Generating Interactive Worlds with Text

Angela Fan,[*1,2] **Jack Urbanek,**[*1] **Pratik Ringshia,**[1] **Emily Dinan,**[1]
Emma Qian,[1] **Siddharth Karamcheti,**[1] **Shrimai Prabhumoye,**[1] **Douwe Kiela,**[1]
Tim Rocktäschel,[1,3] **Arthur Szlam,**[1] **Jason Weston**[1]
[1]Facebook AI Research
[2]LORIA, Nancy
[3]University College London
light-dms@fb.com

Abstract

Procedurally generating cohesive and interesting game environments is challenging and time-consuming. In order for the relationships between the game elements to be natural, common-sense has to be encoded into arrangement of the elements. In this work, we investigate a machine learning approach for world creation using content from the multi-player text adventure game environment LIGHT (Urbanek et al. 2019). We introduce neural network based models to compositionally arrange locations, characters, and objects into a coherent whole. In addition to creating worlds based on existing elements, our models can generate new game content. Humans can also leverage our models to interactively aid in worldbuilding. We show that the game environments created with our approach are cohesive, diverse, and preferred by human evaluators compared to other machine learning based world construction algorithms.

1 Introduction

A large component of fantasy and science fiction literature is *worldbuilding*: putting together an elaborate context, with interesting (but believable) details, that can serve as a backdrop to a story (or for many stories). Successful worldbuilding requires common-sense knowledge about the real world and an understanding of the expectations of the audience.

In this work, we present a machine learning (ML) approach to creating a cohesive and interesting world built from elements of the text-based fantasy game environment LIGHT (Urbanek et al. 2019). These crowd-sourced elements, including descriptions of locations, characters, and objects, provide a rich source of supervision for learning common-sense relationships. Previous work in LIGHT focused on static, single-location settings using the crowd-sourced data. Instead, we focus on creating full environments for players to explore. We show how ML algorithms can learn to assemble these different elements, arranging locations and populating them with characters and objects. We use models to learn how to answer questions such as: *Where is the ornate trunk likely to be? What is likely to be inside it? Where is the knight likely to be?* These considerations are necessary for building a cohesive game environment.

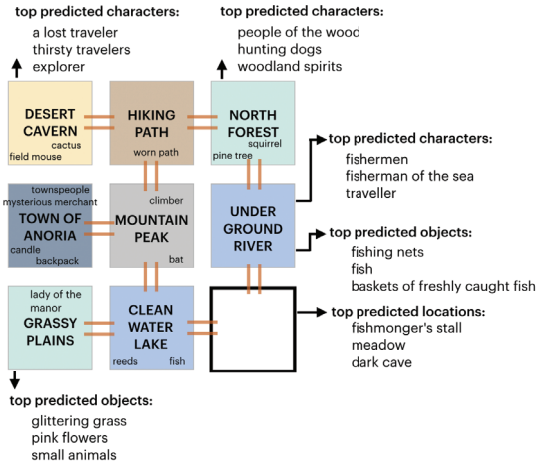

Figure 1: Sample Constructed Game World. Models arrange locations, then populate them with characters and objects. Top predictions are shown. Table 1 shows the descriptions associated with the location *Town of Anoria*, placed in middle left of this generated world.

We demonstrate that our proposed models can construct rich game environments that are diverse and preferred by human evaluators. We also develop models to generate descriptions of new locations, characters, and objects. Finally, we demonstrate that these machine learning tools can aid humans interactively in designing new game environments.

2 Constructing Game Environments

In this section, we detail methods for learning to build game elements from compositions of sub-elements; and worlds from these elements.

2.1 Background on LIGHT

LIGHT is a multi-player text-based fantasy-themed virtual world. It consists of a set of crowd-sourced *game locations, characters, and objects*, and a game engine that controls the

[*] Joint first authors.

Name:	Town of Anoria
Description:	Town of Anoria has lots of cobble stone streets and wood houses of one story. [...] The town of Anoria is inland and takes a long time to reach the sea, [...]
Neighbors:	Mountain's Peak
Characters:	townspeople, mysterious merchant
Objects:	candle, backpack

(a) Example location: Our model placed the Town of Anoria with an exit to the Mountain Peak, and placed characters and objects inside this location.

Character:	Mysterious Merchant
Persona:	I am the mysterious merchant of the village. I sell rarities from around the world that can not be purchased anywhere else. [...]
Description:	The merchant in town came and went without a crack of the grass beneath his feet. No one knew when he was gone, nor when he returned home [...]
Carrying:	pouch, cane
Wearing:	hat, coat
Wielding:	dagger

(b) Example character: Our model placed the Mysterious Merchant in the Town of Anoria, along with other townspeople.

Object	Description	Affordances
pouch	The pouch is made of fine silk cloth, colored bright red. It has a leather string keeping it sealed.	container gettable
cane	The cane is made of a very uncommon, ornate wood.	gettable
dagger	The dagger is curved with a golden hilt.	gettable weapon

(c) Example objects: Pouch, Cane, and Dagger, all carried by the Mysterious Merchant.

Object	Inside the Object
pouch	coins, eyeglasses
backpack	wallet, bedroll, tools

(d) Example objects within container objects: Our model placed additional objects inside the Pouch and the Backpack.

Table 1: Game Elements include locations, characters, objects, and objects within containers. Elements have descriptions and annotations such as what a location contains.

interactions between these. Characters can speak to each other via text, send emotes like *grin* or *ponder*, and take actions to move to different locations and interact with objects. Some example actions include *go north*, *get shovel*, or *unlock door*. The game engine represents the game state as a graph, and the actions by characters amount to operations on the graph. The locations, characters, and objects were crowd-sourced using Amazon Mechanical Turk. Crowd-workers were asked to provide names and descriptions for each of these aspects through natural language, for a total of 663 locations, 1755 characters, and 3462 objects. See Table 1 for examples and Figure 1 to see how our work combines the elements into a playable game environment.

2.2 Building a Game World

Urbanek et al. focused on modeling character dialogue and action in pre-built locations. ML models were trained to play the game by mimicking the actions and dialogues of human players in fixed settings built by crowd-workers. In contrast, in this work, we study models for assembling the game itself rather than agents that play it. Since these elements were separately crowd-sourced, we can compose them to create a large number of different game environments.

We describe our approach from the top down. First, in Section 2.3 we discuss connecting pre-built locations together to form a world. In 2.4, we give our methods for filling a location with characters and objects. We describe the additional data collected to model objects *contained* within other objects. Next, in 2.5, we discuss how to generate new game elements. In Section 2.6 we describe how these methods can be modified and utilized for interactive world-building. Finally, in Section 2.7, we describe how to bring these models together to create a new game world.

2.3 Building Maps by Arranging Locations

We describe our method to train machine learning models to arrange locations in LIGHT.

Locations in LIGHT Each *location* represents a place with a *name* and *description*. The description provides background information about the location and what a player might see as they enter it. Crowd-workers provided examples of neighboring locations, as well as what characters and objects would be present within the location.

Using Machine Learning to Place Locations Game locations must be spatially arranged so as to create a logical and cohesive environment for players to explore. For example, the *Wizard's Reagent Room* being located near the *Wizard's Tower* would make a more intuitive game experience compared to locations being randomly placed.

To train models for this task, we use the example neighbors for each location provided by crowd-workers, obtaining triplets of (location name, location description, location neighbors). We partitioned this into a training, validation, and test set such that the locations are distinct in each set (see Table 2). As each location can have multiple neighbors, the individual datapoints available for the prediction task is larger than the number of total locations collected.

We consider a variety of different *ranking models* for this task, in two settings. In the first, models have access to the location name only, and in the second, they additionally have access to the location description information. These models compare the human annotation of neighboring locations with a variety of negative candidates. These negative candidates can be thought of as distractor locations from the

dataset that the model must distinguish from the human annotated location, similarly to how negative training data is sampled in the knowledge base population literature (Bordes et al. 2013). Models are trained to maximize the score of the human response and minimize the scores of the negative candidates. When constructing a new world at test time, the placed location is the highest scoring candidate from the model prediction. We use two machine learning approaches:

- *Starspace:* The Starspace (Wu et al. 2018) model learns a bag-of-words embedding for the location information (e.g. name and description). The model encodes the location information as well as the negative candidates, and trains to maximize the inner product of the true human annotation. We initialized the Starspace model using *fasttext*, a method for learning vector representations of individual words. This initialization allows the model to begin training with a better understanding of the text.
- *BERT-based Models:* Recent work (Devlin et al. 2019) in natural language processing has shown strong performance of the BERT model, which learns to encode text in a left-to-right and right-to-left fashion by training on large quantities of text data available online. We use the BERT-based models proposed in (Urbanek et al. 2019; Humeau et al. 2019) to encode the location information and the negative candidates. We explore two variants:

  (1) *Bi-Encoder*, which encodes the candidates and input context separately. This model scores the candidates by calculating the dot product between these embeddings.

  (2) *Cross-Encoder*, which concatenates the context with each candidate before encoding, allowing this model to build a context-dependent representation of each candidate. This model scores candidates by projecting the vector representation of text to a scalar.

As we have a limited quantity of data for the task, we found that using input dropout to prevent overfitting was crucial for good performance for both of these models.

We compare these models that learn from the training data with three baselines:

- *Random:* We report a random baseline that selects a random candidate from the provided negative candidates.
- *Data Proportional:* Instead of selecting candidates fully at random from the provided negative candidates, we select proportional to the number of times that candidate appears in the training set. This leverages the data annotation information and reflects that some candidates are more likely to be used than others.
- *Information Retrieval:* This model selects the candidate with the largest word overlap using TF-IDF weighting.

To create a map for a new game, models are used to predict the neighboring locations of each existing location. For each new location added, the model will fill in the surroundings. A location can connect to up to four neighboring locations, though not all connections need to be filled. To make the game environment more interesting and diverse, locations cannot appear multiple times in one map (e.g. *Berka's Forest Inn* is only located in one place).

Adding Filler Locations A challenge with using crowdsourced data for all of the locations is that crowd-workers often write exciting and complex locations. However, when players explore the game environment, this tendency leads to each location being complex and overwhelming. To remedy this, we create a set of 25 filler locations such as *abandoned shack*, *empty closet*, and *storage room* that provide additional content between the exciting locations that crowd-workers described. Filler locations can appear multiple times (e.g. there can be multiple *empty closets*).

2.4 Adding Characters and Objects to Locations

We describe how to apply our methods to add characters and objects to predicted locations.

Characters in LIGHT Each character is described by a *name*, *persona*, and a *description*. The persona provides information about the character, such as their background and motivation, while the description describes the character's appearance. LIGHT also has annotations of objects characters would carry, such as a *Wizard* holding a *staff*.

Objects in LIGHT Each object represents an item that characters can interact with, such as *get shovel*. Objects have a *name*, a *description*, and a set of *affordances*. The description lists what the object looks like and what it might be able to do. The affordances represent object properties, such as gettable and drinkable. These are used by the game engine to determine the set of possible interactions of the object. For example, objects with the drinkable affordance can be interacted with using the action *drink*. Objects can be inside other objects, to represent for example *coins inside a wallet*. We crowd-sourced additional annotations of object size and examples of other objects that could be inside.

Using Machine Learning to Place Characters and Objects Using the characters and objects associated to locations from LIGHT as ground-truth, we create training, validation, and testing data (see Table 2) to fit models to place characters and objects in locations, as well as object within objects. To collect objects within objects data, crowd-workers were given an object with the container affordance and asked to name multiple objects that could be inside.

We place characters and objects using the models described in Section 2.3. Here, instead of predicting neighboring locations, models are given locations and trained to predict characters and objects, or given objects and trained to predict which objects could be inside. For example, the *character prediction* task would receive as input the location *Wizard's Reagent Room* and predict *Wizard*. As the amount of data for each task is low, we employ multi-task learning and train all of the tasks (locations, characters, objects, and containers) together to increase the quantity of training data.

2.5 Generating New Game Elements

Adding new elements to the existing LIGHT game is complex: descriptions, object affordances, character personas,

Split	Train	Valid	Test
Locations	914	109	110
Characters	529	305	305
Objects	359	318	256
Object Containers	359	318	256

Table 2: Dataset Statistics for World Generation: arranging locations next to each other, placing characters and objects within locations, and placing objects within objects.

and other details would need to be written. Instead, we propose using generative machine learning models to create additional content based on the name of the new item (either a location, a character, or an object). We use the same training, validation, and testing splits used in the world construction task (see Table 2). These generated items can be added to the game environment, so newly generated game worlds can incorporate them along with existing crowd-sourced elements.

We use the Transformer (Vaswani et al. 2017) neural network architecture to create a Sequence-to-Sequence model to make the following predictions:

- Given location name, predict background and description
- Given character name, predict persona and description
- Given object name, predict description and affordances

We compare the Transformer in two settings: with and without *pretraining*. As the dataset for generating new game elements is small, the generative model can be trained on a larger corpus and finetuned on this task. We use a large dataset of 2 billion Reddit comments for pretraining. Reddit comments are chosen because they are close to natural human conversation and exhibit elements of creativity and story-telling that may help generate interesting descriptions.

To be able to handle new vocabulary and ease learning, we use byte-pair encoding (Sennrich, Haddow, and Birch 2016) to model subwords. Similar to Section 2.4, we multi-task prediction location, character, and object description, location background, and character persona with one model. We use top-k sampling (Fan, Lewis, and Dauphin 2018) to reduce repetition during generation. Object affordances are predicted with a separate model, as multi-label classification between seven possibilities is distinct from the other tasks.

2.6 Aiding Human Game Design

Machine learning models can be applied to automatically generate game environments for players, but they can also be used to aid humans in game design. Many existing game engines assist in fast and intuitive creation of different worlds already, for example providing level design tips or improving pathfinding (Graham, McCabe, and Sheridan 2003). Our methods can be used to automatically suggest neighboring locations or which characters and objects to place in the existing locations, speeding up world design.

2.7 Proposed Algorithm for World Generation

How do we use our proposed models collectively to make a new game world? First, an empty map grid is initialized

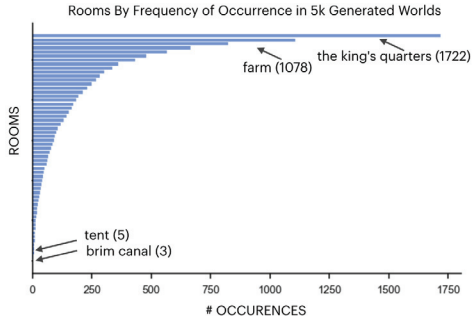

Figure 2: Frequency of Location Placement in 5000 generated game environments using our models.

to represent the number of possible locations. A percentage of grid positions are marked inaccessible to make exploration more interesting. The central location is populated randomly. We use the best performing model to iteratively fill in neighboring locations until the entire grid is populated. Then, for each placed location, the model is used to predict which characters and objects should populate that location. Finally, the model is used to predict if objects should be placed inside existing objects. Figure 1 displays an example generated world, with model predictions shown for missing elements. See Appendix for further details.

In an interactive setting where players are able to design their own worlds, we use models to provide suggestions for which elements to place. If players enter names of game elements not present in the dataset, our generative models are used to write descriptions, personas, and affordances.

3 Related Work

Procedural Content Generation in Games Using algorithms to aid game generation is a growing field as the popularity of gaming rises. Recent work has made progress on level design in various game settings (Guzdial and Riedl 2018; Khalifa et al. 2016; Summerville et al. 2016; Van der Linden, Lopes, and Bidarra 2013; Vara 2014), including rhythm games (Lin, Riedl, and Xiao 2019), physics games (Stephenson and Renz 2016), dungeon exploration (Shaker et al. 2016), and social games (Risi et al. 2012).

Much prior work has focused on the task of level generation, but there are other facets of games that could be generated. For example, weapons, various items, and characters are present in game levels (Liapis, Yannakakis, and Togelius 2014; Liapis et al. 2018). We focus on how the various facets could fit together within a text-based game, and how we can use them to generate an entire game environment.

Text-Based Games Many settings for content generation in text-based games have been explored. For example, Barros, Liapis, and Togelius (2016) use text from Wikipedia to link various entities for the generation of murder mystery games. Ammanabrolu and Riedl (2019) represent a text-based adventure game as a graph and learn how to adventure within this world. Work has been done to generate Sporcle-

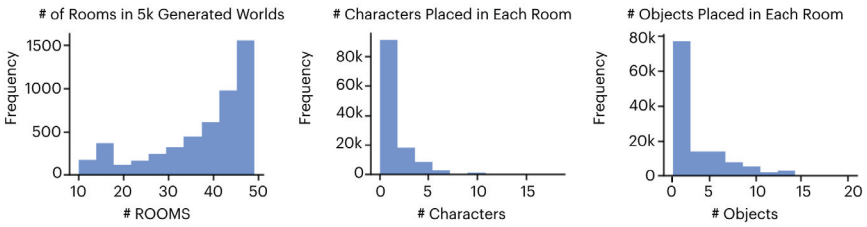

Figure 3: Distribution of Locations, Characters, and Objects in 5,000 generated maps. Our method generates fairly large maps (the maximum size is set to 50) and places 1-3 characters and objects in each location.

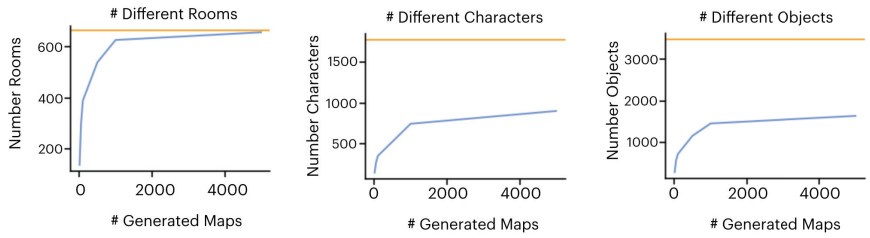

Figure 4: Number of Different Locations, Characters, and Objects as a Function of Generated Maps. As additional maps are generated, a greater diversity of game elements appears. The orange line denotes the total number of elements in the dataset.

like textual quizzes.[1] However, designing generative algorithms to create a full environment for a multi-player text game has not been deeply explored.

Generation using Machine Learning Generative modeling is an important topic in machine learning, outside of games or other creative endeavors. Recent works have demonstrated impressive models of images (Karras et al. 2018) and text (Radford et al. 2019). Recently, statistical ML has also been proposed for creative endeavors. For example Gatys, Ecker, and Bethge show how users can manipulate the style of images using convolutional neural networks and (Zhu et al. 2017; Sbai et al. 2018) describe ML-aided fashion design. There has been work in ML for music generation, see e.g. Magenta[2] or Briot, Hadjeres, and Pachet (2017) for a survey. Most related to our world construction are methods for generating stories, poetry, and scripts (Fan, Lewis, and Dauphin 2018; Ghazvininejad et al. 2016; Janghorbani et al. 2019; Marti et al. 2018).

Work in content generation with machine learning has incorporated human guidance. For example, several works incorporate human control such as length and style to improve summarization, dialogue, and text simplification (Fan, Grangier, and Auli 2018; See et al. 2019; Martin et al. 2019). Wang et al. (2018) generates portions of images after human editing.

4 Evaluation and Results

We discuss several evaluations of both elements and worlds, compare methods, and discuss their successes and failures.

[1]https://github.com/vjuylov/txt2lvl

[2]https://magenta.tensorflow.org/

4.1 Diversity of Generated Worlds

Our proposed method can be used to automatically create a variety of diverse game worlds. We generate 5,000 worlds with a maximum size of 50 arranged locations and analyze these generations to understand the properties of created game environments.

Locations The generated maps are very diverse. Figure 4 shows the number of map generations required to generate the full number of locations in the dataset. With 500 generations, a large majority of different locations have been used. Over 95% of locations in the dataset are used after 5000 generations. The most commonly placed location is *the king's quarters*, in 34% of the generated worlds (see Figure 2). Some locations are used very sparingly, such as the *brim canal* (0.06% of the worlds). Allowing our modeling approach to decide the map size, 80% of the generated worlds have more than 30 locations (see Figure 3) and about 40% of the worlds have the maximum number locations. Example generated maps are shown in the Appendix.

Characters Around 65% of characters in the dataset are generated after 5000 maps (Figure 4). The lower coverage is most likely because there are very specific characters created by crowd-workers that are not scored highly by models, and thus not often placed. To provide a concrete example, a specific qualified character might be in the dataset, such as *an old, wizened priestess*, but if that character is only mentioned once in the training set, a model might score a more generic character higher, such as *priestess*. The maximum number of characters placed in one room is around 15, but

most locations have 0-3 characters present (see Figure 3).[3]

Objects Similar to characters, around 60% of objects in the dataset are generated after 5000 maps, shown in Figure 4. Some locations contain a large number of objects, such as the *Treasure Chamber*, but most locations contain about 1-3 objects that players can interact with (Figure 3).

4.2 Quality of Generated Worlds

Automatic Evaluation We first use *automatic evaluation* to compare the quality of different machine learning approaches to the location, character, object, and container prediction tasks. We measure *Hits at 1*, or the percentage of time the correct candidate is ranked first amongst the negative candidates. If the model always predicted what the crowd-workers annotated, then this metric would have the value 100. Containers are evaluated in the *Name only* variant — as crowd-workers were able to write any object, not all of their written choices have descriptions.

Results are shown in Table 3. Leveraging the data distribution to weight random sampling provides a strong baseline for characters and objects, as a few are quite common. Providing the description text is helpful for improving prediction quality compared to having access only to the name feature. Amongst the various approaches, Starspace models show strong performance, particularly on the location prediction task. The Bi and Cross Encoder models are very large neural networks and may be overfitting on the much smaller LIGHT world creation training data. Further, they are pretrained on non-domain specific data, which may negatively impact performance.

Human Assessments We conduct *human evaluation* to compare the various approaches to world generation. We compare the performance of two models pairwise by starting in the same location and using the models to iteratively predict subsequent locations. Locations are then populated with characters and objects. After each location, human evaluators are asked which model was able to place more logical and interesting characters and objects. After five steps through predicted locations, human evaluators are asked which model path they prefer as more natural, cohesive, and interesting. We compare four different approaches:

- *Random:* Locations, characters, and objects were randomly selected from the set of all possible datapoints.
- *Starspace:* The model described in Section 2 was used to predict which locations should be linked in the path, and the characters and objects present in each location.
- *Data Created Paths:* This method uses the existing dataset of annotated locations and their neighbors to construct a path. The characters and objects present in each location are from the original crowd-sourcing tasks. In contrast to Starspace, the number of possible paths that could be created with existing data is limited. For example, if a room has only one annotated neighbor, it would always be arranged in the same manner.
- *Human Annotated Paths:* Human evaluators constructed paths by manually linking locations. Here a single evaluator created an entire path — in contrast, in *Data Created Paths*, annotators during initial data collection only provided a one-step neighbor, rather than one person creating the entire path. The characters and objects are from the original crowd-sourcing. While human-created paths could be high quality, such a method does not scale to large worlds as it is costly and time consuming.

[3]ML approaches are known to reflect data biases (Zhao et al. 2019; Brunet et al. 2019). We found that there are a greater number of male characters in LIGHT, and this is reflected in the generated environments. We plan to investigate this in a follow-up work.

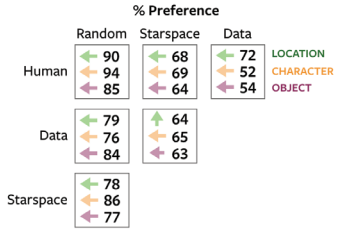

Figure 5: Human Evaluation of World Construction. The number indicates percentage preference, with the arrow pointing to the winner. The first row in each box is location preference, second character, and third object.

As shown in Figure 5, human evaluators prefer Human Annotated Paths the most, but Starspace prediction models perform strongly as well. Starspace is strongly preferred over Random and the predicted location paths are preferred over Data Created Paths over 60% of the time.

4.3 Generation of New Game Elements

To evaluate the quality of automatically generating new game elements using our proposed models, we compare F1, a metric of word overlap. For this metric, the text is lowercased and the overlap between tokens is computed. Pretraining increases the performance on all generation tasks, as shown in Table 4. For character descriptions and personas, the effect of pretraining is minimal. We hypothesize this is due to the slightly more templated nature of written personas, as many begin with *I am a*. Example generations are shown in Table 5. Our generative models are able to write interesting, new, and generally coherent descriptions for a variety of different game elements (see Appendix for additional examples). We analyze the n-gram overlap of our generated game elements with the training set to understand how much of the written text is novel. We find that 34% of generated 3-grams are present in the training set (largely common phrases), but only 2.5% of generated 5-grams are present in the training set. As we are generating text with top-k sampling, the models do not tend to copy long sequences.

4.4 ML-aided interactive world creation

To quantify if models can aid players in designing their own worlds, evaluators designed a nine-location game environ-

Feature	Model	Locations	Characters	Objects	Containers
	Random	8.2	5.9	5.9	5.7
	Data Proportional	0.0	9.8	20.1	5.9
Name Only	Information Retrieval	18.2	7.5	8.2	9.6
	Fasttext	9.1	12.8	15.6	**27.4**
	Starspace	44.5	17.7	13.3	20.1
Name and Description	Information Retrieval	30.0	19.0	21.9	—
	Fasttext	28.2	17.0	16.8	—
	Starspace	**45.5**	35.7	**47.3**	—
	BERT Bi-Encoder	30.2	30.2	34.0	—
	BERT Cross-Encoder	28.2	**36.1**	35.5	—

Table 3: Comparison of Various Approaches to Worldbuilding. We report Hits at 1 on the test set for arranging locations and populating with objects, characters, and placing objects within container objects. Starspace models perform well on all tasks.

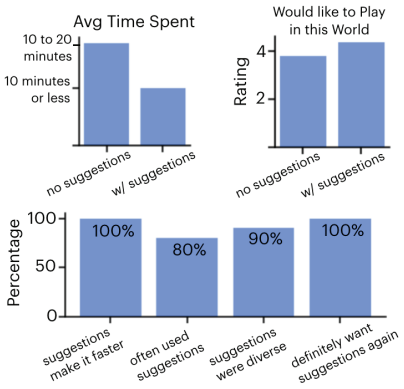

Figure 6: Machine Learning Models Aiding Human Game Design. Having access to model suggestions for which elements to place next allows evaluators to create fun and interesting game environments more quickly.

Model	Location	Character	Object	Affordance
- Pretrain	23.3	28.4	32.5	57.0
+ Pretrain	26.3	29.3	38.8	62.6

Table 4: Comparison of Game Element Generation. We report F1 on the test set.

Given Location:	The forest
Generated Background:	This forest has been a part of the kingdom for centuries. It is a great resource to have when needed. Most people consider it a necessity to have a peaceful and peaceful environment.
Generated Description:	The forest is filled with wild animals that live in the forest, but are unable to climb up trees. It is clear the trees are not in use.
Given Character:	A Princess
Generated Persona:	I love to help people and I love my family. I spend my days fighting, protecting my people and my children. I love to help the royal family!
Generated Description:	She is a fierce looking princess that has been trained to protect any people.
Given Object:	wooden sword
Generated Description:	This wooden sword is worn down and damaged, to say the least, it is a sword that would be useful for a knight's duty.
Generated Affordances:	gettable weapon

Table 5: Generations of New Game Elements, given the desired name. Examples are chosen from the test set.

ment. Evaluators were explicitly told the goal was to make a text-based game interesting and fun. To add game elements, they have access to a search bar with autocomplete, so they can type what they wish to place and select from a list (see Appendix for an image of the user interface). Half of the evaluators have access to model predictions, which are surfaced as suggestions at the top of the search dropdown. However, they can choose to ignore the suggestions.

Evaluators created 10 game environments with access to model suggestions and 10 without. While they reported similar satisfaction with the diversity and quality of their generated worlds in both settings, the amount of time spent was different. Evaluators spent 10 minutes or less to create maps with suggestions and 10-20 minutes without suggestions (Figure 6, top left). Those with suggestions said they would want to play an actual game in their created world more (Figure 6, top right). Finally, evaluators had a positive reaction to model suggestions (Figure 6, bottom): 100% of evaluators agreed that suggestions made it faster to create a world and definitely would want to have them again, 80% said they often chose from suggestions, and 90% said the suggestions were diverse. Freeform feedback was positive, with comments such as *suggestions foster creativity* and *especially for characters, the suggestions showed what I wanted*. Additional results are shown in the Appendix.

5 Conclusion

We proposed a method to procedurally generate game environments by using machine learning algorithms to arrange locations, place characters and objects within those locations and objects within containers, and write descriptions for new game elements. We explored different neural network based models for these tasks, and show with various automatic metrics and human studies that the maps generated by our approach are cohesive, interesting and diverse. Finally, we show that our machine learning approach can be used to aid humans in creating game worlds as well. Together, these steps show a path to creating cohesive game worlds from crowd-sourced content, both with model-assisted hu-

man creation tooling and fully automated generation.

References

Ammanabrolu, P., and Riedl, M. 2019. Playing text-adventure games with graph-based deep reinforcement learning. In *NAACL-HLT*, 3557–3565.

Barros, G. A.; Liapis, A.; and Togelius, J. 2016. Murder mystery generation from open data. In *Proceedings of the International Conference on Computational Creativity*.

Bordes, A.; Usunier, N.; Garcia-Duran, A.; Weston, J.; and Yakhnenko, O. 2013. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, 2787–2795.

Briot, J.; Hadjeres, G.; and Pachet, F. 2017. Deep learning techniques for music generation - A survey. *CoRR*.

Brunet, M.; Alkalay-Houlihan, C.; Anderson, A.; and Zemel, R. S. 2019. Understanding the origins of bias in word embeddings. In *ICML*.

Devlin, J.; Chang, M.; Lee, K.; and Toutanova, K. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, 4171–4186.

Fan, A.; Grangier, D.; and Auli, M. 2018. Controllable abstractive summarization. In *ACL Workshop on Neural Machine Translation and Generation*.

Fan, A.; Lewis, M.; and Dauphin, Y. 2018. Hierarchical neural story generation. In *ACL*.

Gatys, L. A.; Ecker, A. S.; and Bethge, M. 2016. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2414–2423.

Ghazvininejad, M.; Shi, X.; Choi, Y.; and Knight, K. 2016. Generating topical poetry. In *EMNLP*, 1183–1191.

Graham, R.; McCabe, H.; and Sheridan, S. 2003. Pathfinding in computer games. *The ITB Journal* 4(2):6.

Guzdial, M., and Riedl, M. 2018. Automated game design via conceptual expansion. In *Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Humeau, S.; Shuster, K.; Lachaux, M.-A.; and Weston, J. 2019. Real-time inference in multi-sentence tasks with deep pretrained transformers. *arXiv preprint arXiv:1905.01969*.

Janghorbani, S.; Modi, A.; Buhmann, J.; and Kapadia, M. 2019. Domain authoring assistant for intelligent virtual agent. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, 104–112. International Foundation for Autonomous Agents and Multiagent Systems.

Karras, T.; Aila, T.; Laine, S.; and Lehtinen, J. 2018. Progressive growing of gans for improved quality, stability, and variation. In *ICLR*.

Khalifa, A.; Perez-Liebana, D.; Lucas, S. M.; and Togelius, J. 2016. General video game level generation. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, 253–259. ACM.

Liapis, A.; Yannakakis, G. N.; Nelson, M. J.; Preuss, M.; and Bidarra, R. 2018. Orchestrating game generation. *IEEE Transactions on Games* 11(1):48–68.

Liapis, A.; Yannakakis, G. N.; and Togelius, J. 2014. Computational game creativity. Citeseer.

Lin, Z.; Riedl, M.; and Xiao, K. 2019. Generationmania: Learning to semantically choreograph. In *Proceedings of the 2nd Workshop on Knowledge Extraction from Games*.

Marti, M.; Vieli, J.; Witoń, W.; Sanghrajka, R.; Inversini, D.; Wotruba, D.; Simo, I.; Schriber, S.; Kapadia, M.; and Gross, M. 2018. Cardinal: Computer assisted authoring of movie scripts. In *23rd International Conference on Intelligent User Interfaces*, 509–519. ACM.

Martin, L.; de la Clergerie, E.; Sagot, B.; and Bordes, A. 2019. Controllable sentence simplification.

Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; and Sutskever, I. 2019. Language models are unsupervised multitask learners.

Risi, S.; Lehman, J.; D'Ambrosio, D. B.; Hall, R.; and Stanley, K. O. 2012. Combining search-based procedural content generation and social gaming in the petalz video game. In *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Sbai, O.; Elhoseiny, M.; Bordes, A.; LeCun, Y.; and Couprie, C. 2018. Design: Design inspiration from generative networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 0–0.

See, A.; Roller, S.; Kiela, D.; and Weston, J. 2019. What makes a good conversation? how controllable attributes affect human judgments. In *NAACL-HLT*, 1702–1723.

Sennrich, R.; Haddow, B.; and Birch, A. 2016. Neural machine translation of rare words with subword units. In *ACL*.

Shaker, N.; Liapis, A.; Togelius, J.; Lopes, R.; and Bidarra, R. 2016. Constructive generation methods for dungeons and levels. In *Procedural Content Generation in Games*. Springer. 31–55.

Stephenson, M., and Renz, J. 2016. Procedural generation of levels for angry birds style physics games. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Summerville, A.; Guzdial, M.; Mateas, M.; and Riedl, M. O. 2016. Learning player tailored content from observation: Platformer level generation from video traces using lstms. In *12th Artificial Intelligence and Interactive Digital Entertainment Conference*.

Urbanek, J.; Fan, A.; Karamcheti, S.; Jain, S.; Humeau, S.; Dinan, E.; Rocktäschel, T.; Kiela, D.; Szlam, A.; and Weston, J. 2019. Learning to speak and act in a fantasy text adventure game. In *EMNLP*.

Van der Linden, R.; Lopes, R.; and Bidarra, R. 2013. Designing procedurally generated levels. In *9th Artificial Intelligence and Interactive Digital Entertainment Conference*.

Vara, C. F. 2014. Creating dreamlike game worlds through procedural content generation. In *Seventh Intelligent Narrative Technologies Workshop*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. *NIPS*.

Wang, T.-C.; Liu, M.-Y.; Zhu, J.-Y.; Tao, A.; Kautz, J.; and Catanzaro, B. 2018. High-resolution image synthesis and semantic manipulation with conditional gans. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 8798–8807.

Wu, L. Y.; Fisch, A.; Chopra, S.; Adams, K.; Bordes, A.; and Weston, J. 2018. Starspace: Embed all the things! In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Zhao, J.; Wang, T.; Yatskar, M.; Cotterell, R.; Ordonez, V.; and Chang, K. 2019. Gender bias in contextualized word embeddings. In *NAACL-HLT*, 629–634.

Zhu, S.; Urtasun, R.; Fidler, S.; Lin, D.; and Change Loy, C. 2017. Be your own prada: Fashion synthesis with structural coherence. In *Proceedings of the IEEE International Conference on Computer Vision*, 1680–1688.