Usnea: An Authorship Tool for Interactive Fiction using Retrieval Based Semantic Parsing

Ben Swanson*
Google
pwnr@google.com*

Boris Smus*
Google
smus@google.com

Abstract

The reader of a choose your own adventure novel and the user of a modern virtual assistant have a subtle similarity; both may, through the right lens, be viewed as engaging with a work of Interactive Fiction. This literary form emerged in the 1970s and has grown like a vine along the branch of modern technology, one guided by the advances of the other. In this work we weave together threads from the Interactive Fiction community and neural semantic parsing for dialog systems, defining the data model and necessary algorithms for a novel type of Interactive Fiction and open sourcing its accompanying authoring tool. Specifically, our work integrates retrieval based semantic parsing predicates into the branching story structures well known to the Interactive Fiction community, relaxing the relatively strict lexical options of preexisting systems.

1 Introduction

Interactive Fiction (IF) is a diverse genre of art and entertainment that is most well known in the context of video games, from text adventures (e.g. *Zork*), to classic point and click adventures such as *Monkey Island* to award winning modern games like *80 Days* (Time Magazine Game of the Year 2014). Less familiar to the general public is the literary tradition that recognizes IF as high art on par with the novel and poem, and produces compelling work collected in ever-growing online repositories[1]. The signature techniques of IF include branching story structure, multiple endings, and the use of lamps to solve complex problems.

IF was born, they say, in 1979 as ADVENT, a text based cave exploration game written by a father to amuse and delight his daughters (Niesz and Holland, 1984). Over the last 40 years, the interactive affordances of technology have grown from text in a terminal to include modern marvels such as graphics, audio, touchscreens, virtual reality, and speech recognition, and with their added complexity has come the creation of authorship software that allows nontechnical authors to harness these media. This is exemplified in Inform[2], a compiled programming language whose lines of code are themselves grammatical English sentences.

Authorship tools for IF define some structure of a story and provide suggested algorithms or software itself to realize this structure in a form that a reader can digest, which taken together we will call a *model specification*. Our particular model specification is inspired by recent work in neural dialog systems for virtual assistants. While ADVENT and Alexa may seem to have little in common, they are both clearly a turn taking interaction between a *system* and a *reader*[3]. Their internal workings are also similar, as it is no coincidence that the sub-genre of parser games like *Zork* shares a token in its name with the semantic parsers used in a dialog agents; they share the common ancestor of tree-structure parsers from the early days of computational linguistics (Woods, 1973).

We make use of retrieval based semantic parsing (Yao et al., 2019), a variant of nearest neighbor classification using inverse semantic similarity as its distance metric. One particular strength of this method is that the semantic similarity metric, or semantic kernel (Altınel et al., 2015), can be pretrained on general domain text pairs. Instantiating a domain specific semantic parser is tantamount to the definition of *exemplars*, strings paired with class labels that indicate their known semantics. Crucially, this is a task that can be done with no machine learning or programming background. It

* equal contribution
[1]https://www.ifarchive.org

[2]http://inform7.com
[3]We avoid the term "user" in this work in order to differentiate between users of the authoring tool (*authors*) and users of the resulting literary work (*readers*).

also avoids explicit formal semantic representations such as FrameNet (Baker et al., 1998), eliminating the learning curve required to bootstrap supervised discriminative semantic parsing systems such as Wang et al. (2015).

Our model grants the reader a more freeform mode of interaction compared to many other examples of IF; as long as the author can guide and anticipate the readers' inputs into close neighborhoods of their exemplars, the reader can both decide what to say and how to say it. For the author, we optimize for efficiency with features that collapse common patterns observed in user testing and components designed for the iterative tuning of the semantic parser exemplars. We also consider a third user type, the programmer who wishes to extend our open source library, by implementing our tool in Angular with extensive modularity via dependency injection. The general tool architecture is as an AppEngine hosted website, with Firebase for persistance.

2 Interactive Fiction

2.1 Overview

In its broadest sense, IF is an entertainment experience in which the reader is not a passive observer, or equivalently a sequence of *reader turns* and *system turns*. The reader turns need not be self-composed, as in the Choose Your Own Adventure novel where the system turns are the book chapters and the reader turns are selected from pre-written choices. While the system turns need not be simple text and often include audiovisual components, for clarity we restrict the scope of our discussion of IF in this work to the class in which our model lies where both user and system turns are purely textual.

Of the many members of this class, the two that we keep top of mind in our model design are parser games and dialogues. In parser games (Zorklikes), the reader commonly plays as the protagonist and system turns describe the reader's current observations of the world around them. The reader has a semantically rich but linguistically constrained set of options supplied by a semantic parser that recognizes combinatoric verb / object pairings. Our second focus is on dialogues, where the system and reader turns are both conversational utterances. This is motivated by the emergence of virtual assistants as a potential delivery mechanism for IF and the existing use of branching dialogues in story rich video games.

2.2 History, Abridged

While IF has rarely gained recognition in the mainstream media outside video games, it has been an area of literary and academic interest for several decades. Ziegfeld (1989) is especially prescient; in their discussion of the then nascent intersection of computer technology and storytelling, they explore possibilities for the use of branching that both include and expand on some of our own ideas, as well as raise thoughtful questions that remain unanswered today as to the eventual place of IF in art history.

The first documented piece of IF was ADVENT, written in the late seventies. Ziegfeld (1989) describes some other early IF work done in collaboration with well-known authors such as Michael Crichton, Ray Bradbury, and Arthur C Clarke. This early notoriety has faded, but the community of writers has remained continuously productive; Montfort (2005) provides a tour of the first two and a half decades of IF, and the conference Narrascope[4] is a hub for modern authors.

Many authorship tools have been created for IF, often with general purpose features that have enabled unanticipated expressions of their models' underlying mechanisms. Of particular importance is the authorship tool Twine[5], whose easy to use interface has inspired a surge of IF work.

Outside of pure entertainment, IF has proven potential in education(Squire, 2003), specifically as a language learning tool(Baltra, 1990). One recent example that uses a state based model specification similar to our own is Ramanarayanan and LaMar (2018), in which IF is used an assessment tool and a correlation is demonstrated between proficiency level and a learned student specific MDP parameter.

3 Related Work

In NLP literature, perhaps the closest touchpoint to this work is Jonell et al. (2018), where a open domain chatbot is constructed by crowdsourcing appropriate responses for known chat histories. They describe a nightly iteration process in which the day's user utterances are clustered using a similarity function into paraphrase clusters and then passed to crowdworkers to provide appropriate followup system turns. They incrementally grow a

[4]https://narrascope.org/

[5]http://twinery.org

directed graph based chatbot model of a similar structure to our own, but approach the authorship process in a drastically different manner and target only the desiderata of chitchat dialog as opposed to the general class of IF.

(Koller et al., 2018) also creates a directed graph editing tool for dialogs, with API hooks allowing the control of Lego Mindstorms robots. While this application would at first glance seem quite different, the data model is similar. The visual similarity of our tools alone demonstrates their near isomorphism; in both cases the authorship process is tantamount to the definition of a graph topology and the filling of schema for graph components. The major difference arises instead from our use of retrieval based semantic matching and focus on IF.

Our use of retrieval based (or paraphrase based) methods draws from recent work in semantic parsing (Berant and Liang, 2014) (Fader et al., 2013) and one-shot classification (Koch et al., 2015) powered by the growing availability of general domain semantic similarity training data (Yang et al., 2018) (Cer et al., 2018). One notable addition in our work is the introduction of anti-examples for tuning parser quality.

4 Model Specification

Our model can be considered a marriage between the intuitive design principles of Twine and modern methods in dialog systems. A typical dialog system design consists of components responsible for Language Understanding, Dialog Management, and Language Generation (Bohus and Rudnicky, 2009) (Shum et al., 2018).

4.1 Language Understanding

We employ retrieval based semantic matching for Language Understanding (LU), a close variant of nearest neighbor classification. We assume a finite set of unique semantic intents that our LU system can recognize. Formally, this method requires

$$\begin{aligned}
\mathcal{U}&: \text{a set of exemplar strings}\\
\mathcal{A}&: \text{a set of anti-exemplar strings}\\
\mathcal{E}&: \text{a set of semantic intents}\\
\mathcal{M} : \cup\{\mathcal{U}, \mathcal{A}\} \to \mathcal{E}&: \text{a mapping to semantic intents}\\
\mathcal{D}(x, y) \to \mathbb{R}&: \text{a string similarity function}
\end{aligned}$$

and semantic parsing is done with Nearest Neighbor (NN) classification using $\frac{1}{\mathcal{D}}$ as the distance metric. The classifier has an additional rejection option, triggered when no member of $\mathcal{U}$ produces a similarity with the user utterance that exceeds some author determined τ.

We augment the traditional NN classification algorithm with anti-examples, letting $\mathcal{A}$ be a set of anti-example strings. Their specific functionality is outlined below in Algorithm 1, which shows the full semantic parsing algorithm for an input x.

Algorithm 1: Semantic Parsing of input x

```
Result: Semantic intent e $\in \mathcal{E}$ or REJECT
$\mathcal{S} \leftarrow \cup\{\mathcal{U}, \mathcal{A}\}$;
while $\mathcal{S} \neq \varnothing$ do
    $z^* \leftarrow \arg\max_{z \in \mathcal{S}} \mathcal{D}(x, z)$;
    if $\mathcal{D}(x, z^*) \geq \tau$ then
        if $z^* \in A$ then
            forall $w \in \mathcal{S} \mid \mathcal{M}(w) = \mathcal{M}(z^*)$ do
                $\mathcal{S} \leftarrow \mathcal{S} - w$
            end
        else
            return $\mathcal{M}(z)$
        end
    else
        return REJECT
    end
end
return REJECT
```

4.2 Dialogue Management

Our Dialogue Management (DM) system is based on a directed graph, representing the definition and evolution of dialog state by enumerating the finite set of all possible states (the graph nodes), and the allowed transitions between them (the graph edges). We assume a unique global start node and allow multiple end nodes. A node's outgoing edges are either directly associated with a semantic enum index as described above or marked as a RepeatedFail edge. The DM traverses the former when its outbound edge is returned by the semantic parser and the latter when the semantic parser has returned REJECT n times in a row, for some author controlled n.

The fundamental task of the DM is to determine $\mathcal{U}$ and $\mathcal{A}$ to be used in Algorithm 1, and a Markov-like memoryless model that returns all outgoing edges of the current node is sufficient for simple dialogues. However, to enable more complex storytelling opportunities we add a global state which we call the *World*.

The World is implemented as a string to string map, and each edge may assigned a *Condition* and *Mutation*. A Condition is any boolean predicate on the World, which allows its edge as a candidate only if it is satisfied. A full Condition is the conjunction of boolean subconditions, where each subcondition is the presence or absence of a specific key or key/value pair in the World. A Mutation is a list of operations to be performed on the World, and is executed upon its edge's traversal. We permit the use of mutations that set key value pairs or delete keys from the World.

We add one further augmentation: a boolean AutoAdvance property on a node that, if enabled, immediately randomly traverses an outgoing edge instead of waiting for the reader turn. This simple feature provides flexibility in graph design, allowing patterns we observed a desire for in user testing.

- **Node**
  - PROMPT : STRING
  - REPROMPT : STRING
  - AUTOADVANCE? : BOOLEAN
- **Edge**
  - MUTATIONS : ⧫
  - CONDITIONS : ⧫
  - EITHER
    - MATCHCANDIDATES : ⧫
  - OR
    - REPEATEDFAILMESSAGE : STRING
    - REPEATEDFAILN : NUMBER

Figure 1: Schema for graph specification, omitting standard graph topology information for a single source directed graph. Fields with complex types ⧫ are discussed in Section 5.

4.3 Language Generation

Language Generation is simplified nearly to its limit in our specification; each node is directly associated with a system turn, its Prompt, produced when the node is visited. To allow for more forgiving reader experiences we add two additional optional sources of system text. The first is a Reprompt, shown in response to a REJECT from the semantic parser, allowing the author to guide the reader towards a sucessfully parseable utterance. The second source of system text is an optional message attached to RepeatedFail edges, to allow acknowledgement of the perhaps unexpected transition.

5 Authoring Tool

Authoring under our model specification requires both defining the topology of a directed graph and filling the schema of node and edge properties (Figure 1). Some schema fields have complex types, which have multiple isomorphic data models of negligible difference provided they fulfill their functional purpose. MUTATIONS specifies a set of string to string map mutations and CONDITIONS a set of simple boolean key/value lookups, while MATCHCANDIDATES are tuples of exemplar strings and booleans indicating if they are anti-examples.

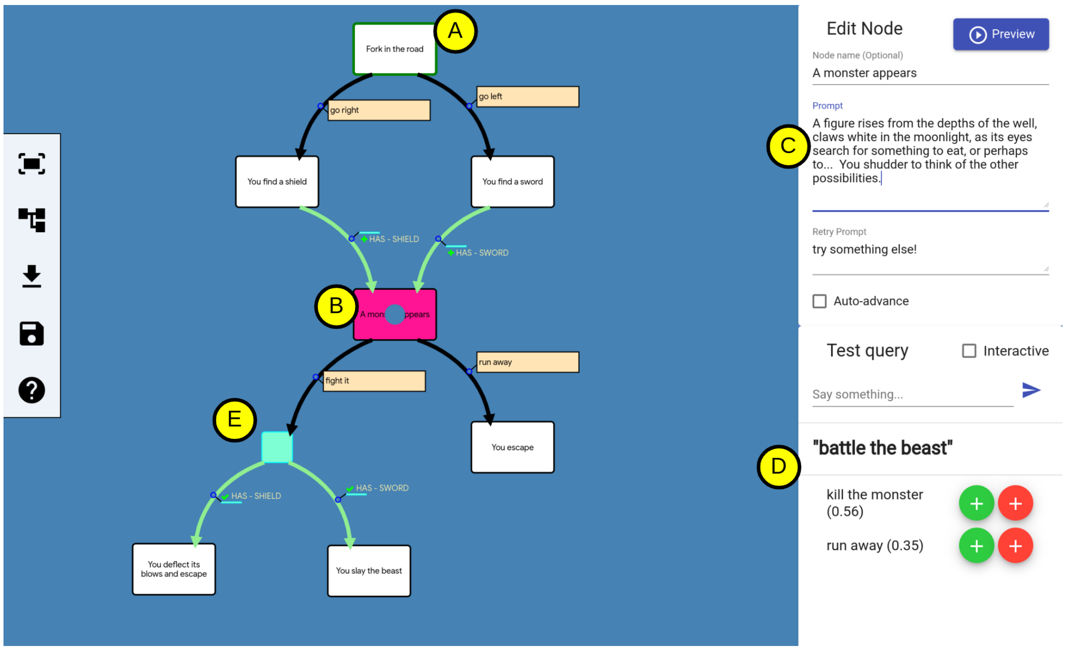
5.1 Main Authoring Tool

The home screen of the tool (Figure 2) consists of two panels. On the left is the graph editor, which visualizes and provides editing options for the graph topology. On the right is the the graph inspector, which allows the editing of the schema data outlined in Figure 1.

The graph inspector has a focus that determines the schema editing options that it displays. Clicking on a node or edge in the graph editor will focus it on that node or edge, while clicking on the background will focus the inspector on an editor for global hyperparameters such as the project name and persistence filename. Both node and edge editors contain buttons that delete them from the graph, a sole exception to topology being the domain of the graph editor alone.

The graph editor has its own toolbar containing the following actions: Save, CopyToClipboard, AutoZoom, AutoLayout. We consider all of these to be self-documenting with the exception of AutoLayout, whose intention is to produce a well packed organization of the graph through the following process.

Our AutoLayout algorithm first topologically sorts the graph and checks it for cycles. If no cycles exist then for each node n we calculate $l(n)$, the length of the longest path from the start node. The nodes are then arranged in rows of regular spacing such that each n appears in row $l(n)$. A best effort is made to choose a relative ordering of nodes in each row that minimizes visually tangled webs of edges, and we allow and rely on manual adjustment to achieve the author's ideal visual organization.



Edit Node

Node name (Optional)
A monster appears

Prompt
A figure rises from the depths of the well, claws white in the moonlight, as its eyes search for something to eat, or perhaps to... You shudder to think of the other possibilities.

Retry Prompt
try something else!

☐ Auto-advance

Test query ☐ Interactive

Say something...

"battle the beast"

kill the monster (0.56) + +

run away (0.35) + +

Figure 2: The main editor window, showing a story with start node A. The node B is selected, and so its details appear in the inspector on the right. Note that its display name is distinct from the richer game text (C). The results of using the Node Tester on the potential input "battle the beast" are shown in (D). They demonstrate that this utterance would cause a transition to node E, an AutoAdvance node that would immediately transition to one of two end states depending on the player's previous choices.

5.2 Node Tester

The authorship process is by nature iterative, and we recognize the importance of facilitating the editing of existing content. In particular, we anticipate a feedback loop in which the author observes or imagines reader turns and wishes to test them and tune the semantic matching model based on the results. For example, an author might want to verify that some reader turn matches, or does not match, a particular edge and add it to that edge as either a positive or anti-example if the desired behavior is not observed.

In aid of this use case, we provide an interactive node tester, found in the graph inspector when focused on a node (lower right in Figure 2). This tool allows the author to probe with a potential utterance and view the most similar exemplar from each possible edge and their similarity scores. With one click, the author can then add their probe utterance to an existing edge as a positive or anti-example, or create a new edge and target node with the probe utterance as a positive exemplar on the new edge.

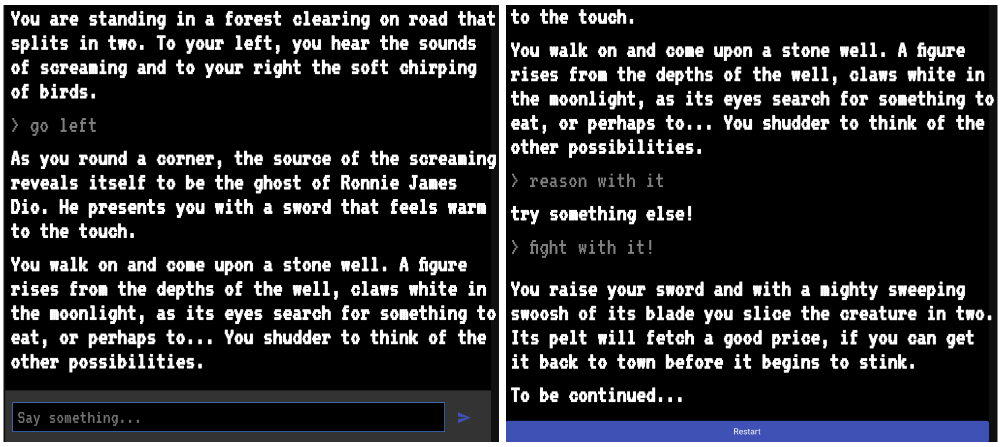
5.3 Preview Mode

We follow the footsteps of Twine in the integration of a modest implementation of a reading program in the authoring tool, and allow the author to trigger this "Preview Mode" starting from any node in the graph. While we intend the debugging of semantic matches to be more easily performed using the Node Tester, Preview Mode is useful for authors to get early feedback on their work without investing in bespoke reader software.

As an alternative, we also add a toggle for Interactive Mode to the Node Tester described above; in this mode if the probe user utterance would match an edge, the editor selection and inspector focus automatically move to that edge's target node. This is functionally equivalent to (although certainly less immersive than) the full Preview Mode and allows the authors to maintain their cognitive connection with the graph editor and inspector while testing the boundaries of the semantic parser.

5.4 Implementation Considerations

We implement our tool in Angular, a web application platform that supports dependency injection. This not only facilitates the introduction of custom node metadata and UI, but also enables modularity in the services that drive the editor. Two key services, the persistence (save/load) mechanism, and the semantic similarity function $\mathcal{D}$, are injected and as such easy to override with custom code. We hope that allowing flexibility in the storage medium


You are standing in a forest clearing on road that splits in two. To your left, you hear the sounds of screaming and to your right the soft chirping of birds.

› go left

As you round a corner, the source of the screaming reveals itself to be the ghost of Ronnie James Dio. He presents you with a sword that feels warm to the touch.

You walk on and come upon a stone well. A figure rises from the depths of the well, claws white in the moonlight, as its eyes search for something to eat, or perhaps to... You shudder to think of the other possibilities.

to the touch.

You walk on and come upon a stone well. A figure rises from the depths of the well, claws white in the moonlight, as its eyes search for something to eat, or perhaps to... You shudder to think of the other possibilities.

› reason with it

try something else!

› fight with it!

You raise your sword and with a mighty sweeping swoosh of its blade you slice the creature in two. Its pelt will fetch a good price, if you can get it back to town before it begins to stink.

To be continued...

Figure 3: Preview Mode, displaying two successive screenshots of a reading of the story shown in Figure 2. Note that in the second panel the reader response "reason with it" is rejected, leading to the use of the retry prompt.

will allow for easier integration with custom reader software and injection of $\mathcal{D}$ is in anticipation of future superior models, some options for which are discussed below.

We provide one instantiation of each service. Our persistence service implementation uses Firebase, a popular cloud database, with instructions on how to configure Firebase credentials when launching a server in the repository's README. For $\mathcal{D}$, our implementation uses the open source Universal Sentence Encoder of Cer et al. (2018).

We note that significantly better similarity models are easily attainable as Cer et al. (2018) predates recent breakthrough techniques in pretraining (Devlin et al., 2018) (Yang et al., 2019). Furthermore, the dataset of paraphrase pairs used to train the encoder of Cer et al. (2018) is drawn from the SNLI corpus (Bowman et al., 2015) with additional unsupervised multi-task training data taken from sources such as Wikipedia and web news; this broad coverage will likely give reasonable results for any domain, but the specific flavor of IF or its subtypes (e.g. dialouge) would likely benefit from domain specific fine-tuning.

6 Discussion

We present a flexible model specification for a new flavor of Interactive Fiction inspired by recent trends in retrieval based dialog systems, and provide an accompanying authorship tool. Our tool is deployed as an AppEngine app, is written in Angular, and is open source [6].

Our model specification makes use of semantic matching based predicates to traverse a directed graph, tracing out a "reading" of the piece. The use of semantic matching allows active reader ideation of their role while remaining within guard rails that maintain narrative cohesion. Furthermore, its use of text based exemplars in a non-parametric model with a pretrained semantic kernel permits the iterative tuning of the semantic parsing system by an author with no programming or machine learning background.

Interactive Fiction is an art form with an uncertain future that is connected in no small way to its proximity to games and the social norms separating games and fine art. Ziegfeld (1989) muses that IF may either be like American poetry waiting for its Walt Whitman, or like the cutup poetry fad of the beat poets, bound for obscurity. They perhaps did not expect that the question would remain unanswered for thirty years. We hope foremost that authors will enjoy using our tool to create something they care about, and that readers will enjoy their creations.

[6]http://borismus.github.io/usnea

References

Berna Altınel, Murat Can Ganiz, and Banu Diri. 2015. A corpus-based semantic kernel for text classification by using meaning values of terms. *Engineering Applications of Artificial Intelligence*, 43:54–66.

Collin F Baker, Charles J Fillmore, and John B Lowe. 1998. The berkeley framenet project. In *Proceedings of the 17th international conference on Computational linguistics-Volume 1*, pages 86–90. Association for Computational Linguistics.

Armando Baltra. 1990. Language learning through computer adventure games. *Simulation & Gaming*, 21(4):445–452.

Jonathan Berant and Percy Liang. 2014. Semantic parsing via paraphrasing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1415–1425.

Dan Bohus and Alexander I Rudnicky. 2009. The ravenclaw dialog management framework: Architecture and systems. *Computer Speech & Language*, 23(3):332–361.

Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. 2015. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*.

Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. 2018. Universal sentence encoder. *arXiv preprint arXiv:1803.11175*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Anthony Fader, Luke Zettlemoyer, and Oren Etzioni. 2013. Paraphrase-driven learning for open question answering. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1608–1618.

Patrik Jonell, Mattias Bystedt, Fethiye Irmak Dogan, Per Fallgren, Jonas Ivarsson, Marketa Slukova, Ulme Wennberg, José Lopes, Johan Boye, and Gabriel Skantze. 2018. Fantom: A crowdsourced social chatbot using an evolving dialog graph. *Proc. Alexa Prize*.

Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. 2015. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2. Lille.

Alexander Koller, Timo Baumann, and Arne Köhn. 2018. Dialogos: Simple and extensible dialog modeling.

Nick Montfort. 2005. *Twisty Little Passages: an approach to interactive fiction*. Mit Press.

Anthony J Niesz and Norman N Holland. 1984. Interactive fiction. *Critical Inquiry*, 11(1):110–129.

Vikram Ramanarayanan and Michelle LaMar. 2018. Toward automatically measuring learner ability from human-machine dialog interactions using novel psychometric models. In *Proceedings of the Thirteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 117–126.

Heung-Yeung Shum, Xiao-dong He, and Di Li. 2018. From eliza to xiaoice: challenges and opportunities with social chatbots. *Frontiers of Information Technology & Electronic Engineering*, 19(1):10–26.

Kurt Squire. 2003. Video games in education. *Int. J. Intell. Games & Simulation*, 2(1):49–62.

Yushi Wang, Jonathan Berant, and Percy Liang. 2015. Building a semantic parser overnight. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1332–1342.

William A Woods. 1973. Progress in natural language understanding: an application to lunar geology. In *Proceedings of the June 4-8, 1973, national computer conference and exposition*, pages 441–450. ACM.

Yinfei Yang, Steve Yuan, Daniel Cer, Sheng-Yi Kong, Noah Constant, Petr Pilar, Heming Ge, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. 2018. Learning semantic textual similarity from conversations. *arXiv preprint arXiv:1804.07754*.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, pages 5754–5764.

Jiaqi Yao, Keren Wang, Zhengguo Xu, and Jikun Yan. 2019. Classvector: A parameterized prototype-based model for text classification. In *Proceedings of the 2019 11th International Conference on Machine Learning and Computing*, pages 322–326. ACM.

Richard Ziegfeld. 1989. Interactive fiction: A new literary genre? *New Literary History*, 20(2):341–372.