A Creative Dialog Generator for Fallout 4

Khalid Alnajjar
khalid.alnajjar@helsinki.fi
Dept. of Computer Science & HIIT, University of Helsinki
Helsinki, Finland

Mika Hämäläinen
mika.hamalainen@helsinki.fi
Dept. of Digital Humanities, University of Helsinki
Helsinki, Finland

ABSTRACT

This software demonstration describes a mod for Fallout 4 that will adapt in-game dialog to the context of the current state of the game. The dialog is generated by a computationally creative back-end software during the game play. The mod solves the problem of Fallout 4 not supporting dynamically generated dialog by showing dialog in an overlay application on top of the game window.

CCS CONCEPTS

• **Computing methodologies → Discourse, dialogue and pragmatics**; **Natural language generation**; *Natural language processing*; • **Applied computing** → Computer games.

KEYWORDS

dialog generation, contextual adaptation, video game dialog, computational creativity

ACM Reference Format:
Khalid Alnajjar and Mika Hämäläinen. 2019. A Creative Dialog Generator for Fallout 4. In *The Fourteenth International Conference on the Foundations of Digital Games (FDG '19), August 26–30, 2019, San Luis Obispo, CA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3337722.3341824

1 INTRODUCTION

Fallout 4[1] is an open world RPG (role-playing game) that gives the player a great freedom to explore the wasteland and complete dozens of side quests. The game has a massive number of hand-written dialog to cater for credible characters the player might not even encounter if he is not undertaking all the side quests in the game.

However, the predefined dialog comes with a problem of adaptability. The dialog will stay the same regardless of the condition of the player or the wider state of completion of the game. A player who has slaughtered an entire city will be still called a hero in the NPC (non-player character) dialog in the next city, and so on.

For this particular reason, we have decided to seek for a solution to the poor contextual adaptability of the Fallout 4 dialog. We describe our work on a computationally creative generative system that can take in a piece of existing dialog and adapt it to fit the context of the game state dynamically during the game play. Our generator is not just a piece of separate code, but it actually integrates with Fallout 4 via a mod we have developed for the purpose of showing externally generated dialog in the game.

While dialog generation for video games and digital media has received some research attention in the past [3, 9, 11, 13], for the best of our knowledge, our approach is the first one using machine learning to generate a virtually unlimited number of novel contextually adapted dialog for a video game. To further widen the impact of our research, we are releasing the code for the mod with an open source license on Zenodo[2] together with this paper.

Our work relates to the field of computational creativity, in which a computer is used to produce artefacts that would be deemed creative if observed by people cf. [17]. Computational creativity has grown as a field with a wide range of topics covered throughout the past decade such as slogan generation [1, 14, 15], poem generation [5, 6] and humor generation [2, 18]. Our demo brings computational creativity close to a real human user in a video game setting.

2 SHOWING GENERATED DIALOG IN FALLOUT

Fallout 4 does not support showing dynamically generated dialog, but instead, the game engine expects all the dialog be written and voice acted beforehand. Nothing new can be displayed during the run-time. For our approach to be integrated into the game, this requires a workaround to solve the problem. Our solution involves an external overlay application that shows the dialog on top of the game window. The whole process is depicted in Figure 1.

Our mod adds a creative NPC character called *Mr Creative* into *Diamond City*. This character has empty dialog with empty dialog options, so that the actual generated dialog can be shown instead. A script written in the Fallout 4 modding language Papyrus is attached to the NPC and it will log out different game state variables when the player initiates dialog with the NPC. The logged game state has 8 values of the player object (such as health, strength, endurance) and 104 values of the game stats (such as quests completed, monsters killed, locks picked and so on).

The log file is continuously checked by a log daemon, and once new log is available, the daemon will parse the game state variables output by the Papyrus script and send an HTTP request to the generator script that is running as a Flask[3] based web application. This will inform the generator that the player has entered in a dialog with the creative NPC character.

[1]Fallout is a registered trademark of Bethesda Softworks LLC in the U.S. and/or other countries. All Rights Reserved.

[2]https://doi.org/10.5281/zenodo.3232863

[3]http://flask.pocoo.org/

FDG '19, August 26–30, 2019, San Luis Obispo, CA, USA
© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-7217-6/19/08…$15.00
https://doi.org/10.1145/3337722.3341824

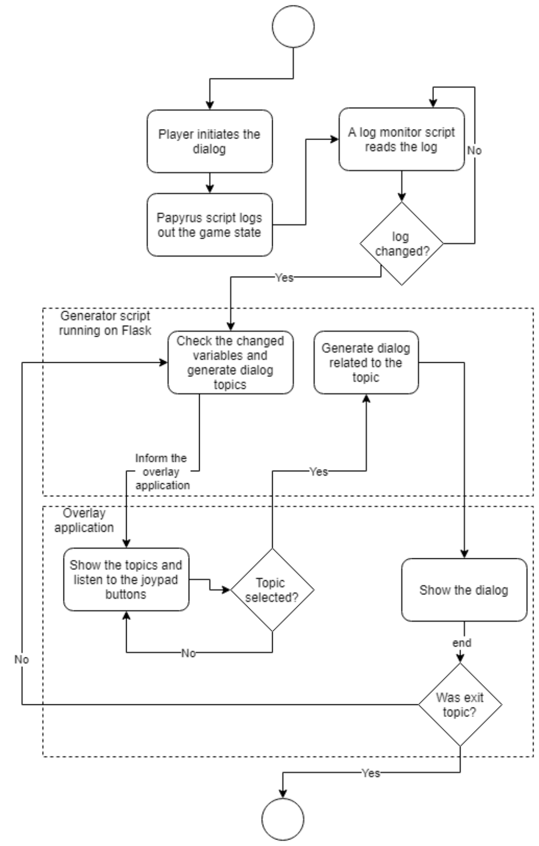
Player initiates the dialog
Papyrus script logs out the game state
A log monitor script reads the log
log changed?
No
Yes
Generator script running on Flask
Check the changed variables and generate dialog topics
Generate dialog related to the topic
Inform the overlay application
Yes
Overlay application
Show the topics and listen to the joypad buttons
Topic selected?
No
Show the dialog
end
Was exit topic?
No
Yes

Figure 1: The flowchart of the mod displaying the generated dialog.

The generator compares the game state to the previously stored one and picks the changed game state variables as potential dialog options. Out of these variables, three are picked at random as dialog options to be shown to the player. The fourth dialog option is always fixed as a dialog ending *bye* option. Once the dialog options are ready, the generator will make an HTTP request to the overlay application with the options.

Once the overlay application gets the new dialog options, it makes its application window visible and reshapes it to the shape of the dialog option text. This way it does not look like there was an external window on top of the game window as seen in Figure 2. The overlay application starts to listen to the joy-pad buttons. The player can pick a dialog option in Fallout 4 by pressing one of the following Xbox 360 controller buttons: X, Y, B or A. When the player makes the choice, the overlay application registers the same button press as Fallout 4, sends an HTTP request to the generator script with the picked option and makes itself invisible.

The generator generates two pieces of utterances. One to be displayed as uttered by the player and another by the NPC. As generation is a computationally demanding process, the generator application generates possible dialogues into a cache before they are actually needed. After the generation, the overlay application is informed by another HTTP request.

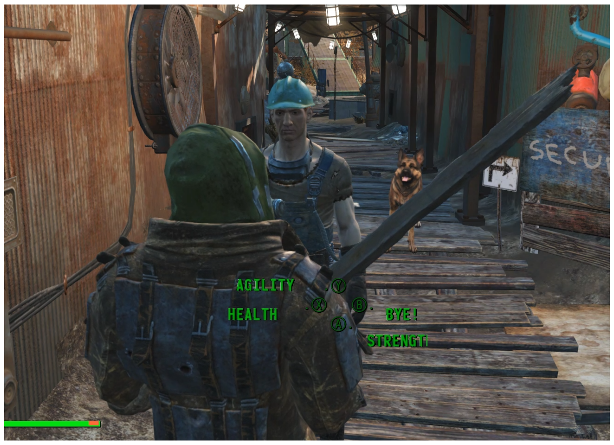

Figure 2: Dialog options shown in the game.

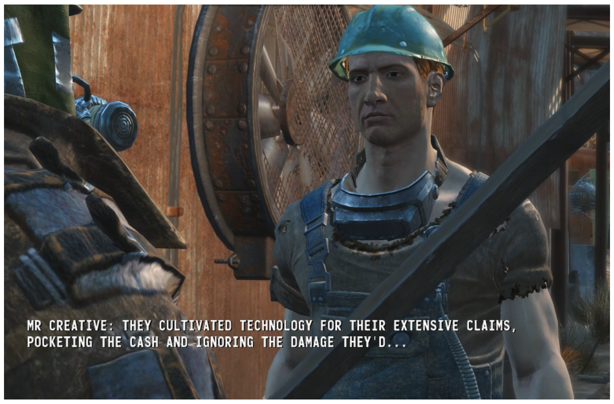
MR CREATIVE: THEY CULTIVATED TECHNOLOGY FOR THEIR EXTENSIVE CLAIMS, POCKETING THE CASH AND IGNORING THE DAMAGE THEY'D...

Figure 3: The actual dialog shown by the overlay application.

Finally the overlay application starts showing the generated dialog as seen in Figure 3. The dialog is shown for a predefined duration of time that corresponds to how long Fallout takes to show the empty dialog of the NPC character. After the dialog sequence is over, either new options are requested from the generator, or if a dialog ending option was picked, the overlay application just hides its application window and the dialog finishes.

3 DIALOG GENERATOR

The dialog is generated by picking at random two consecutive utterances in the existing Fallout 4 dialogues. These are the initial utterances that will undergo the contextual adaption process given the player picked topic. The topics are the names of the game state variables that have increased from the last time the player had a conversation with the NPC.

Each game state variable is by hand assigned with four key words describing an increase in that variable. For example, for *Locks Picked*, the words are *burglar*, *crook*, *robber* and *suspicious*, and for *Locations Discovered*: *wanderer*, *discovery*, *traveller* and *explore*. The four seed words associated with the chosen variable together with the two randomly picked existing utterances serve as an input for the generation pipeline as shown in Figure 1.

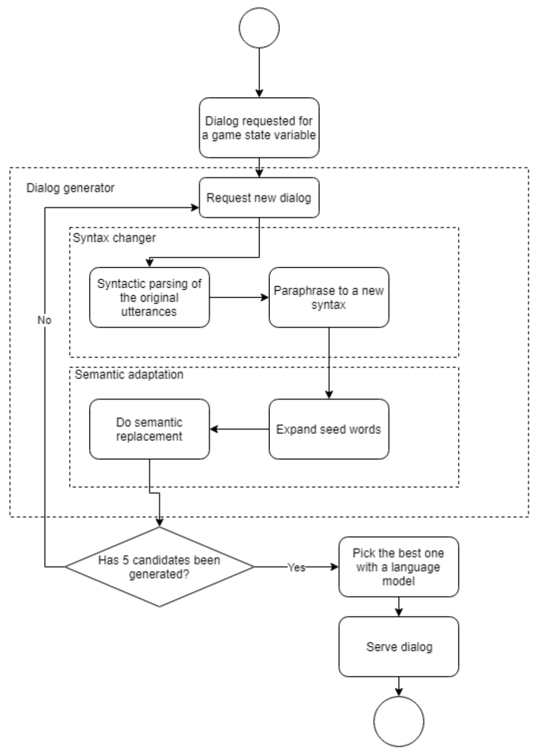
Dialog requested for a game state variable
Dialog generator
Request new dialog
Syntax changer
Syntactic parsing of the original utterances
Paraphrase to a new syntax
No
Semantic adaptation
Do semantic replacement
Expand seed words
Has 5 candidates been generated?
Yes
Pick the best one with a language model
Serve dialog

Figure 4: The flowchart of the dialog generation pipeline.

The dialog adaptation process starts by parsing the dialog to an abstract syntax with spaCy [8]. By abstract syntax we mean a structure, where closed class part-of-speech words are replaced with a placeholder indicating only their POS tag. This abstract syntax is then paraphrased by using a sequence-to-sequence syntactic paraphrasing model trained with OpenNMT [10]. We train the model with otherwise the default settings but use the copy attention mechanism and use a BRNN (bi-directional recurrent neural network) encoder instead of the default one-directional RNN encoder.

The model is trained by using Opusparcus [4] subtitle corpus tailored for paraphrasing as it has parallel data of subtitles written by different translators for the same movies. We convert the data-set into the aforementioned abstract syntax form and use the first one hundred thousand parallel sentences where the syntactic structure of the paraphrase is different from the original.

The syntax is changed by randomly picking from the top 5 paraphrased syntactic structures generated by the OpenNMT model. We replace the placeholders of the new syntax with the words from the original utterance. This step only changes the syntax for more structural variety in the final output. Changing the vocabulary semantically is done by the next step of the pipeline.

The semantic change process begins after the paraphrasing phase. The process can be divided into 4 steps, which are 1) finding and selecting words in the utterance to change, 2) retrieve semantically similar words (candidates) to the selected ones, 3) evaluate the fitness of these candidates and 4) apply the semantic change.

During the first step, the process highlights words in the utterance that match a list of predefined parts-of-speech (e.g. verbs, adjectives, nouns, …etc). Out of the highlighted words, a random non-empty subset of words is chosen to be replaced. In following step, the processes uses a pre-trained semantic model [12] for retrieving the top k semantically similar words (candidates) to each highlighted word. We empirically set k to be 300. The retrieved candidates are then evaluated during the next phase of the process, where the process checks whether these candidates are different from the original word, match its part-of-speech, are semantically similar to it and are semantically similar to the context (i.e. the passed seed words). To measure the semantic similarity to the context, we represent words describing it as a single vector in the semantic model (called centroid) by averaging their corresponding vectors. The semantic similarity to the context is measured by calculating the dot product of the candidates' vectors with the centroid (i.e. cosine similarity).

Any candidate word that does not meet these conditions is pruned out. From the remaining candidates, a random candidate for each highlighted word is selected to be its replacement. Afterwards, the replacements are applied and the resulting utterance is returned.

Finally, the new replacement words are inflected to match the original morphology by Pattern [16]. The whole generative pipeline is repeated 5 times after which the output dialogues are ranked by using a 3-gram language model, picking the highest ranking dialog and serving that for the player. The language model is trained with all the data in Opusparcus with a language model tool called KenLM [7].

4 CONCLUSIONS

In this software demonstration paper, we have presented our work on contextually adapting dialog in Fallout 4 to a set of game state variables. The work has involved overcoming the practical problem of showing dynamically created dialog in the game that only supports static, predefined dialog. In addition to that, we have presented our initial work on using modern machine learning methods in generating adapted dialog in a computationally creative fashion.

The mod presented in this paper is fully functional with Fallout 4 and can be, with minor adjustments, installed on any Windows machine running the game. Therefore, the source code of the mod presented in this paper is made openly available on Zenodo[4].

REFERENCES

[1] Khalid Alnajjar, Hadaytullah Hadaytullah, and Hannu Toivonen. 2018. "Talent, Skill and Support." A Method for Automatic Creation of Slogans. In *Proceedings of the 9th International Conference on Computational Creativity (ICCC 2018)*. Association for Computational Creativity, Salamanca, Spain, 88–95.
[2] Khalid Alnajjar and Mika Hämäläinen. 2018. A Master-Apprentice Approach to Automatic Creation of Culturally Satirical Movie Titles. In *Proceedings of the 11th International Conference on Natural Language Generation*. Association for Computational Linguistics, Tilburg University, The Netherlands, 274–283.
[3] Marc Cavazza and Fred Charles. 2005. Dialogue generation in character-based interactive storytelling. In *Proceedings of the First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. AAAI Press, Marina del Rey, California, 21–26.

[4]https://doi.org/10.5281/zenodo.3232863

[4] Mathias Creutz. 2018. Open Subtitles Paraphrase Corpus for Six Languages. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC-2018)*. European Languages Resources Association (ELRA), Miyazaki, Japan, 1364–1369.

[5] Hugo Gonçalo Oliveira. 2017. A Survey on Intelligent Poetry Generation: Languages, Features, Techniques, Reutilisation and Evaluation. In *Proceedings of the 10th International Conference on Natural Language Generation*. Association for Computational Linguistics, Santiago de Compostela, Spain, 11–20.

[6] Mika Hämäläinen. 2018. Harnessing NLG to Create Finnish Poetry Automatically. In *Proceedings of the Ninth International Conference on Computational Creativity*. Association for Computational Creativity, Salamanca, Spain, 9–15.

[7] Kenneth Heafield. 2011. KenLM: Faster and Smaller Language Model Queries. In *Proceedings of the EMNLP 2011 Sixth Workshop on Statistical Machine Translation*. Association for Computational Linguistics, Edinburgh, Scotland, United Kingdom, 187–197. https://kheafield.com/papers/avenue/kenlm.pdf

[8] Matthew Honnibal and Ines Montani. 2017. spaCy 2: Natural Language Understanding with Bloom Embeddings, Convolutional Neural Networks and Incremental Parsing. *To appear* (2017).

[9] Christopher Kerr and Duane Szafron. 2009. Supporting dialogue generation for story-based games. In *Proceedings of the Fifth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. AAAI Press, Stanford, California, 154–160.

[10] Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander Rush. 2017. OpenNMT: Open-Source Toolkit for Neural Machine Translation. In *Proceedings of ACL 2017, System Demonstrations*. Association for Computational Linguistics, Vancouver, Canada, 67–72.

[11] Jonathan Lessard, Etienne Brunelle-Leclerc, Timothy Gottschalk, Marc-Antoine Jetté-Léger, Odile Prouveur, and Christopher Tan. 2017. Striving for Author-friendly Procedural Dialogue Generation. In *Proceedings of the 12th International Conference on the Foundations of Digital Games (FDG '17)*. ACM, New York, NY, USA, Article 67, 6 pages.

[12] Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. 2017. Advances in Pre-Training Distributed Word Representations. *CoRR* abs/1712.09405 (2017). arXiv:1712.09405

[13] Hannah Morrison and Chris Martens. 2017. A Generative Model of Group Conversation. In *Proceedings of the 12th International Conference on the Foundations of Digital Games (FDG '17)*. ACM, New York, NY, USA, Article 66, 7 pages.

[14] Gözde Özbal, Daniele Pighin, and Carlo Strapparava. 2013. BRAINSUP: Brainstorming Support for Creative Sentence Generation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Sofia, Bulgaria, 1446–1455.

[15] Andraž Repar, Matej Martinc, Martiň Znidaršič, and Senja Pollak. 2018. BISLON: BISociative SLOgaN generation based on stylistic literary devices. In *Proceedings of the 9th International Conference on Computational Creativity (ICCC 2018)*. Association for Computational Creativity, Salamanca, Spain, 248–255.

[16] Tom De Smedt and Walter Daelemans. 2012. Pattern for python. *Journal of Machine Learning Research* 13, Jun (2012), 2063–2067.

[17] Dan Ventura. 2014. Can a Computer be Lucky? And Other Ridiculous Questions Posed by Computational Creativity. In *Artificial General Intelligence*, Ben Goertzel, Laurent Orseau, and Javier Snaider (Eds.). Springer International Publishing, Cham, 208–217.

[18] Thomas Winters, Vincent Nys, and Daniel De Schreye. 2019. Towards a General Framework for Humor Generation from Rated Examples. In *Proceedings of the Tenth International Conference on Computational Creativity*. Association for Computational Linguistics, Charlotte, North Carolina, U.S., 274–281.