form the input text for the following paragraphs creates a continuing context, which is a key element in creating procedural stories [11]. We base our work on *GPT-2 Adventure* and extend it as discussed in the following section. Tambwekar et al. note the lack of player guidance when using language models for story generation and present a reward-shaping system which re-trains a language model in order to reach a given goal [24]. Language models calculate the probability of word or character sequences $p(x)=p(x_1,x_2,...,x_n)$, frequently in an autoregressive manner $p(x)=p(x_1)*p(x_2|x_1)*p(x_3|x_2,x_1)$.

The training process is unsupervised and allows the use of very large amounts of data. The application of models trained in this way helps with various tasks, such as text translation, summaries, question answering, and text generation (see Figure 1). The following paragraph shows how a hybrid approach of templating, language models and related NLP techniques can lead to new, interesting interactive narratives. It also discusses how the weaknesses of the respective approaches (predictability, limited domain and uncontrollability) can be circumvented.

3 A HYBRID APPROACH OF LANGUAGE MODELS AND STATE MANAGEMENT

We use the creative possibilities of language models in our approach and add controls to allow the player to interact with the story generator.

In the following sections we will look at the three phases of initialization, runtime, and ending as outlined in Figure 2. Subsequently, we discuss the challenges of coherence, action generation and performance of the story generator.

The creation of story generation systems is a long-standing field of procedural content generation (PCG). A story generation system is designed to generate coherent, credible, and dramatically meaningful narratives. A

Input symbols $s_1,\ldots,s_{n-k-1}$

"Story" system uses storytelling elements designed to drive a particular purpose, and is the most popular type of PCG. There are also "Story Templates" which use procedural story generation techniques to achieve a particular goal. It is recommended to use a story generation system for all your PCG content. The system used by our story generation system is a system called the K-Pole, which is described in detail on its page on the wiki. The system can be defined by the type of elements used in narrative, or the type of gameplay elements used.

Sampled output symbols $s_{n-k},\ldots,s_n$

Figure 1: The introduction of this paper as generated by a GPT-2 model.

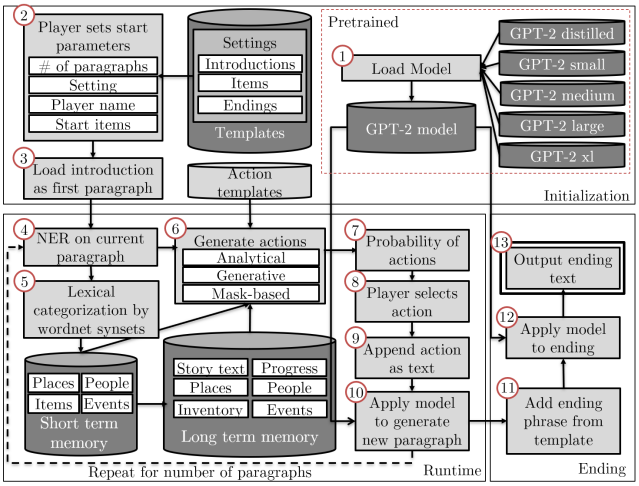

Figure 2: The hybrid story generation approach is split up into the three steps initialization, runtime, and ending.

3.1 Initializing Language Model and Templates

The initialization phase is a two-step process. The model is loaded before the generator is started (step 1 in Figure 2). The player can choose between different pre-trained models which differ mainly in the number of their parameters (82 to 1558 million). The models will be examined again later with regard to diversity and speed of application. We did not retrain a model as done by Whitmore [27].

In a second step of the initialization (step 2 in Figure 2) the player selects various parameters such as the number of paragraphs the narrative should contain, a player name, optional party members, and a set of items for the game. Afterwards, one of multiple possible introductions is loaded from the settings object and presented to the player (step 3 in Figure 2).

Since the introduction is the basis of the story, it should contain as much information about characters, locations and items in the story as possible. Only with a sufficiently large amount of information can the language model establish a context and later credibly continue the story. In addition to printing out the introduction as a first paragraph, the items of the game are added to a visible inventory so that the player knows what she is carrying around.

3.2 Course of the Plot

The Runtime phase is a repetitive process that continues until the player has completed a number of paragraphs of her choice. It starts with a named entity recognition (NER) on the last paragraph - which is the introduction in the first run. Entities are not only recognized but also categorized with *WordNet Synsets* in order to select the group of nouns that the player can physically interact with [10] (step 4 and 5). Those categories are

- noun.animal (nouns denoting animals),
- noun.artifact (nouns denoting man-made objects),
- noun.food (nouns denoting foods and drinks),
- noun.plant (nouns denoting plants), and
- noun.object (nouns denoting natural objects (not man-made)).

This avoids later actions like *Take love* or *Talk to faith* in the action generation. Such actions may seem creative in some places, but more often they disturb the flow of the game because they are confusing or inappropriate.

Those entities are stored in a *short term memory* with a life-cycle spanning the current paragraph. They are also transferred to a *long term memory* to be referenced later. This allows the generator to relate to past events or past conversations and hence add control and consistency to the stories.

3.2.1 Redundancy Avoidance of Entities. Entities can occur more than once with different names in a paragraph. The NER process will detect the expressions *Elisabeth* and *Queen* from the sentence *Elizabeth II is Queen of the United Kingdom*. For a human being it is clear that both terms refer to the same person but for a NER component it is not. Thus, the story generator will treat *Elisabeth* and *Queen* as separate entities and might thus simultaneously generate the actions *Talk to Queen* and *Talk to Elisabeth*. This leads to an inconsistency in the story and disrupts the flow of reading, so expressions that refer to the same entity must be reduced to a single expression. We use *coreference resolution* to detect such expressions and keep the first expression found while discarding all others. We used the implementation by Hugging Face as presented by Wolf in [28]; it is based on word vectors and neural networks and detects references reliably.

3.2.2 Action Generation. As mentioned in the introduction we implemented three methods to generate actions that the player can use to continue the story (see Figure 3 and step 6 in Figure 2).

Such an action is not only a verb and an object but has multiple attributes as shown in Listing 1. The *type* informs about the named entity type, the *action* about the action type, the *sentence* about the full sentence based on a template as it is added to the story if the

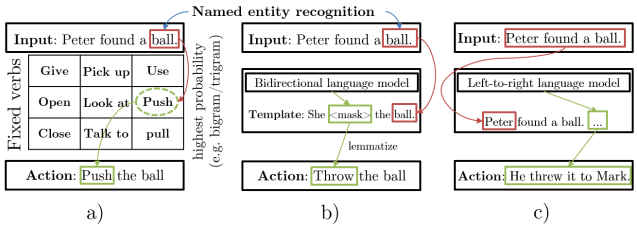

Figure 3: The three different ways to offer fixed actions to continue a story: The analytical (a), the mask-based (b) and the generative approach (c).

Listing 1: The action object potion with its individual properties

```
{"name":"potion", "type":"item", "action":"use", "
    sentence":"Peter bravely used the potion
    without hesitation.", "simple": "Peter used
    the potion.", "probability":0.76}
```

player picks this action, the *simple sentence* contains a simplified version of the template sentence, and the *probability* is the sentence probability calculated using the underlying language model.

Approach a) in Figure 3 uses NLP to analyse the previous story paragraph and offer senseful actions such as *take*, *pick up*, *visit*, *use* or *push* from a static list in combination with an item, person, or place.

Since the actions are derived from the recognized entities, a paragraph with many entities can lead to an equally high number of actions. Therefore prioritizing the actions is useful. We do this in step 7 by checking the actions for plausibility. This check computes the probability of occurrence of a combination of the action and the object in question. Several approaches were considered. First, we evaluated ngrams (bi- and trigrams) to calculate the probability *p* of an action, e.g., *Peter used a potion* would be much more likely than *Peter talks to a potion*. The trained ngram model was based on the *Reuters corpus* containing 10.788 news documents with 1.3 million words [12]. In many cases, however, even simple combinations such as *takes an apple* could not be found in our generated list of trigrams. Hence, we evaluated semantic similarities in the second place using word vectors to calculate probabilities. Here, too, the result was unsatisfactory, since the similarity often could not give any information about the connection between verbs and nouns. For example, the combination of *push* and *love* was rated higher than *take* and *apple*. The probability calculation based on the GPT-2 model finally brought convincing results. Calculating the loss as an error of the model for the given sentence helps to figure out which sentence is more likely to be found in the training corpus of GPT-2. We have observed that the use of a simple sentence reduced to subject, predicate and object, and thus linguistically normalized, provides a better comparable probability than the embellished sentences based on the templates.

The approach b) is also based on the extracted entities of the previous paragraph. Here, a simple sentence is formed from the combination of a personal pronoun and the object, in which the position of the verb is masked. Using a bidirectional language model (such as BERT) allows to replace this placeholder by the statistically most reasonable verb.

Gender	Sequence	Score	Lemmatized verb
male	He catches the ball.	0.049	catch
male	He dropped the ball.	0.033	drop
male	He threw the ball.	0.026	throw
female	She dropped the ball.	0.051	drop
female	She throws the ball.	0.039	throw
female	She threw the ball.	0.035	throw

Table 1: Results of the masked sentence "He <mask>the ball." and "She <mask>the ball." including the probabilities and the lemmatized verb.

Table 1 shows sequences that fill the masked field in "He <mask>the ball." and "She <mask>the ball." Although it cannot be explicitly requested that a verb is used for the mask, linguistically there is rarely another option for the model. Two particularities stand out: First, the chosen personal pronoun plays a role in the choice of the verb. Thus, the probabilities differ for *he* and *she* as the first word in the masked phrase. Secondly, it can happen that the same verb is used again in a different tense. Thus lemmatizing the verbs is recommended. Here one can clearly see the gender bias, which is based on the language model's training data. Bordia and Bowman state that this bias can even be amplified in the models and propose methods to measure and minimize this bias [3]. The probability used to decide which verb fits best comes with resolving the masked word so that is does not have to be calculated manually as it is done in approach a). Control over items while using the actions *use*, *take*, and *combine* can be maintained by using *WordNet Synsets* and checking the chosen verb for synonyms. If the verb is *take* or a synonym, the object can be added to the player's inventory. Usage of an item is processed analogously. Combining items is a special case and can once again be achieved by masking the target object in a sentence. A combination is always preceded by two objects in the inventory. Masking is used to determine the statistically most probable result of the combination using BERT. Let us look at an example.

Input: He combined iron and hammer to receive a <mask>.

- **Output 1:** blade
- **Output 2:** handle
- **Output 3:** shield

The items *iron* and *hammer* are removed from the inventory and the masked object, determined by the BERT model (blade, handle, or shield) is added in return.

Approach c) uses the same language model that is deployed for the paragraph generation and to create the subsequent sentence using the previous paragraph as input (see c) in Figure 3). The result is a freely formulated sentence which can not be fit to the attributes of an action object (see Listing **??**) without a huge effort, because the generator does not generate sentences in the pattern of *subject, verb, object*. As a consequence, control over using, taking and combining items in the player's inventory is lost. Furthermore,

the following example shows that actual actions are generated only in rare occasions.

Input: The conference was over and Diana was on her way home. She...

- **Output 1:** was in charge of organizing the conference, taking the oath of office.
- **Output 2:** walked onto the street and then the entire group of people had heard about the situation.
- **Output 3:** turned around and asked me if I had any reservations.

3.2.3 Generating new Paragraphs. If the player has chosen an action to perform in step 8, its full clause is appended to the text of the story (step 9). This new text forms the basis for the next generation process. When generating texts using GPT-2 in step 10, the input of a source text is crucial. Too short input does not give the model enough context to generate a credible continuation. An input that is too long increases the memory consumption when calculating the subsequent text and also the duration of the generation process. In this evaluation the previous 300 characters were used. Steps 4 to 10 are now repeated until the initially set number of paragraphs is reached.

3.3 Coming to an End

If this is the case, a template for introducing the **Ending** phase is initialized. In step 11, an analysis determines the sentiment of the last *n* sentences. Based on this, a template of the respective mood (positive or negative) is loaded, and placeholders for the protagonist are filled (step 12). In a last step the generator creates a final paragraph based on the template before the game ends. We have found that an ending is more believable if there are only one or two paragraphs following the template-based ending. Several paragraphs would give the impression that the story is to be continued after the end, and the effect of the prepared paragraph heading for an end fades away. Mostafazadeh et al. contribute the *Story Cloze Test* which is able to determine the end to a four-sentence story based on the *ROCStories* corpus, a collection of 50.000 commonsense stories [17]. This test allows a precise evaluation of the quality of the generated ends.

4 IMPLEMENTATION, OBSERVATIONS AND ENHANCEMENTS

This section focuses on the observations we made while implementing and testing our approach. We discuss the technical realisation, performance, character diversity, coherence, and the alignment of sentiments. The exemplary output of a story generation after all enhancements discussed in this section will also be presented (see Figure 4).

4.1 Technical Realization and Performance

We did two separate implementations to generate a story based on language models. On the one hand, Twine was used, a tool that allows to write interactive, non-linear stories and make them available to the players on a website. When using Twine, it is essential that all branches of a story are precalculated since the game is compiled and therefore cannot be updated while playing. The advantage is obvious: The entire calculation can be done on a suitable system, and the generated story can be played on any device. If desired the game can be passed on, and the limitation that language models usually do not provide a reproducible output can be avoided. The disadvantage is that all paths of a story have to be generated, no matter whether the player enters them or not. If only one action is offered to the player, a total of num_paragraphs = story_depth paragraphs have to be calculated. If more than one action is offered, num_paragraphs equals to:

$$\text{num\_paragraphs} = \frac{\text{num\_actions}^{\text{story\_depth}} - 1}{\text{num\_actions} - 1} \tag{1}$$

The variable story_depth corresponds to the number of paragraphs a player sees to get to the end of the story. Equation 1 shows the exponential character of the calculation when increasing the possible actions per paragraph. When measured on an Intel i7 processor and a GeForce GTX 1080 TI with 11 GB of RAM the generation took approximately 10 seconds for one single paragraph (including NLP preprocessing, applying templates, and generating new texts; based on the GPT-2 small model).

The second implementation reacts to the user input and calculates the subsequent paragraphs and actions on the fly, during the game. Since there is no precalculation of possible subsequent paragraphs in this approach, the depth of the story corresponds to the number of paragraphs to be calculated. This eliminates the exponential growth of precalculation. The disadvantage is that the device on which the story is played must have the capability to generate the text in real-time using a neural network-based language model.

Even with suitable hardware, the generation of a new paragraph takes some time, so that the flow of gameplay may be disturbed. Furthermore, this requirement prevents the possibility to play the game on mobile devices, unless there is a constant connection to a corresponding cloud service. For these two reasons we refer our further considerations to the precalculated variant.

4.2 Character Diversity

We have observed that the number of characters increases with the number of paragraphs played. Recurring characters are extremely rare, which has a negative influence on the story's context. Thus, there is no long-term relationship between protagonist and antagonist, lasting love affairs or friendships. Except for the main character named in each paragraph and the two optionally pre-defined party members, no relationship can be built up with another character, as these usually have a life span of only a few paragraphs. The fact that people are forgotten during the course of the game is due to the fact that the language model never takes the whole story as input, but only the last *n* sentences.

It quickly became apparent that the number of characters involved had to be controlled in order to keep the course of the story within a certain boundary. For this purpose, a simple replacement of new names in the text with already known names was done, excluding those of the main character and his two companions. Care was taken to replace them with the same gender, so that personal pronouns continue to match the gender of the person. The

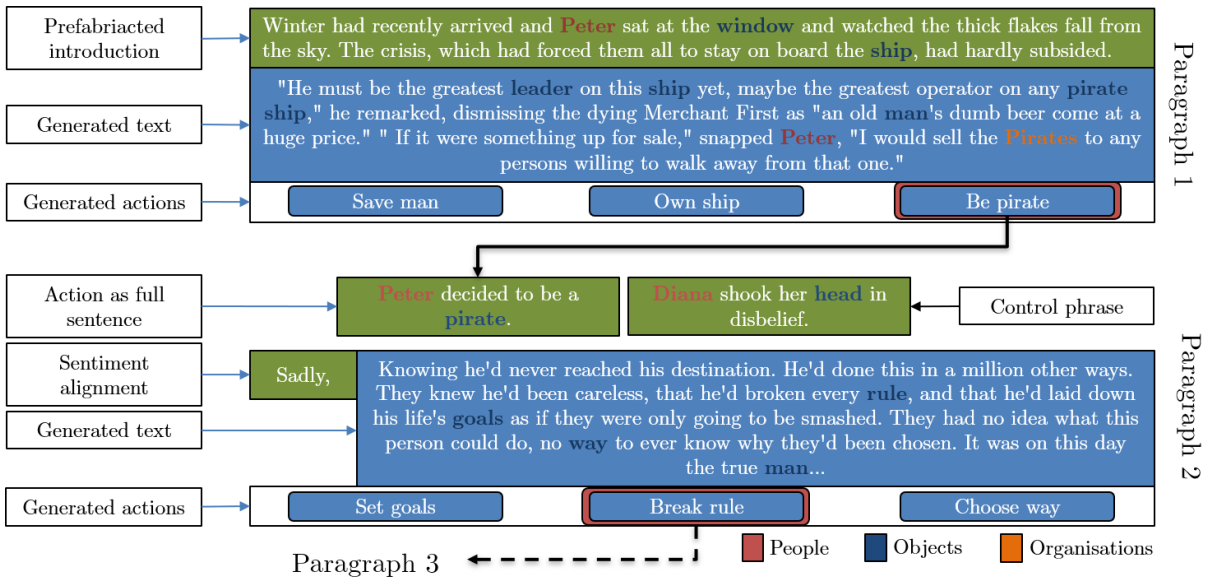

Figure 4: First two paragraphs of a story with all their components. Paragraphs have been shortened for reasons of space. The green components are based on text templates, the blue boxes are generated using language models. Named entities are highlighted according to the legend in the bottom right corner.

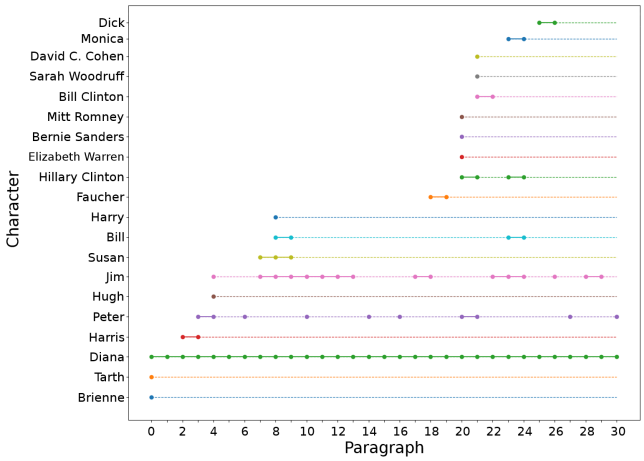

Figure 5: Nearly each new paragraph introduces a new character except the main character and the two party members.

gender detection is done by the *gender-guesser* library which relies on the dictionary of first names by of Jörg Michael [13]. It has also been observed that when a story begins with an excerpt from well-known literature, characters from the book appear. On the one hand, this results in an interesting story, on the other hand, it does not prolong the time during which a character appears in the story.

4.3 Coherence

One of the biggest challenges in generating text adventures automatically is to make the context of the story clearly visible and maintain it across all sections. Although the results of the generated texts using transformer-based language models are very credible in themselves, there is a lack of a red thread when generating longer texts. Through the measures mentioned in Section 3, such as the regular naming of companions or items that the player carries in his inventory, a positive effect on coherence was observed. The following list shows some examples for our self-written control sentences:

- [person] rolled [hisher] eyes.
- [person] cleared [hisher] throat.
- [person] took a deep breath.

We have measured the coherence by applying the four stage topic coherence pipeline presented by Roeder, Both and Hinneburg. They split a word set into pairs of words. Based on a reference corpus the word probabilities are calculated. Then the entire agreement of all pairs is calculated using Normalized Pointwise Mutual Information (NPMI). All these scores are aggregated (arithmetic mean) to a final *uMass* coherence value with a range of $-14 < x < 14$ [20]. We use a word set of ten topics extracted from the respective texts on the basis of Latent Dirichlet allocation (LDA) [2]. In order to determine a reasonable number of topics, we have calculated the coherence of a prepared text for different numbers of topics. The highest coherence and the most meaningful topics was found to be at a number of ten topics. Figure 6 shows a comparison of a story with (left) and without (right) naming companions and inventory items. It shows that the coherence can be slightly improved by interspersing existing information on persons and inventory. This can be explained by the fact that the LDA extracts persons and objects as topics and recognizes them later in subsequent paragraphs.

On a perceptual level, it was also observed that coherence is not always the most important factor when it comes to the entertainment value of the game. Often it is also the game mechanics or a

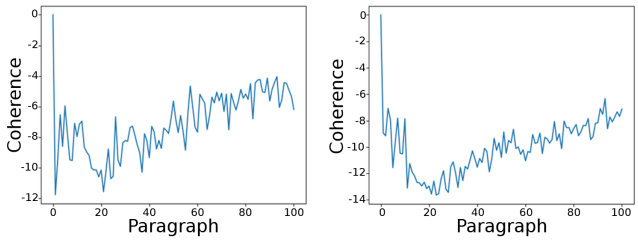

Figure 6: A story with 100 paragraphs with (left) and without (right) elements supporting coherence.

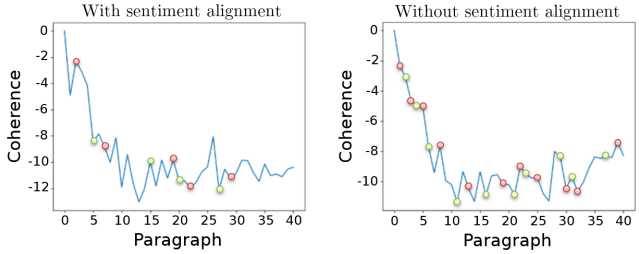

Figure 7: A story with 40 paragraphs with and without sentiment alignment. The red dots signify a negative mood change, the green dots signify a positive mood change.

humorous element that causes a positive reaction from the players. Nevertheless, measures that increase the coherence of the story are important to increase the general acceptance of a procedurally generated story.

4.4 Aligning Sentiments

We have also observed the relationship between positive and negative moods in individual paragraphs by means of a sentiment analysis, and have noticed that mood does not remain the same over a certain number of paragraphs, but changes indiscriminately [1]. Thus, we introduced control words like *luckily* or *unfortunately* as further input for the text generator.

Figure 7 shows that we were able to influence the mood of some paragraphs through the control words. In our example, the mood did not change 31 times (left), but 19 times (right). On the other hand, it also shows that this instrument does not seem to have any influence on the coherence since the two upper graphs are similar.

4.5 Actions

We have evaluated three different approaches constructing actions that the player can select to continue the story. The analytical and the mask-based approach both aim to generate concrete actions. The generative approach produces the most appropriate sentence that continues the story without necessarily containing an activity. Table 2 gives an overview of the observed characteristics of each method. The analytical approach is certainly the one that can and must be controlled most. It allows a more targeted interaction with people and objects. Partially, the actions get implausible because

[1]We measure mood by applying the bert-base-multilingual-uncased-sentiment model by NLP Town [18]

	Analytical	Mask-based	Generative
Controllability	high	medium	low
Diversity	low	medium	high
Interactivity	high	high	low
Complexity	high	low	low
Authenticity	medium	high	high

Table 2: Three approaches to dynamic action generation.

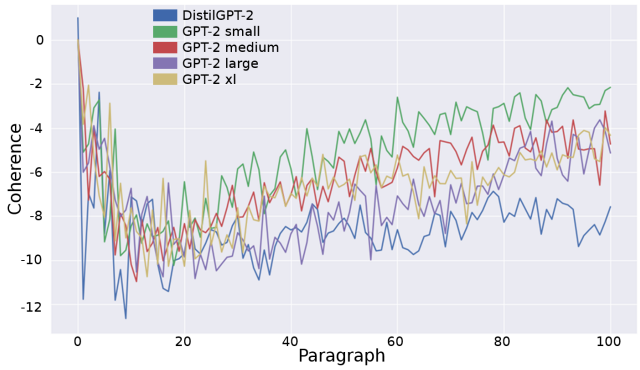

Figure 8: Development of coherence for the different language models in stories with a length of 100 paragraphs.

interactions with non-material objects can occur, e.g., *take love*. Also, actions are often repeated, which is why a list of already generated actions was kept in the implementation, allowing to hide already offered actions. Repeated actions have not been negatively noticed with the mask-based and generative approaches.

In our tests, the mask-based approach offered a good balance between controllability and variety. The generative approach tends to create a narrative out of the text adventure that is more a novel and less a game.

Actions in which objects could be picked up were very rare in our tests. This results in an even lower probability of being able to combine two objects. As mentioned in Section 3.2.2 synonyms for the verb *take* were used to increase the frequency of such an action. The use of the wordnet corpus [16] turned out to be difficult, because the synonyms for *take* can have other meanings than taking objects. For this reason, a curated, static list of matching synonyms was created and used.

4.6 Different Language Models

We have analyzed the various language models for their coherence using the Topic Model Coherence described above. In each case, stories with one possible action and a length of one hundred paragraphs were considered. Figure 8 describes, similar to the previous figures, the context of the topics extracted from the paragraphs via LDA. The consideration is done holistically and not sequentially from two consecutive paragraphs. It shows that GPT-2 small achieves the highest average coherence. The models medium, large and xl are quite close to each other, and the model DistilGPT-2 achieves a visibly lower coherence in the generated example stories. This can be explained by the fact that GPT-2 small is able to

establish a logical connection between the paragraphs (which is apparently difficult for DistilGPT-2), but due to its limited domain it deals more often with the same topics. The variety of contents of medium, large and xl seems to have the effect that the models tend to change subjects frequently.

5 EVALUATION

To determine whether our method can effectively tell more credible, coherent stories, sample Twine stories with a length of ten paragraphs were generated for play testing. Players were told to pay explicit attention to the coherence and credibility of actions after each paragraph. Half of the games contained neither a unification of characters nor control sentences. The other half did. 100% of all players found the games with activated coherence control were more credible and realistic. 40% noted, however, that the character behavior is implausible from time to time.

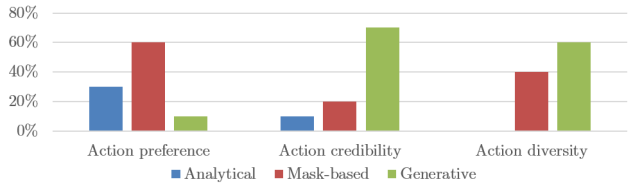

Figure 9: Evaluation of action generation methods

80% of the test players pointed out that especially the interaction with objects and inventory created a feeling of immersion. It also emphasized the character of a game as opposed to a simple narrative. The evaluation of the action generation methods show that the formulation of the mask-based approach was preferred (see Figure 9). Contrary, the credibility and diversity of the generative actions were perceived most distinctly. However, the players stated in the freeform feedback that the game drifted more into the literary realm by using actions entirely generated by a language model.

6 CONCLUSION

We have shown a practical approach to creating procedural, interactive stories for games based on five different language models. Three approaches to a creative and credible action generation were evaluated in terms of their diversity and controllability. It could be shown that an analysis of the previous texts via a classical NER, *WordNet Synsets* and the subsequent application of a mask-based approach to find a suiting verb leads to the most credible and yet controllable actions that can be offered to the player to continue the story of the game. By limiting the number of different characters and using control sets, we were able to improve the coherence of the stories. We have measured this coherence based on LDA and Topic Model Coherence. We believe that the use of language models trained via neural networks will give a boost to the procedural generation of stories for text-based games.

REFERENCES

[1] Ernest Adams. 2014. *Fundamentals of game design.* Pearson Education.
[2] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent Dirichlet Allocation. *J. Mach. Learn. Res.* 3, null (March 2003), 993–1022.
[3] Shikha Bordia and Samuel R Bowman. 2019. Identifying and reducing gender bias in word-level language models. *arXiv preprint arXiv:1904.03035* (2019).
[4] Yun-Gyung Cheong, Mark O Riedl, Byung-Chull Bae, and Mark J Nelson. 2016. Planning with applications to quests and story. In *Procedural Content Generation in Games.* Springer, 123–141.
[5] Simon Colton, Geraint A Wiggins, et al. 2012. Computational creativity: The final frontier?. In *Ecai*, Vol. 12. Montpelier, 21–26.
[6] Kate Compton, Ben Kybartas, and Michael Mateas. 2015. Tracery: An Author-Focused Generative Text Tool. In *Interactive Storytelling*, Henrik Schoenau-Fog, Luis Emilio Bruni, Sandy Louchart, and Sarune Baceviciute (Eds.). Springer International Publishing, Cham, 154–161.
[7] Margaret Cychosz, Andrew S Gordon, Obiageli Odimegwu, Olivia Connolly, Jenna Bellassai, and Melissa Roemmele. 2017. Effective scenario designs for free-text interactive fiction. In *International Conference on Interactive Digital Storytelling.* Springer, 12–23.
[8] Angela Fan, Mike Lewis, and Yann Dauphin. 2019. Strategies for structuring story generation. *arXiv preprint arXiv:1902.01109* (2019).
[9] Angela Fan, Jack Urbanek, Pratik Ringshia, Emily Dinan, Emma Qian, Siddharth Karamcheti, Shrimai Prabhumoye, Douwe Kiela, Tim Rocktäschel, Arthur Szlam, et al. 2020. Generating Interactive Worlds with Text.. In *AAAI.* 1693–1700.
[10] Christiane Fellbaum. 1998. A semantic network of english: the mother of all WordNets. In *EuroWordNet: A multilingual database with lexical semantic networks.* Springer, 137–148.
[11] Darius Kazemi. 2019. Keeping Procedural Generation Simple. *Procedural Storytelling in Game Design* (2019), 17–22.
[12] David D Lewis, Yiming Yang, Tony G Rose, and Fan Li. 2004. Rcv1: A new benchmark collection for text categorization research. *Journal of machine learning research* 5, Apr (2004), 361–397.
[13] Jörg Martin. 2017. 40.000 names. https://www.heise.de/ct/ftp/07/17/182/
[14] 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.*
[15] Michael Mateas and Andrew Stern. 2003. Façade: An experiment in building a fully-realized interactive drama. In *Game developers conference*, Vol. 2. 4–8.
[16] George A Miller. 1998. *WordNet: An electronic lexical database.* MIT press.
[17] Nasrin Mostafazadeh, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James Allen. 2016. A corpus and evaluation framework for deeper understanding of commonsense stories. *arXiv preprint arXiv:1604.01696* (2016).
[18] Yves Peirsman and Johan Leys. 2020. NLP Town. https://www.nlp.town
[19] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog* 1, 8 (2019), 9.
[20] Michael Röder, Andreas Both, and Alexander Hinneburg. 2015. Exploring the Space of Topic Coherence Measures. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining* (Shanghai, China) *(WSDM '15).* Association for Computing Machinery, New York, NY, USA, 399–408. https://doi.org/10.1145/2684822.2685324
[21] Melissa Roemmele. 2018. *Neural Networks for Narrative Continuation.* Ph.D. Dissertation. University of Southern California.
[22] Noor Shaker, Julian Togelius, and Mark J Nelson. 2016. *Procedural Content Generation in Games.* Springer.
[23] Tanya Short and Tarn Adams. 2017. *Procedural generation in game design.* CRC Press.
[24] Pradyumna Tambwekar, Murtaza Dhuliawala, Lara J Martin, Animesh Mehta, Brent Harrison, and Mark O Riedl. 2018. Controllable Neural Story Plot Generation via Reinforcement Learning. *arXiv preprint arXiv:1809.10736* (2018).
[25] Jack Urbanek, Angela Fan, Siddharth Karamcheti, Saachi Jain, Samuel Humeau, Emily Dinan, Tim Rocktäschel, Douwe Kiela, Arthur Szlam, and Jason Weston. 2019. Learning to speak and act in a fantasy text adventure game. *arXiv preprint arXiv:1903.03094* (2019).
[26] Nick Walton. 2019. AI Dungeon. https://aidungeon.io/
[27] Nathan Whitmore. 2019. GPT Adventure. https://quicktotheratcave.tumblr.com/post/187432425523/shall-we-play-a-game-a-gpt-2-text-adventure
[28] Thomas Wolf. 2017. State-of-the-art neural coreference resolution for chatbots. https://medium.com/huggingface/state-of-the-art-neural-coreference-resolution-for-chatbots-3302365dcf30
[29] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. HuggingFace's Transformers: State-of-the-art Natural Language Processing. *ArXiv* abs/1910.03771 (2019).
[30] Lili Yao, Nanyun Peng, Ralph Weischedel, Kevin Knight, Dongyan Zhao, and Rui Yan. 2019. Plan-and-write: Towards better automatic storytelling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 7378–7385.