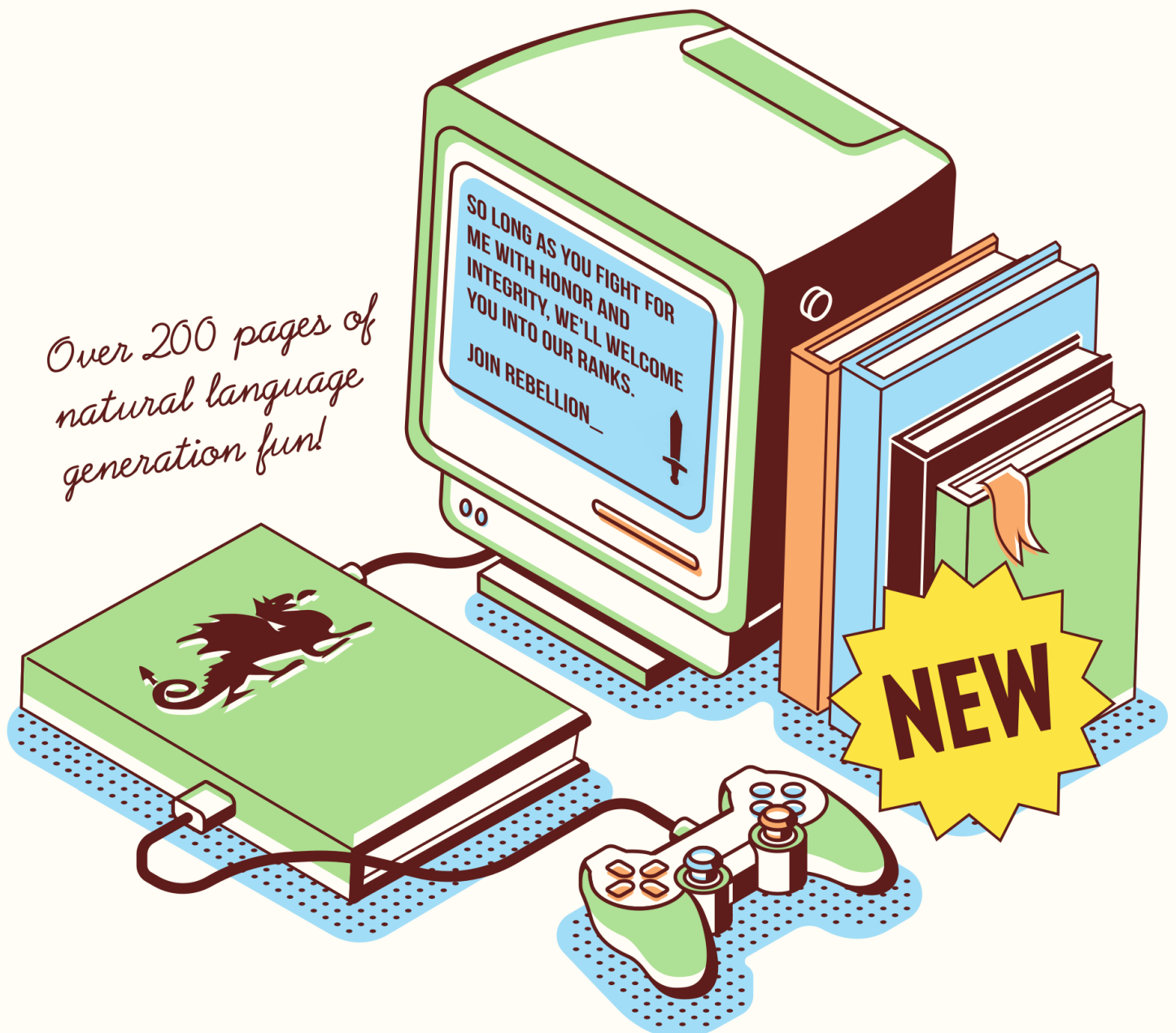


Judith van Stegeren

FLAVOR TEXT GENERATION FOR ROLE-PLAYING VIDEO GAMES

PHD THESIS



FLAVOR TEXT GENERATION FOR ROLE-PLAYING VIDEO GAMES

Judith van Stegeren

**FLAVOR TEXT GENERATION
FOR ROLE-PLAYING VIDEO GAMES**

DISSERTATION

to obtain
the degree of doctor at the Universiteit Twente,
on the authority of the rector magnificus,
prof. dr. ir. A. Veldkamp,
on account of the decision of the Doctorate Board
to be publicly defended
on Friday 25 March 2022 at 16.45 hours

by

Judith Emma van Stegeren

Born on the 29th of September, 1990
in Utrecht, The Netherlands

This dissertation has been approved by:

Supervisor
Co-supervisor

prof. dr. D.K.J. Heylen
dr. M. Theune



This publication is part of the project
Data2Game (with project number 055.16.114)
of the research programme Professional Games
for Professional Skills which is financed by the
Dutch Research Council (NWO).

**Human Media
Interaction.**

The research reported in this thesis has been
carried out at the Human Media Interaction
(HMI) research group of the University of
Twente.



SIKS Dissertation Series no. 2022-1

The research reported in this thesis has been
carried out under the auspices of SIKS, the
Dutch Research School for Information and
Knowledge Systems.

UNIVERSITY
OF TWENTE. | **DIGITAL SOCIETY
INSTITUTE**

DSI Ph.D. Thesis Series No. 22-001

ISSN: 2589-7721

Digital Society Institute

P.O. Box 217, 7500 AE Enschede, NL

ISBN: 978-90-365-5310-0

DOI: 10.3990/1.9789036553100

Typeset with \LaTeX

Printed by Ipskamp

Coverdesign by Meteoriet Design

Copyright © 2022 Judith van Stegeren, The Netherlands.

Graduation committee:

Chair / secretary	prof. dr. J.N. Kok
Supervisor	prof. dr. D.K.J. Heylen
Co-supervisor	dr. M. Theune

Committee members:

Prof. dr. F.M.G. de Jong	University of Twente, Enschede, NL
Prof. dr. G.J. Westerhof	University of Twente, Enschede, NL
Prof. dr. ir. P. Spronck	Tilburg University, Tilburg, NL
Prof. dr. E. Reiter	University of Aberdeen, Aberdeen, UK
Dr. M. Cook	Queen Mary University of London, London, UK

Paranymphs:

Dr. L. Gatti
Dr. J.B. van Waterschoot

Contents

1	Introduction	1
1.1	Narrative and text in video games	2
1.2	Flavor text	5
1.3	Research context	7
1.4	Key issues	9
1.5	Aim, scope and research questions	12
1.6	Organisation	14
I	Background	17
2	Procedural content generation	19
2.1	Definition and applications	19
2.2	Key concepts	21
2.3	Approaches	23
2.4	Open problems and debates	26
2.5	PCG for textual assets	28
2.6	Games and coherence	36
2.7	Conclusion	40
3	Natural language generation	43
3.1	Definition and applications	43
3.2	Approaches	44
3.3	Popular NLG methods for PCG	47
3.4	Story generation	52
3.5	Evaluation	56
3.6	Conclusion	60
II	Flavor text generation	61
4	Low-cost story generation in the wild	63
4.1	Introduction	64

4.2	Related work	65
4.3	Data	66
4.4	Text generation methods	68
4.5	Methods for narrative coherence	71
4.6	Discussion	75
4.7	Conclusion	79
5	Generating flavor text headlines	81
5.1	Introduction	82
5.2	Related work	83
5.3	Description of Churnalist	86
5.4	Text generation with Churnalist	96
5.5	Evaluation	98
5.6	Conclusion	112
6	Rebuilding Churnalist with GPT-2	115
6.1	Introduction	115
6.2	Method	116
6.3	Error analysis	120
6.4	Discussion	123
6.5	Conclusion	126
7	NPC dialogue generation from annotated video game quests	129
7.1	Introduction	129
7.2	Related work	132
7.3	Method	134
7.4	Evaluation	139
7.5	Discussion	147
7.6	Conclusion	149
8	Collecting video game corpora	151
8.1	Introduction	152
8.2	Related work	152
8.3	Quality of video game corpora	155
8.4	Obtaining new video game corpora	158
8.5	Datasets	160
8.6	Discussion	174
8.7	Conclusion	178
III	Conclusion	179
9	Conclusion	181
9.1	Summary	181

9.2 Revisiting the research questions	183
9.3 Directions for future work	189
Bibliography	193
Summary	223
Samenvatting	227
Acknowledgements	231
About the author	235
SIKS Dissertation Series	239

1

Introduction

In 50 years time, video games have evolved from a niche entertainment medium to something that is present in almost every part of society. Video games are now part of our culture, just like other ubiquitous media such as books, music and films [193]. In 2020, they are not only used for entertainment, but also for applications such as education, sports, research and therapy.

Even though the video games industry is maturing, developing video games remains expensive, complex and challenging. In the book *Blood, Sweat and Pixels* [235], journalist Jason Schreier describes the game development process for ten well-known games, including *Stardew Valley* [18], *Diablo III* [38] and *Pillars of Eternity* [197]. Schreier interviewed hundreds of game developers to find out why developing games is still so complex, even though the games industry has been around since the 1970s. Schreier suggests that games are *intrinsically* complex to develop. Video games are interactive and non-linear; the tools and technology required to develop them is constantly changing; and they revolve around the elusive concept of ‘fun’, which is hard to define, hard to measure and also hard to plan and predict. Amato [5] described additional challenges in the game development industry: game development teams typically work within strict technical boundaries, while also juggling scheduling and budget constraints, and requirements from publishers and distribution platforms. The cost of creating games also keeps increasing because players expect increasingly more playable content per game. Simultaneously, there is a growing demand for games that incorporate forms of adaptivity and personalization.

Procedural content generation (PCG), i.e. creating new game content automatically with software, is a technique that can be used to solve these problems [5, 37, 240]. Procedural generators can be used for scaling up the creation of game assets, i.e. all files, data and code that form the building blocks of a video game, and personalizing game content. In this thesis, we investigate procedural content generation as a way to scale up the creation of textual video game assets, such as character dialogue, stories, item descriptions, and other in-game text. We

are specifically interested in *flavor text*: textual game assets that convey so-called *flavor* (or backstory) to the player. In the coming sections we will describe these concepts in more detail.

In Section 1.1, we start with a short description of the role of narrative and text in video games. In Section 1.2 we will zoom in on flavor text, which is a specific type of video game text. We will then introduce the two most relevant research fields for the work in this thesis, namely procedural content generation and natural language generation, in Section 1.3. Section 1.4 describes a few important issues that we should take into account when researching flavor text generation for games. In Section 1.5 we will describe the research goal and research questions that underlie the work in this thesis. We end this chapter with a short overview of the rest of this thesis.

1.1 Narrative and text in video games

In the early 1980s, video games had simple graphics, such as *Pacman* and *Tetris*, or were fully text-based, such as the adventure game *Zork* [138]. Recent video games, such as *Red Dead Redemption 2* [219] and *The Witcher 3* [60], have graphics that are almost photo-realistic in quality. Although video game graphics have evolved considerably since text-only adventure games, written text still plays an important role in video games, as it is a straightforward way of communicating the game's narrative to the player.

According to Dansky [79, p. 5], the purpose of game narrative is to provide a framework or a justification for the gameplay encounters happening in the game. It achieves these goals through three techniques: immersion, reward and identification.

Immersion is the state of mind where a person is completely absorbed in what they are doing. In the context of games, immersion is the feeling that the real world temporarily has ceased to exist, and the game world is the player's reality. Game narrative provides context for game events, and a sufficiently believable context provides immersion. Narrative can also function as a reward for the player's effort and progress, and can motivate the player to keep moving forward throughout the game. Finally, narrative can also help with identification. Well-written narrative helps the player to identify and distinguish the elements of the game (world), e.g. dangerous monsters, friendly allies, and useful items. This information contextualizes the events in the game and provides justification for them. It also establishes the player's place in the game world, and helps the player make decisions.

In handbooks for game writers [23, 84, 83], we can find examples of the various types of text that game writers have at their disposal to communicate game narrative via text: linear narratives, branching narratives, dialogue, help text, scripts for cut scenes and cinematics, voice actor directions and strategy guides to accompany the game. Which of these types of text are present in

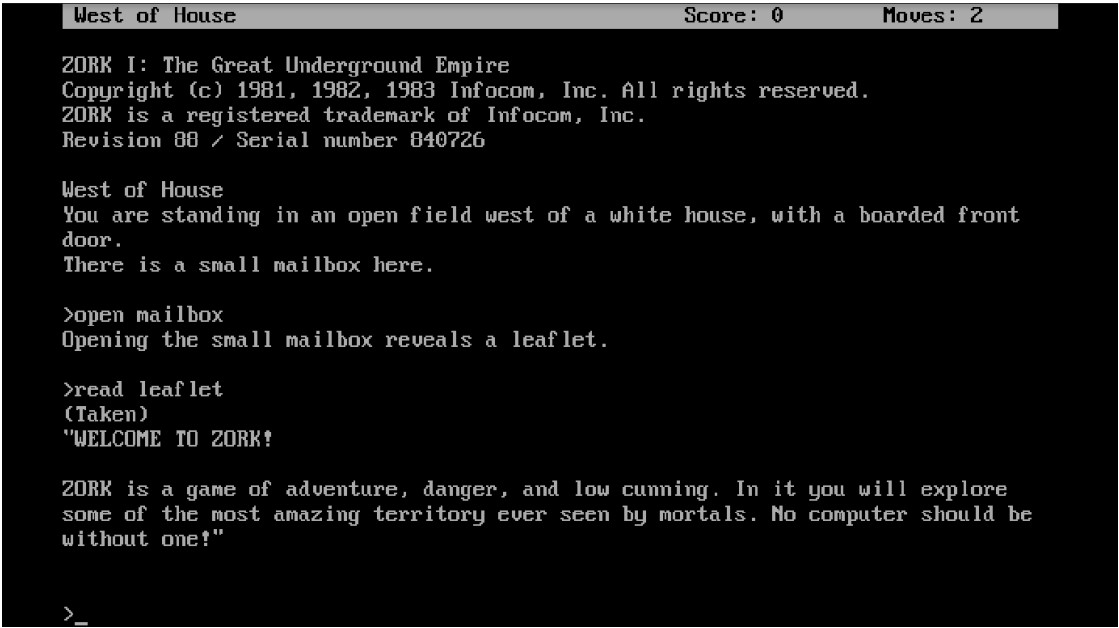


Figure 1.1: Screenshot from *Zork: The Great Underground Empire* [138], which shows the opening lines of the game, and a few player actions. In *Zork*, the player interacts with the game by typing natural language text, such as ‘enter house’ and ‘use lamp’. In this screenshot, the player (character) is reading a leaflet. The leaflet can be found in the mailbox of the white house, which is the starting location of the game.

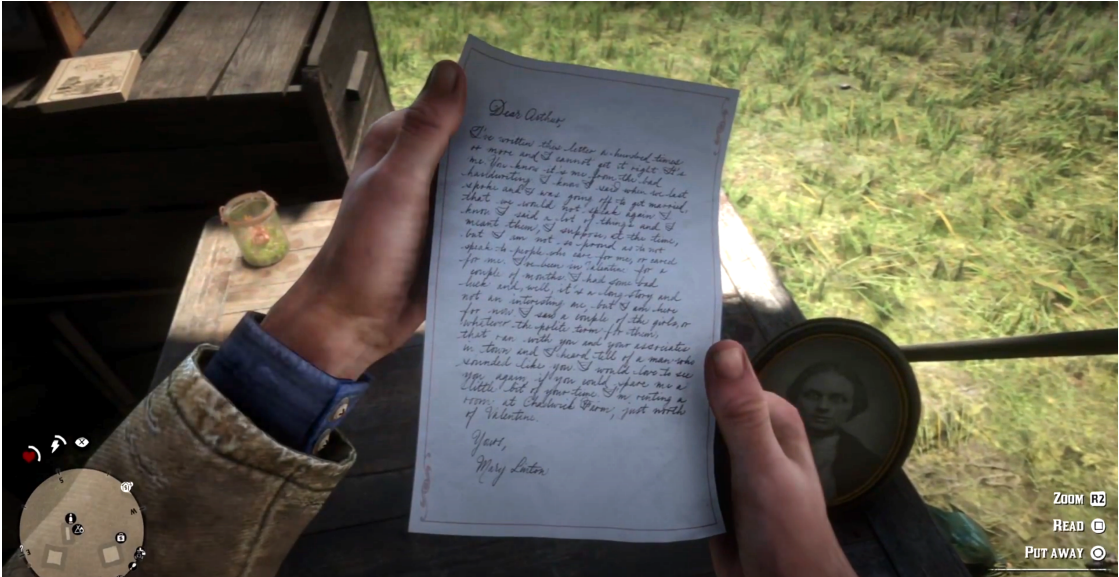


Figure 1.2: Screenshot from *Red Dead Redemption 2* [219]. Main character Arthur Morgan reads a letter. The letter points towards an in-game location. If the player subsequently visits this location, they can start a new mission (a quest, or a part of the game’s storyline).



Figure 1.3: Screenshot from the introduction animation of platform action game *Super Metroid* [196].

a given game depends on various aspects of the game, such as the genre of the game (action game, platform game, roleplaying game, first-person shooter, puzzle game), the genre of the game’s narrative (horror, fantasy, science fiction, romance), the setting and gameplay mechanics.

Some game genres lean more heavily on narrative than others. A platform game such as *Super Metroid* [196] has hardly any in-game text. The game opens with a short animation in which the main character Samus talks about her past, see Figure 1.3. Apart from short intro and outro animations, the game features no explicit textual narrative.

In contrast, role-playing game *Morrowind* [28] depends almost fully on text, see Figure 1.4 and 1.5. The screenshots demonstrate the two most important types of in-game text. In *Morrowind*, important in-game events are logged in the player character’s journal (Figure 1.4), and most quests can only be solved by searching for clues hidden in non-player character dialogue (Figure 1.5).

Fighting games and multiplayer strategy games are often light on narrative, whereas adventure games and (computer) role playing games often depend almost entirely on narrative: the play experience through the game corresponds precisely to the character growth through the course of the narrative [79].

The research in this dissertation focuses on game narrative in role-playing video games: video games where the player takes on the role of a virtual character. Roleplaying games (RPGs) are often narrative-driven [190] and allow for many different interesting types of narrative content. A survey by Ip [139] shows that RPGs are particularly narrative-heavy compared to other video game genres. This means, as we will see in later chapters, it is a perfect application domain for text generation research.

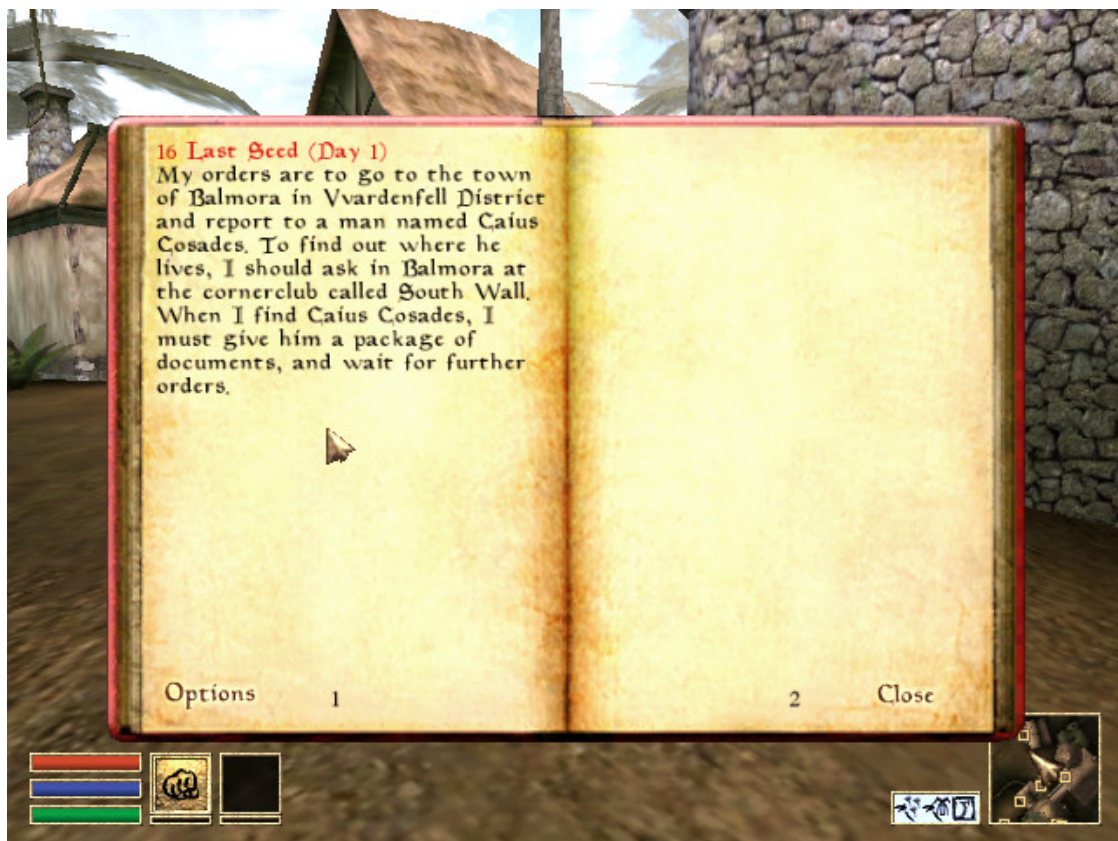


Figure 1.4: Screenshot from Morrowind [28]. All important game events are recorded in the player character's journal, which acts as a narrative logbook for the entire game.

1.2 Flavor text

Backstory, or *flavor*, is one of the techniques that game writers use to draw the player into the game world. Consequently, flavor text is text that helps a player with experiencing immersion.

The term *flavor text* can mean different things. Many researchers and practitioners use the term in their writing, but very few people actually define what they mean by it. Academic texts on procedural content generation for video games often distinguish between functional and decorative game assets [271, 127]. A game asset is decorative if it can be removed from the game without breaking the game. In research, the term *flavor text* is used interchangeably with decorative textual game assets.

However, text books on game writing from the video games industry stress the importance of embedding flavor in every piece of text [23, 84], which would suggest it is hard to make this strict distinction between functional and decorative text in practice, and that every functional in-game text should be considered flavor text as well. Furthermore, although flavor text is often decorative, it is not necessarily unimportant. This goes especially for role-playing games, where immersion is a key aspect of the gameplay. Since flavor text helps to create



Figure 1.5: Screenshot from Morrowind [28]. Most of the game narrative is communicated via text boxes with NPC dialogue, such as here, where the player character is having a conversation with an Imperial soldier named Sellus Gravius.

this immersion, it can be considered an important ingredient of a role-playing game. In other words, it is hard to define flavor text based on its (lack of) functionality. For the remainder of this dissertation, we define flavor text as any text that communicates game narrative to the player.

This dissertation focuses on generating flavor text for role-playing games. Flavor text is especially suitable for experimenting with new natural language generation, as its primary function is to immerse the player, not to inform. Consequently, the truthfulness of the content is less important than the affect it evokes in the player and whether the style of the text is coherent with the game. Changing flavor text will typically not break a game, which is also an added benefit.

Flavor text is originally a term from the world of board games and tabletop games. War games, table top games where players reenact battles using miniatures, often come with a booklet with information about famous battles and their historical context. When people started to play fictional war games, for example in fantasy and science fiction settings (such as Warhammer), these booklets turned into elaborate descriptions of the game setting and backstories for the corresponding fictional armies [285]. Similarly, children's toys and collectables from the 1980s, such as Transformers and G.I. Joe, featured elaborate descrip-



Figure 1.6: Magic: The Gathering card ‘Pacifism’, from the expansion set ‘Ikoria: Lair of Behemoths’ (2020). Art by Kev Walker. The italicized text at the bottom of the card is flavor text. In Wizards of the Coast’s official description of Magic: The Gathering [286], we can read that flavor text has no effect on gameplay; it’s a bit of story information about the card.

tions and backstories of characters on their packaging.

Probably the first game to use in-game flavor text is trading card game *Magic: The Gathering* [100]. Figure 1.6 shows a Magic: The Gathering card, with italicized flavor text at the bottom of the card. According to Dave Howell [135], who was production manager of Magic: The Gathering in 1993 [192], the designers had some space left on the game cards, and the team decided to fill this space with a quote that provides context or backstory for the entity that is represented by the card. Initially called ‘color text’, referring to the ‘color commentary’ by sports commentators, the italicized quotes were sourced from the bible or literature and meant to add variety and interest to the cards. Flavor text for Magic: The Gathering evolved to an important way of communicating the game lore to the players, extending other methods of conveying the game’s narrative such as marketing materials, images, websites, strategy books, etc. [77]

1.3 Research context

There are two research fields that are especially relevant for flavor text generation: natural language generation (NLG) and procedural content generation (PCG). I will introduce each field in the sections below. In Chapter 2 and 3,

developments in these research areas that are relevant to this thesis will be discussed in more detail.

1.3.1 Natural language generation

Natural language generation is the research field that focuses on the automatic creation of text from data. The input for natural language generation systems can consist of text (text-to-text generation), images, facial expressions, human gestures (image-to-text generation), or other types of data (data-to-text). NLG systems can write many different types of text, that span the breadth of human writing: from news articles, sports game reports, exams, questionnaires and financial reports to poems, letters, jokes, movie scripts and entire novels. Another prevalent example of text generation technology in day-to-day life is machine translation, such as GoogleTranslate, which uses text-to-text generation.

The pipeline architecture as described by Reiter and Dale [213] has been the leading architecture for natural language generation systems for a long time. This architecture divides the generation process in a number of distinct steps. An alternative to the pipeline architecture is end-to-end generation, a method in which a singular, integrated system takes care of all generation steps in one go. In the past three years, end-to-end generation based on Transformers [275, 85, 209], a type of neural architecture, has increased in popularity because of its good performance for a wide variety of NLP and NLG tasks.

The ultimate goal in NLG is, of course, to build systems that can write texts that are on par with human writing. Given this goal, NLG systems are typically evaluated on output text quality, using properties such as fluency and grammaticality. Depending on the aim of the specific system, they can also be evaluated on properties like factualness, common sense, internal logic, coherence, linguistic style, and affect. Evaluating NLG systems is a research field in its own right, as there are many ways to do this. We can conduct automatic evaluations with statistical metrics, or manual evaluations with human judges. NLG systems can be evaluated in isolation, in an artificial context, or in a real-world context.

1.3.2 Procedural content generation

Procedural content generation is the research area that deals with the automatic creation of video game content or video game assets, such as levels [260], textures [93], music [205], quests [199], puzzles [80], dialogues [164] and narratives [153]. Instead of generating content of a particular type, some researchers [97, 72, 117] investigate the automatic creation of entire video games. In-game texts are also a type of game content, but compared to game assets such as video game levels and textures, there is little academic research in that area. However, recently game developers have been increasingly experimenting with procedural narrative [241].

When it comes to evaluating procedural content generation systems, the evaluation goals are similar to those of NLG systems, but not the same. Both fields evaluate generated artifacts on two types of properties: form and function. NLG systems are typically evaluated on textual properties such as grammaticality, clarity and coherence, and properties that fit the intended application of the text generation. In procedural content generation for games, the goal of generating artifacts is already given: inclusion in a video game, possibly after manual post-processing (refining, filtering, rewriting) the output. Consequently, there is an explicit focus on video game related aspects such as fun and gameplay in PCG evaluation.

Creativity, especially how we can define and measure it, also plays an important role in PCG research. Since generated game assets are often placed in the game together with human-created content, the PCG content should not be easily discernible from manually created content. Because of this, procedural content generation research also builds on the field of affective computing, which deals with computational models of emotion, and computational creativity [69, 19], which researches how we can create creative artifacts automatically, such as dance, theater, music, paintings, and poems.

1.4 Key issues

There are three key issues that should be addressed if we want to investigate text generation for video games. Firstly, there are multiple research gaps that need to be addressed to increase the maturity of flavor text generation and text generation for games in general. Secondly, we need to investigate text generation while taking game-level coherence into account, something that is still largely unexplored in existing research. Finally, we need to address practical problems with NLG systems, which stand in the way of adoption of NLG by the game development industry. We will discuss these issues in the following three sections.

1.4.1 Research gaps

Although PCG is maturing as a field, there is very little research on generating textual game assets. Existing studies generate either entire games or specific game assets. If the generated assets give rise to in-game text, such as quests or characters, this text is typically produced using rule-based NLG methods, such as templates or grammars, e.g. [179, 229, 119, 117]. Rule-based NLG methods have the advantage that they are very predictable and robust, which are important properties for game designers. However, this means that most PCG research that touches on textual assets does not reflect the enormous developments of natural language generation in the past few years, that are driven mostly by statistical and machine learning methods.

The same applies to video games as application for natural language generation research. Although Gatt and Krahmer [101, p. 137] identify games as a promising application domain for natural language generation, video games are not yet considered seriously as application domain for natural language generation research. If we search the anthology of the Association for Computational Linguistics¹, the most important repository for computational linguistics research papers, for papers mentioning ‘video games’, we find very few relevant research papers. This can be explained by the dominant theme in NLG research: natural language generation systems discussed in academic literature tend to generate non-fiction text, for information-driven applications such as expert systems and report generation systems. Although there is a research community that focuses on generating fiction, such as stories, poems and jokes, this is a relatively a small community within the computational linguistics field.

1.4.2 Coherence

Coherence, or lack of coherence, is a property of text. However, the word ‘coherence’ has not been used consistently in research literature. Spencer and Fitzgerald [247] reviewed the literature for coherence and already found 27 different definitions of coherence. They observed that the term coherence was often not defined at all, or used similarly as cohesion. According to Morris and Hirst [189], coherence is a term for making sense; a text is coherent if it makes sense. They contrast coherence with cohesion, which refers to ‘sticking together’. If a text is cohesive, this means that ‘the text all hangs together’. In this thesis, we follow the definitions of Karmiloff-Smith [144], which were also used by Shapiro and Hudson [236] in their work on narrative analysis. This seems the most relevant definition for the work in this thesis, given that we are also talking about narratives, albeit for video games. Karmiloff-Smith defines coherence as the temporal/causal structure of story content, and cohesion as the linguistic devices (anaphoric, connective, etc.) that tie a span of sentences together [144, p. 62]. Narratives can be coherent without being cohesive, and vice versa. Shapiro and Hudson discuss both concepts in the context of narratives produced by children. To create a coherent story, all parts of the story must be structured so that the entire sequence of events is interrelated in a meaningful way. A cohesive story is created by using linguistic reference devices, such as connectives, that tie a span of sentences together to form a whole [236, p. 960].

A text can be coherent at different text levels: at the sentence level, at the paragraph level, or at the level of the entire text. The latter is called global coherence. We can also analyse whether multiple pieces of text form a coherent whole. If we are looking at coherence between multiple pieces of text, we call it intertextual coherence.

Intertextual coherence is especially relevant for narrative-heavy video games,

¹ACL Anthology, <https://aclanthology.org/>

where the player typically gets to know the game narrative and game setting via a collection of short texts (dialogues, item descriptions, cut scenes). The player does not read and interpret this flavor text in isolation, but in the context of the larger game narrative. Consequently, when we generate video game text we need to consider another level of coherence: apart from local coherence (sentence level, paragraph level) and global coherence, flavor text should be coherent at the game level, which we define as follows: a piece of flavor text is coherent at the game level if a reader perceives a plausible connection between the game (game setting, game world, game genre, game narrative) and the text.

1.4.3 Practical problems

Amato [5] and Short and Adams [240] name various practical problems that stand in the way of large-scale adoption of procedural content generation by the games industry. These problems also hold for text generation in particular. They can be broadly divided into three categories: risk, cost, and the gap between research and practice.

Risk

When using procedural content generation, the consistency and quality of generated assets is not guaranteed. Using live procedural content generation, where generated content is directly used in a video game, might therefore be too risky for publishers, compared to working with human designers [240]. This can be mitigated by only using generators to pre-generate content, and letting human editors check, filter, and polish generated content before including it in a game. Similarly, high-budget games might aim for an explicitly ‘authored experience’ [37] [240, p. 7], i.e. a well-defined experience so that a player will experience a game exactly like the authors intended. This is at odds with the variation and flexibility inherent in procedural content generation. The inverse is also true; if the procedural content generation is rule-based, and the set of rules relatively small, the procedurally generated content might be too predictable, bland or boring for inclusion in a game! Because of the unpredictability and uncertainty of procedural content generation, publishers and developers will often opt for the more expensive but controllable approach: working with human-created content instead of incorporating procedural content.

Cost

Designing a procedural generator takes time and expertise. Depending on the chosen method for generation, a developer needs to create rules, templates or grammars, perform data-processing on text corpora, or train one of more machine learning models. Even with sophisticated procedural generators, the quality of the output cannot be guaranteed. According to Amato [5], this keeps many

producers of video games from using PCG. An often-heard argument for using procedural content generation is that it saves time. Once an organisation has invested in developing a procedural generator, it can create large amounts of game content for free. However, in the case of text generation for games, we might just be transforming the task of writing video game text to the task of building a text generator. Since text generation experts are currently more rare than game writers, using procedural text instead of human-written text seems a costly choice for game developers.

Gap between research and industry

Academic research tends to build on work from academia. In academic literature, new techniques are mostly applied on toy examples, which might not be representative for industry practices. On the other hand, the game industry typically works with industry tools. If there is innovation, it comes from within these companies, and tools might be kept in-company. New innovations might be kept secret as intellectual property to keep an edge over competitors, and sharing or publishing about new technology might not have high priority within companies. Technology and code reuse between games from one development company is not standard practice, let alone between games from different companies. On the contrary: Schreier [235] describes that it is common practice to start from scratch when a game company starts developing a new game. Even when game industry innovations are shared within the game development community, many of these innovations are published outside of the scope of academia. New technologies from industry might not be known to the research community, or they are not investigated further because the techniques are not easily generalizable across games. This divide between research and industry has led to a knowledge gap in procedural content generation.

By choosing particular approaches for procedural content generation, these problems can be mitigated. The knowledge gap can never be solved completely, but by improving controllability and researching PCG techniques that are not so costly, we can hopefully make procedural content generation for video game text more accessible to the game industry. By working with realistic resources as much as possible, e.g. industry-grade programming libraries and text from real-world commercial games, we hope our findings translate better to games industry practice than research that builds solely on academic resources.

1.5 Aim, scope and research questions

In Section 1.4 we described the key issues with regard to flavor text generation for games: research gaps in NLG and PCG research, the lack of attention for game-level coherence, and some practical problems that stand in the way of wide-spread adoption of PCG in the game development industry. Given these

challenges, the main goal of the research in this dissertation is:

Research goal. *To investigate and develop methods for generating coherent flavor text for video games, while keeping in mind the risks and costs.*

Since ‘video games’ is such a broad domain, we will focus on a specific subset of games: role-playing video games. We will focus on *role-playing games*, because flavor text is particularly useful for this type of game. The role of flavor text is to help with immersion, and immersion is essential for role-playing games. We will use role-playing games as a literary source for studying existing flavor text, and as the application domain for our text generation outputs.

Most text generation research uses datasets that were not taken from commercial video games, but from ‘research games’ or ‘toy examples’ written by researchers or research participants. In order to mitigate the gap between industry and research, we will use only *commercial* video games as use cases and data sources. By using real-world data, we will hopefully improve the applicability of our findings to industry.

The research goal is implemented via a set of research questions. We will now list the research questions and describe the rationale behind each question.

As stated in Section 1.3, the existing work on text generation for games is very limited. However, flavor text shares some characteristics with other types of fiction. We start by investigating the existing methods for generating coherent works of fiction. We focus specifically on methods that have been applied in the wild, that might have been described outside of the scope of existing research literature, in order to keep the methods usable in practice.

Research question 1. *Which methods have been used in the wild for generating coherent works of fiction?*

Research question 1 will be addressed in Chapter 4. In this chapter, we discuss four methods for creating coherent long-form stories. One of these methods, ‘evoking coherence’, focuses on creating text that is *perceived* as coherent by readers. In the follow-up chapters, we assess whether we can transfer this method from generating works of fiction to generating coherent flavor text.

Research question 2. *How can the ‘evoking coherence’ method from Chapter 4 be adapted for generating coherent flavor text for video games?*

This research question will be addressed in Chapter 5 and Chapter 6. The pipeline architecture of Reiter and Dale has been a popular approach for NLG systems for a long time. However, in 2018 researchers developed a Transformer-based architecture for language models. This new architecture was popularized by various open source libraries for NLP and NLG, which led to a wide-spread adoption and fast increase in the number of online NLG systems, especially for creative text generation. Transformers are able to produce highly coherent but often non-factual outputs. We will investigate whether Transformers can be used for flavor text generation:

Research question 3. *How can we use Transformer-based language models to generate flavor text for video games?*

This research question will be addressed in Chapter 6 and Chapter 7.

If we can use Transformers for flavor text generation, how does this approach compare to a modular approach in terms of game development risk and cost? We will compare the modular approach used in Chapter 5 with the Transformer-based approach from Chapter 6 in the context of generating flavor text.

Research question 4. *What are the advantages and disadvantages of rule-based and Transformer-based approaches for generating coherent flavor text?*

This question is explored in Chapter 5 and Chapter 6.

If we want to use data-driven NLG methods, such as language generation using Transformers, we need training data from the video games domain. However, there is a scarcity of text data from video games, which constitutes a practical challenge that stands in the way of using NLG for generating coherent flavor texts.

Research question 5. *If we want to generate coherent flavor text using data-driven NLG methods, what are the requirements for training data, and where can we find data that fulfill these requirements?*

This question is answered in Chapter 8.

1.6 Organisation

The remainder of this dissertation is organized as follows. The first part of this dissertation contains two chapters that describe the two relevant research fields of the work in this dissertation: procedural content generation (Chapter 2) and natural language generation (Chapter 3).

The second part consists of chapters in which we address the research questions from Section 1.5. In Chapter 4 we explore contributions to NaNoGenMo to find approaches for story generation in the wild. We then discuss how these approaches could be applied to procedural text generation for video games. In Chapter 5 we continue with the ‘evoking coherence’ approach from Chapter 4, and incorporate this in a rule-based headline flavor text generator called Churnalist. The text generator is then evaluated on language quality and functionality. In Chapter 6 we describe a new version of Churnalist, which uses a Transformer-based language model. We explain the training method and compare the language quality of the two versions of Churnalist. Although the generator is meant for creating flavor text for games, both versions are based on headlines data, i.e. non-video-game text. In Chapter 7 we apply the language model training method to text from a video game. The language model is then used to generate NPC dialogue, which functions as flavor text for role-playing

game quests. The language models from Chapter 6 and 7 require training data for fine-tuning the model for a specific application. However, currently there are not many video game corpora available. In Chapter 8 we describe potential places where video game corpora can be found. We also discuss what constitutes a high-quality video game corpus, and present three corpora from three commercial role-playing games that can be used for future research.

In the final part of this thesis (Chapter 9), we provide a summary of the various chapters. We then revisit the research questions and try to answer them based on the findings of part 2. We reflect on our findings, and wrap up with some ideas for future work.

Part I

Background

Procedural content generation

This chapter describes important concepts and relevant work in procedural content generation (PCG). We start by defining procedural content generation, and describing the various applications of PCG in game development. We then give an overview of the most common approaches: constructive methods, search-based methods and methods based on logic solvers; machine learning methods and methods that build on reinforcement learning. We also discuss two popular subtopics in PCG: generating assets that incorporate affect, emotions and sentiment, and using real-world data for PCG. Finally, we discuss a series of challenges and open problems within the PCG field. This ends the general part of this chapter.

Starting from Section 2.5 we focus on procedural content generation for textual assets, such as NPC dialogue, game narratives and flavor text. We describe the most important work on PCG for text: authoring tools, generating stories and quests, qualitative procedural content generation, dialogue and flavor text. In Section 2.6 we discuss coherence in the context of game assets, and describe various approaches for generating coherent procedural content from the literature.

2.1 Definition and applications

Procedural content generation (PCG) is the automatic creation of game content through algorithmic means [288]. It is used in game development practice to increase replay value, to reduce production cost and effort, to save storage space, and as an aesthetic in itself [258].

A wide variety of game assets can already be produced automatically with procedural content generation. Both in research and game development practice, PCG has been used to create new game assets for video games, including textures [93], music [205], maps [269], levels [260], narratives [153], dialogue [164],

quests [199]. Instead of investigating PCG for specific types of game assets, some researchers [270, 74, 117, 73] have investigated procedural generation of entire games.

PCG can be used to quickly generate large volumes of game content. A generator can function autonomously, or as part of an interactive process of AI-assisted design, where a human designer and an algorithm work together to create game content. According to Yannakakis and Togelius [288], content creation in video game production is still largely manual, and this has grown into a bottleneck for the game development process. Procedural content generation is often described as a labor-saving technology, which game developers can use to scale up their production volume, and create more game assets with fewer people.

One of the earliest examples of procedural content generation used in video games is the dungeon crawling game *Rogue* [1]. *Rogue* used procedural content generation to generate infinitely many new dungeons for the player to explore, thus increasing the replayability of the game [5, 240]. Procedurally generating maps and levels is now common practice in many commercial role-playing games, such as *Diablo* [40], *Torchlight II* [222], *Rogue Legacy* [61], and *Pillars of Eternity* [197]. Indeed, Johnson [142] observed that PCG research has largely focused on game assets that make up virtual spaces, such as maps, terrains, cities, buildings and trees.

In the 1980s, procedural content generation was also used to compress game assets, so that large amounts of game content could be stored on very small storage media. *Elite* [45], a space exploration game, used procedural content generation for compression. It managed to store hundreds of star systems in a few kilobytes of memory by representing each planet as just a few numbers. A deterministic procedural content generator could then unpack these numbers to complete star systems during gameplay [288]. At the moment, the storage size constraint for video games, and consequently the applicability of PCG for compression, is making a reappearance in games for mobile devices such as phones and tablets [5].

In their book on procedural content generation for games, Short and Adams [240] list some of the advantages of PCG.

When done right, procedural content generation can be used to make game development more efficient. As procedural generators are often modular programs that create one type of asset, PCG can increase the possibility of code re-use between games.

Procedural content generation can also be used for quality assurance. We can formalize the meaning of a ‘correct’ or ‘high-quality’ game asset, and use a computer to automatically check the quality of human-created and auto-generated game assets. A computer is much better at following formal quality constraints than a human, and is less prone to accidental mistakes, so combining a formal definition of quality with a procedural generator can lead to higher quality game assets. Note that we can only take this approach if the quality requirements can

be formalized and represented in a format that the generator understands. For example, we could formalize a valid dungeon room in an exploration game as ‘it has at least one entry and one exit, and contains a monster and a piece of treasure’. However, a requirement such as ‘a dungeon room should be fun to explore’ would be very hard to formalize and consequentially less suitable for automatic quality assurance or generation.

Content generators can also produce large volumes of game content with much more detail and variation than a team of human designers could hope to create in the same amount of time. Consequentially, PCG has allowed small game development teams and independent game developers to create large games, something that would not have been possible if all assets had been created manually [37]. Additionally, PCG can use simulations to add detail and realism to a greater extent and scale than any human designer could do manually [240]. For example, software SpeedTree, a procedural generator based on L-systems, is used in the games industry to generate thousands of variations of realistic-looking 3D trees.

Apart from solving very practical problems in the game industry, procedural content generation has additional benefits, both for video game developers and players. It can be used for personalization and adaptation. Lopes and Bidarra [175] discuss different forms of adaptivity for video games in their survey paper. Yannakakis and Togelius [288] proposed experience-driven PCG, i.e. PCG that takes player experience and affect into account during generation, can make it easier to tailor content to individual players or groups of players. PCG allows players to have a unique game experience; if the game generates assets on the fly, no two players will see the same content. This can even give rise to new modes of play, for example when players learn the inner workings of the generator, and start playing with that.

Besides game content creation, procedural content generation research is also starting to contribute to other tasks in game development. Machine learning allows us to build mathematical models of existing game content. These models can be combined with procedural content generators for a variety of purposes, such as game content repair [260, 141], game content recognition, game content critique and game content analysis, i.e. analyzing game content for properties such as relatedness, uniqueness, difficulty, and playability [258].

2.2 Key concepts

2.2.1 Generation

We can distinguish between *online* and *offline* procedural generation. Online generation means game content is generated on-the-fly, at runtime, whereas offline generation refers to generation during game development.

Procedural content generation algorithms can be categorised according to

the level of control that is available to the user of the generator. A generator that converts an input into a game asset by a set of random dice rolls offers relatively little control. On the other hand, some generators allow the designer to specify various parameters that are taken into account during generation. At the end of this side of the spectrum we find *co-creation* or *mixed-initiative* PCG tools. Liapis and Yannakakis [168] explored how mixed-initiative design can be applied for computational creativity systems.

Some generators always generate the same output given the same input, which is *deterministic* generation. Provided that the input is smaller than the generated output, deterministic generation can be seen as a form of (de)compression. When a generator is *non-deterministic*, a single input can lead to various outputs. *Stochastic*, random, probabilistic or aleatoric generation means that the output of the generator is non-deterministic and dependent on chance.

Outputs of a procedural content generator can be placed in an abstract mathematical space called the *generative space*. *Game content space* on the other hand is the mathematical space of all possible game content, regardless whether it was created manually or procedurally. Similarly, *game content generator space* is the space of all possible content generators. Although these latter concepts are very abstract, they are useful when we try to visualize or compare the generative capabilities of a specific generator or approach.

2.2.2 Game content

Togelius et al. [271] distinguish between *necessary* and *optional* game content. Necessary content is required by the players to progress in the game, whereas optional content is that which the player can safely ignore. The difference between necessary and optional content is important to take into account during generator design, as generation flaws in necessary content can lead to critical errors and break the game. On the other hand, optional content can safely be ignored by the player, so quality assurance is less important for this type of game assets.

In their survey on PCG with machine learning, Summerville et al. [258] distinguish *functional* game content and *cosmetic* game content. Functional game content is game content that is directly related to the game mechanics. The authors define functional content as artifacts that, if they were changed, could alter the in-game effects of a sequence of player actions. Cosmetic game content is anything that is not functional. The authors seem to have made this distinction mainly to filter out purely decorative game content such as images (sprites, textures) and sound assets. Their survey focuses on functional game content. The authors argue that cosmetic game content can be generated, consumed and evaluated in isolation, in contrast to functional game content. Additionally, there is already much research on generating purely decorative content outside of games. The authors note that there is a difference between their functional–cosmetic di-

chotomy and the necessary–optional dichotomy of Togelius et al. [271], as it is possible to have optional functional content and necessary cosmetic content. Depending on how flavor text is used in a game, and where it occurs, a piece of flavor text can be necessary or optional, functional or cosmetic.

2.3 Approaches

In this section, we broadly describe the most important approaches for procedural content generation. These approaches are generally independent of the type of game content, e.g. text, sound or images. We start by describing constructive, search-based and solver-based methods. Then, we describe machine learning and reinforcement learning approaches. Finally, we take a deeper look at two applications of PCG: generating assets that incorporate affect, and generating assets from real-world data.

2.3.1 Constructive, search-based and solver-based methods

Constructive methods create content in a data processing pipeline without evaluation. The generator simply constructs output from its input, and is finished. It does not check whether the output is valid according to some set of constraints. According to Summerville et al. [258], this is currently the prevalent approach to PCG in the games industry.

On the other hand, *search-based* methods search a generative space for outputs that fulfill a specification or maximize a fitness function. In their seminal paper on search-based PCG, Togelius et al. [271] describe the related generate-and-test approach for PCG. *Generate-and-test algorithms* have both a generator and a testing component. After generating a potential output, the generator evaluates the output with the testing component to check whether it follows the desired specification. If the output does not pass the evaluation, the content is discarded and the generator generates more outputs, until its output successfully passes the test criteria.

Solver-based methods use logic programming to find outputs that fulfill certain logical constraints. [243]

These three methods have as disadvantage that the building blocks of the generator, i.e. its algorithms, parameters, constraints, etc., are generally created manually, which is why they might not scale as well as methods that use machine learning. [258]

2.3.2 Machine learning

Procedural content generation has profited from the advances in machine learning. The recently renewed interest in neural networks has led to an increase in research that investigates machine learning techniques for generation, such as

Generative Adversarial Networks (GANs). The basic idea of generative machine learning is to build a model of (game) content from a large dataset of examples, and then use this model to produce new, similar artifacts. New content is generated by sampling the distribution of the model.

In their survey paper about PCG with machine learning, Summerville et al. [258] specifically define procedural content generation with machine learning (PCGML) as using machine learning to learn models of game content, based on a dataset of existing game content. Constructive, search-based and solver-based methods might also use machine learning internally, for example to evaluate their outputs. However, this differs from PCGML in that these approaches do not use models of existing game content.

Examples of machine learning methods for procedural content generation are: neural networks, such as long short-term memory networks, autoencoders, and deep convolutional networks; Markov models, such as n-grams and multi-dimensional Markov chains; clustering; and matrix factorization. Recently, Liu et al. [173] have explored deep learning methods for procedural content generation.

According to Khalifa et al. [147], one of the main reasons that PCG approaches based on search and optimization have not been applied widely in the game industry is that they take too much time. The authors propose procedural content generation with reinforcement learning (PCGRL), which draws on reinforcement learning for training gameplaying agents. It also builds on search-based procedural content generation in the search space of content generators, such as in the work of Kerssemakers et al. [146] on the intriguingly-named Procedural Procedural Level Generator Generator (PPLGG). An important difference between other PCG approaches and PCGRL is that the latter searches the space of *game content generators* (or ‘policies’) instead of the space of *game content*.

Search-based procedural content generation requires no training time, but needs large amounts of time for generation, i.e. searching game content space. For PCG with reinforcement learning, it is the other way around. The latter approach requires substantial training time to search the space of content generators, but once an optimal content generator has been found, generation is relatively fast. This means that PCG with reinforcement learning can be a viable method for online generation. Another advantage of this approach is that, compared to supervised machine learning (e.g. PCGML), no training data is necessary. We do not need a dataset with examples of valid game content, which is one of the main challenges related to PCGML according to Summerville et al. [258]. Finally, viewing policies as a multistep generation process is compatible with mixed-initiative procedural content generators. However, the PCGRL approach shares one challenge with search-based procedural content generation: both approaches require a (manually designed) method for representing and evaluating the generated game content.

2.3.3 Incorporating affect

In reply to the increasing demand for player modeling and personalized content in games, Yannakakis and Togelius [288] propose *experience-driven procedural content generation*. In this approach, which builds on generate-and-test PCG [271], the test-component of a generator uses multi-modal computational models of user experience to evaluate generated assets. The authors define user experience as “the synthesis of affective patterns elicited and cognitive processes generated during gameplay”, such as skills, preferences, and emotional profile. This was recently explored in the episode ‘Playtest’ of science fiction series *Black Mirror*. In this episode, a video game playtester experiences a personalized, procedurally generated virtual reality horror game, created in real-time from his worst fears.

Experience-driven procedural content generation builds on the affective loop [261, 133], a concept from human-computer interaction (HCI) research, i.e. a cycle of emotion elicitation, affective detection and modeling, and affect-driven system adaptation. The player’s affective state is modelled during gameplay, and the generator uses this data to generate and evaluate additional game content. Yannakakis and Togelius [288] describe experience-driven PCG as a form of on-line generation; however, it is plausible that the approach can be used for offline generation as well.

Procedural content generation that uses a feedback-loop for evaluating generated content is reminiscent of reinforcement learning, i.e. machine learning that uses a feedback-loop to incrementally improve the performance of the underlying model. Experience-driven PCG can be viewed as a form of reinforcement learning if the generator incrementally builds a player model over time, and uses this to generate increasingly ‘fitting’ content for a particular player. However, not all experience-driven PCG is reinforcement learning, as the former can also be used for sequential adaptation (instead of incremental adaptation), to monitor the player’s affective state throughout the process of playing the game.

2.3.4 PCG with real-world data

Some authors use procedural content generation to create new game assets from real-world data. *Data games* use freely available information, such as open data, to automatically generate game content. The term ‘data games’ was coined by Friberger and Togelius [96], who created a generator that transforms open data to monopoly boards. Using existing data as building blocks for procedural generators has as advantage that it reduces the manual effort that is still required by common PCG approaches.

We can find many examples of data games, both in academia and the video game development community. *Freeciv* [273], an opensource game inspired by

Sid Meier's Civilization series, uses geo-information from OpenStreetMap¹ to generate in-game maps and player positions. *A Rogue Dream* [74] uses Google query results to generate sprites and names for player abilities, enemies and items. Games from the Data Adventure series, i.e. *Data Adventure* [20], *WikiMystery* [21, 22] and *DATA Agent* [117] were created from semantically linked open data from online sources such as Wikipedia, Wikimedia Commons and OpenStreetMap.

2.4 Open problems and debates

Although procedural content generation can potentially solve various practical problems for game developers, Short and Adams [240] describe situations where using procedural content generation might be problematic. These situations are fertile ground for new procedural content generation research.

2.4.1 Controlability

Procedural content generators, especially the systems with non-deterministic components, can be hard to control for a user. If we want to understand the capabilities of a PCG system, we first need to understand the limits of its generative space. It is important that creators and users of procedural systems have a clear mental picture of what the generator can produce, as this can help with guiding a procedural system towards the desired output quality.

If a generator cannot guarantee certain properties in its output, such as fairness (important in competitive games) [240], difficulty level [5] or solvability [258], this can lead to critical problems for the underlying video game. Having a good consistency in generation quality is especially important for online generation, where generated assets are directly included in a game.

Controlability is also important for offline generation. Researchers have been investigating methods for improving the usability of procedural content generators, especially those with mixed-initiative and co-creation functionality. Tracery [70] translates the input grammar on the fly to generation examples, to visualize the impact of modifications of the grammar and provide rapid feedback to the user. Cook et al. [76] created Danesh, a plugin for game development tool Unity that allows procedural generators to be viewed, edited and analysed in a single unified interface. If a user can understand the influence of generator parameters on the generation process, it becomes easier to control the generator, and thus to create higher quality artifacts.

Summerville et al. [258] observed that this problem is even more important for PCG that uses black box machine learning models. How can these models communicate their generative space to a designer, and show how the model is

¹OpenStreetMap, <https://www.openstreetmap.org/>

affected by a change in parameters or training data? Techniques from Expressive Range Analysis (ERA) and Explainable AI for designers (XAID) [292, 76] can help with communicating model's generative space to designers.

2.4.2 Representation of game content

The generative space of a generator is highly dependent on its underlying representation of game content. The representation is again dependent on the chosen generative approach or algorithm, e.g. a machine learning based approach for text generation might require a different representation than a template-based approach. In 2011, Yannakakis and Togelius [288] mentioned that not all game content could be generated yet. Ten years later, this is still the case. This means that we need to design new representations for game content for which no representations exist yet. Additionally, we need to design new representations for game content that are compatible with novel approaches for PCG. An example of such a dataset is the Video Game Level Corpus (VGLC) by Summerville et al. [259], a corpus of video game levels with three types of annotations that can be used for procedural content generation with machine learning.

Many procedural content generators work with some variation of the generate-and-test strategy. The testing component of this approach requires an algorithm for the automatic evaluation of partially generated artifacts. This function is dependent on the definition of quality for the specific game content, which leads to an additional set of design challenges: How do we define quality for game content? How do we model these properties? Can we measure these qualities, given the game content representation? And finally, are the quality metrics compatible with the generative approach?

2.4.3 Cost

Because procedural generators can not be controlled perfectly, procedural content generation is inherently risky [240]. Even though PCG has seen considerable progress in 40 years time, and procedural techniques are used in many commercial games, there are still situations in which using PCG can have a negative impact on the game development process. Generators may not be consistent in the quality of their outputs, which can be fatal for games that incorporate online generation. Including flawed, harmful, offensive, copyright protected, or otherwise unsuitable game content can render a game unplayable or damage the developer's and publisher's reputation. For some publishers this is a reason to discard procedural content generation as an option altogether [37, 5]. Publishers try to mitigate this problem by doing more quality assurance on the generated content, but this requires manual work of human designers, which cancels out some of the benefits of using procedural content generation.

Even without added quality assurance, PCG can be costly; many approaches require the manual authoring of the generator itself, plus any required training

data, models, representations, generative algorithms and evaluation algorithms. In the case of offline generation and human-computer co-creation, human designers are necessary for the generation process, e.g. to curate and polish the resulting generator artifacts.

2.4.4 Challenges related to machine learning

In their survey about PCG with machine learning, Summerville et al. [258] give an overview of open problems and future research areas for PCG. Some of these apply specifically to PCG with machine learning.

As natural language generation systems increasingly build on machine learning, the following challenges also apply to text generation in general and text generation for video games.

- finding sufficient quantities of game data;
- learning from small datasets.
- dealing with the lack of publicly available datasets and standardized benchmarks in PCGML;
- investigating style transfer, i.e. the idea that a machine learning model can be transferred across domains, for game content models.

2.5 PCG for textual assets

Existing procedural content generation research focuses mostly on decorative game content generation and level generation. Examples of decorative content are textures, music and 3D models. Popular topics in level generation include generating platformer levels, dungeons for Zelda-like puzzle games, and maps for competitive real-time strategy games. There is however a small body of work that focuses on textual game assets such as stories, dialogues and flavor text, and game assets that give rise to textual game content, such as quests and socio-cultural concepts [142]. Procedural text generation often uses special authoring tools to help the user create the desired procedural text artifacts.

We will first discuss authoring tools for procedural text generation in Section 2.5.1, and then look at specific types of procedural text generation for games. In Section 2.5.2 we discuss stories and quests, in Section 2.5.3 we describe qualitative procedural generation, and in Section 2.5.4 we discuss dialogue generation for games.

In order to ground the generated textual assets in the game world, most generators that produce text (in the form of stories or dialogue), often also produce other game content that structures, contextualizes or otherwise connects the textual output to the rest of the game. We will look more closely at this contextualization in Section 2.6.

Procedural generation of textual game content also uses many techniques from natural language generation research that do not have video games as explicit envisioned application domain, e.g. creative text generation, story generation and dialogue generation. These will be discussed in more detail in the chapter on natural language generation, in Section 3.4.

2.5.1 Authoring tools

In many cases, designing a generator takes time, effort, and specialized expertise. In order to help designers build generators, and make procedural generation more accessible, various researchers [54, 70, 223] have investigated authoring tools, authoring aids and other systems that support designers in creating procedural content. Although the boundaries between authoring tools for NLG and PCG may be unclear at times, we will focus on those tools that explicitly name video games as the intended application domain.

In his 1995 paper on template-based text generation versus text generation with ‘real NLG’, Reiter already remarked on the necessity of authoring aids for template creation: “How should ‘templates’ used by an NLG system be authored; can we develop a nice authoring environment which enforces any necessary constraints in an intuitive manner?”

Caropreso et al. [54] created an NLG Template Authoring Environment for authoring text templates that requires no knowledge of programming or linguistics of the user. The goal of the system was to make creating narrative content, for example for games, more accessible and user-friendly. The natural language generation functionality of the system is based on SimpleNLG [103], a sentence realization system that creates natural language sentences from grammatical specifications in an abstract language. Users can create templates by writing example sentences, marking the variable parts of the sentence, and specifying links between dependent variables. The system helps the user by showing all possible sentences given this template, which the user can then use to further improve the template. This method for giving users feedback on the authored procedural rules by visualizing the generative space has since then been adopted by many other authoring tools.

Tracery [70] is an authoring tool for context-free grammars, with a focus on rapid user feedback and usability. It is described by Compton, Kybartas, and Mateas [70] as a ‘casual creator’ tool [71], i.e. “an interactive system that encourages the fast, confident, and pleasurable exploration of a possibility space, resulting in the creation or discovery of surprising new artifacts that bring feelings of pride, ownership, and creativity to the users that make them.” The focus on usability and transparency of the generative space has led to a wide-spread adoption of Tracery. The library has been ported to various programming languages, and has been used to procedurally generate text for games, interactive fiction and Twitter bots.

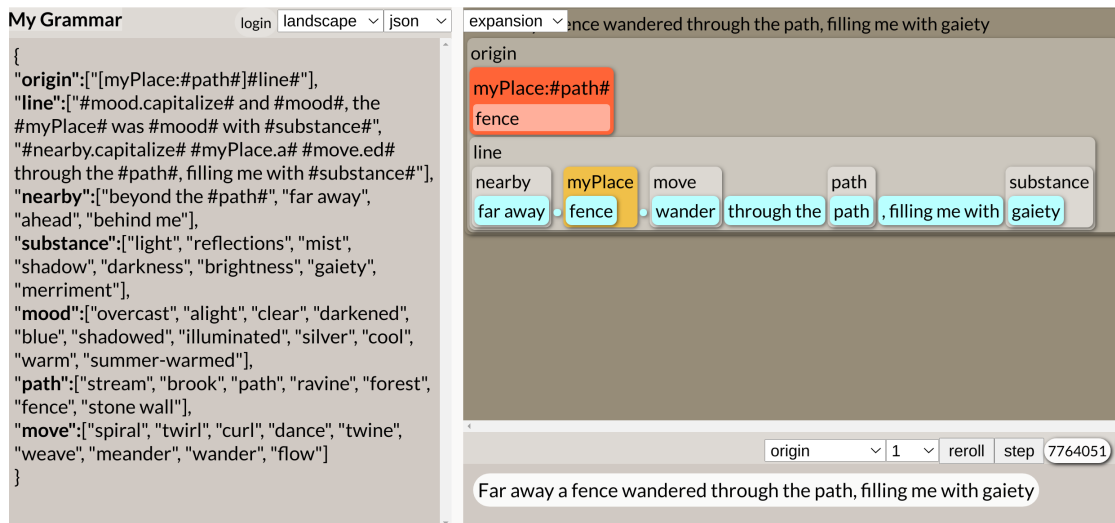


Figure 2.1: Tracery's interface with a grammar (left), visualization of the expansion steps (top-right) and the final expansion (bottom-right).

Expressionist [223] is an authoring tool intended for non-NLG practitioners, which builds on context-free grammars, meant for in-game (at runtime, or on-line) generation of text that depends on the current game state. It combines text generation with context-free grammars, with free-text markup in the form of arbitrary author-defined tags. The design of the tool builds on earlier work on combinatorial dialogue [229] which led to a set of open design challenges for authoring tools [226]. It was also partly inspired by authoring tool Tracery [70], which also uses context-free grammars with some additional annotations. However, the two systems differ in their goal: Tracery is meant for light-weight stand-alone text generation, whereas Expressionist is meant for content generation “that satisfies targeted requests made by an external application, like a game”. Although Tracery supports a few Tracery-specific, developer-authored annotations, such as those for pronouns and specific noun forms, Expressionist accepts arbitrary, author-defined tags for metadata. This metadata is accumulated during production using the Expressionist grammar. Additionally, Expressionist allows the user to assign probabilities to specific production rules, which makes Expressionist a method for using probabilistic context-free-grammars. In order to fully resolve Expressionist grammars and use the related metadata in a game for online generation, users need to implement their own expansion engine, called a Productionist.

Lessard et al. [164] evaluated Expressionist in practice by using it to generate dialogue for a prototype game. In the resulting research paper, they argue that we need to start looking at game dialogue authoring in a different way. At the moment the authoring tool does not make it more easy to generate text, but makes it easier to write or generate modular text. Grammars force us to break up the game dialogue in modular components, which are governed by a set of tags and the underlying generation logic. The authors argue that we need more

‘compositional representational strategies’ for text generation, i.e. ways to break up content in modular units which can be recombined. This is already common practice in music generation (e.g. with samples) and level generation (e.g. with game level tiles).

2.5.2 Stories and quests

We will discuss story generation as part of the natural language generation field in Section 3.4. However, some researchers specifically investigate narratives for inclusion in games. These narratives often have interactive aspects, such as in interactive fiction (IF) and quests, or have a connection with other game content, such as backstories for NPCs or game worlds. Interactive fiction is a text-based narrative-driven interactive game type, the digital equivalent of gamebooks from the 1980s, e.g. the Choose Your Own Adventure books [65]. In the context of PCG, some authors [179, 155] use the words ‘narrative’, ‘story’ and ‘quest’ interchangeably. Story generation is a broad field, which is reflected in the diversity of story generation research for video games: procedural text generators have been used to create content ranging from *fabulae* (abstract stories, or sequences of events) to complete stories, quests, linguistic variations in emergent narrative, and narratives that give rise to complete games.

Montfort [186] has created *Curveship*, an system that focuses on adding narrative variation to interactive fiction. Montfort remarks that IF system developers do typically not separate the telling (realised natural language) from what is told (the content of the story, the events, or the *fabula*). Separating these two components could give developers the possibility of introducing narrative variation by changing aspects of the telling, such as the chronological order of events and viewpoint of narration, without changing the underlying events that make up the narrative. *Curveship* implements this concept by dividing the emergent narrative module from typical IF system in two components: a Simulator and a Narrator. The underlying natural language generation is done with a three-stage NLG pipeline (see Section 3.2), which implements content selection and ordering, microplanning, and surface text realisation.

Interactive fiction, or interactive narratives, are a type of digital game that bridge traditional static, non-interactive texts on one side and fully fledged video games on the other side. *SCHEHERAZADE* is a interactive narrative generation system that automatically generates an interactive fiction by learning from crowd-sourced corpora of example stories. [166, 123] This is another example of a PCG system that can generate entire games.

García-Ortega et al. [99] investigated NPC backstory generation for massive open world games. The authors focus on light-weight modeling techniques, because story generation techniques such as the Belief-Desire-Intention (BDI) approach were deemed too computation-heavy. Their intended application was generating character events for populating NPC backstories in a massive open

world game with many NPCs. The authors use a combination of evolutionary computation, agent-based models and logic programming for story generation.

ReGEN (REwriting Graphs for Enhanced Narratives) [155] is a quest generation system that tries to solve many of the known problems with story and quest generation systems, using a context-aware graph-rewriting framework. The system is similar to *Skyrim*'s Radiant Story generation system [27] for procedural quest generation. The authors aim to solve some of the problems of naive quest generation: better blending of manually authored content and procedural content; solvability of generated quests; and making sure that the quests both reflect and modify the game world in an interesting way.

The authors chose not to use emergent narrative in order to prevent that the stories feel like disconnected narratives. They also place some importance on generator controllability or authorial control, as a lack thereof is a known flaw of emergent narrative systems. Additionally, ReGEN does not change the game world to ensure that character goals can be attained, as this might interfere with other game content, for example manually authored content.

In ReGEN, both quests and the game world are represented as relational graphs. New quests are generated using graph-rewriting. This approach is an alternative to search-based, goal-oriented approaches that have been applied to story generation, which often requires heavy computation. This makes the latter approach less usable in a game development context, especially for online generation.

Malysheva [179] described a design and an implementation of an quest generator that takes the game world state into account during generation. The generator uses quest archetypes, abstract quest templates, as basis for quests. The archetypes are a high-level description of a quest type, where all the specifics are abstracted away into variables. The quest archetype also stores metadata that describes its purpose and context, which the quest generator can use to decide which quest should be instantiated next to progress the story.

Stockdale [255] has created ClueGen, a PCG system for creating text-based murder mysteries. The author describes in this paper why murder mysteries are especially suitable for both procedural content generation for games and story generation. Murder mysteries (and their closest relative in fiction, detective novels) tend to follow a fixed structure, which lends itself well to procedural generation: “one character has a reason to kill another, they do so, they hide the evidence, and an investigation occurs. (...) The entire story is often set in one location, with a small cast of characters who all have varying opinions of each other. As well as this, the actual purpose of detective novels which is to intrigue the reader and get them to attempt to predict the author's intent and determine the killer, means that there is an amount of ‘gameplay’ inherent in the plot, making this a perfect genre to attempt this marriage of gameplay and story.” Additionally, many story generation frameworks work with a cast of characters with their own beliefs and desires (BDI model), which fits this genre particularly

well, “as it will result in characters whose actions can be explained through their motivations (thus creating solvable crimes for the player to investigate).”

ClueGen is one of the first in a line of procedurally generated (murder) mystery games, which has been a popular topic of investigation in academia. Other examples include the Data Adventure series [20, 21, 22, 117], and Mike Cook’s recent information gathering games *Nothing besides remains* and *Condition Unknown* [73].

2.5.3 Qualitative procedural generation

In a 2016 vision paper, Johnson [142] defines the field of qualitative procedural generation (QPG), which “seeks to generate sociological and anthropological concepts and structures in order to develop deep, rich and believable game worlds.” Johnson has developed a experimental Rogue-like game called *Ultima Ratio Regum* (URR) [143] which explores this type of procedural generation. The paper describes four types of qualitative procedural generation from the game: religions and heresies, the generation of languages and styles of speech, the generation of political and ideological beliefs, and the generation of aesthetic styles tethered to particular cultures and reproduced throughout their material cultural artefacts. Although URR also procedurally generates the physical spaces in the game, these are dependent on the generated qualitative game content.

Real-life themes such as politics, social life and religion are already present in many games, but mostly in a static, non-living way instead of a realistic, varied and interactive context. Johnson’s aim for qualitative procedural generation is that these elements have “an effect on every part of the [game] world the player explores, and making different locations within the in-game world highly socio-culturally distinctive.” This type of procedural generation also leads to new forms of gameplay, where the player is incentivized to actively explore and search for information by deducing it from their environment. It also leads to culturally-grounded interactions with NPCs: instead of giving ‘rote’ responses in dialogue, NPCs can have varied conversations using language based on their cultural background. Johnson poses that qualitative procedural generation is not only useful for video games, but also other interactive media, such as historical simulations, museum exhibits, interactive television and interactive storytelling.

QPG is important to text generation for games because the qualitative themes and aspects that Johnson describes (in-game culture, history, religion) are often referenced in game lore, and consequentially in in-game texts. The qualitative elements can be the structures that procedural content generators need to ground their text in the game world, which we also saw in the procedural narrative generators discussed in the previous section.

Johnson’s work on *Ultima Ratio Regum* inspired others to investigate procedural culture as well. Something similar to Johnson’s procedurally generated dialect is used in Kreminski’s *Epitaph* [151]: the game generates a language for

every alien civilisation in the game. This generated language forms the basis of the name of planets, species, cities and prominent citizens of this civilisation, which are integrated in the related emergent micro-narrative. Everything else in the gameplay is static (the technology tree) or random (in-game events), so this is a form of qualitative flavor text generation for an emergent narrative game. In other words, generating procedural languages for NPCs can be a powerful method for creating distinctive, varied game content, even if the rest of the game is random or fixed.

Hall, Williams, and Headleand [126] created CELT (Character-Event Lore Time-line), a system that procedurally generates lore for video game worlds. Their lore takes the form of fables for a simulated religion. The authors explain that fables are often used as tools to pass down traditions and values, and religious stories express morals of an underlying belief system. Because of this connection between moral stories and religious values, the authors propose generating a religion first, i.e. values and gods, then generating fables, and then linking the fables to the religion. The system uses context-free grammars to generate natural language fables, based on a context-free grammar implementation of Christian Booker's "Seven Basic Plots".

This work uses an approach that is a recurring theme in procedural content generation: starting from the end to ensure the final output is complete, solvable or otherwise valid. First, the underlying structure is generated (in this case, the religion itself), and then the text that should express these structures (the fables). The game's player on the other hand, sees the game assets in the reverse order: the text is presented to the player, who can then use these to gather information about the underlying lore. This way of reverse generation, i.e. starting from the end, is reminiscent of an approach in poetry generation, where the poetry generator start with generating the final words or syllables of each line to ensure that every line rhymes correctly. ClueGen [255] uses a similar approach to generate solvable murder mysteries: first, NPC relationships are generated, then the murder, then the motive, and then the rest of the narrative. The emergent narrative elements are only to fill in the gaps between the start and the outcome of the narrative.

Grinblat and Bucklew [119] describe a system for procedurally generating coherent biographical narratives (biographies) for fictional historical figures from their Rogue-like game *Caves of Qud* [95]. The natural language text of the biographies is generated with replacement grammars, that use words from a hand-written knowledge base. The authors evaluated their work in the context of the game and the current user base.

Games that use similar techniques to generate cultural in-game artifacts are Bad News [228], which also contains generated history, and Voyageur [86], which uses grammars to describe extraterrestrial landscapes, biomes, and cultures.

Johnson [142] proposed QPG as an alternative application of procedural con-

tent generation, which has mostly focused on generating virtual ‘physical’ spaces and objects, such as mazes, cities and buildings. Interestingly, qualitative generation can give rise to procedural physical spaces (as in the case of *Ultima Ratio Regum*), and vice versa. The minecraft settlement generating challenge [230], a PCG challenge about generating minecraft settlements for a given procedural map, was recently expanded with a narrative component: participating generators can earn bonuspoints if they can also provide a narrative chronicle of the procedurally generated settlements. Similarly, Kybartas, Bidarra, Meyer, et al. [154] built a procedural population generator which was attached to a narrative generator for evaluation. Their population generator has properties of both space generators (terrain, maps) and culture (primary resource and resulting culture) generators.

2.5.4 Dialogue

In the previous section, we have already seen many games that include procedural dialogue. Johnson’s *Ultima Ratio Regum* [142] implements procedural languages and dialects, which in turn are used for generating NPC dialogue. *ClueGen* [255] and subsequent murder mystery games use NPC dialogue to allow the player to investigate the murder. Players can interrogate NPCs, ask them about their whereabouts and suspicions, gather clues, and accuse one of them of being the murderer.

DATA Agent [117] is another example of a murder mystery game generator. It creates games in which the player takes on the role of an agent in charge of solving a fictional murder of a real-world person. By traveling to various locations, interviewing non-player characters, and gathering facts about suspects, the player can find out which suspect committed the murder. All game content is based on real-world locations, people and topics, with data sourced from Wikipedia. DATA Agent uses Tracery [70] to generate NPC dialogues. Since the game revolves around gathering information by interviewing various people, NPC dialogue is essential to the gameplay. “The game has three types of dialog: essential dialog, fact-giving dialog and flair dialog. Essential dialog gives clues about game objects, progressing the story and making the locations containing them accessible. Fact-giving dialog reveals information about suspects which is important when the player is interrogating a suspect to fact-check their responses. Flair dialog gives information about the current NPC that the player is talking to, such as their birth date or what they are known for, and it is not essential to progress in the game.” [117, p. 6] The information for filling the gaps (items, names, locations, occupations, nationalities, personal facts) in dialogue templates is sourced from Wikipedia pages and their underlying relations.

Authoring tool Expressionist [223] was used to generate NPC dialogue for multiple games of the Expressive Intelligence Studio, which is part of the University of California, Santa Cruz (UCSC): *Snapshot*, *Project Perfect Citizen*, *Talk of the*

Town and *Juke Joint*. Expressionist was used in Snapshot to realize comments on in-game photoblog posts, as surface expressions of the game’s automated photo evaluation system. In Project Perfect Citizen, an Expressionist generates SMS conversations between procedurally generated NPCs. Production rules contain metadata such as relationship type, suspiciousness level and NPC personality traits. Talk of the Town [227, 225] is an AI-driven game where NPCs have their own subjective beliefs about the game world. Expressionist is used for dialogue generation, dialogue management, and natural language understanding. Talk of the Town expands the functionality of Expressionist applied in Snapshot and PPC with backtracking, preconditions, runtime variables and middle-out expansion. Juke Joint [224], which builds on Expressionist’s expansion in Talk of the Town, uses the tool for NPC dialogue authoring, and adds heuristic expansion to the list of features.

Lessard et al. [164] also used Expressionist for a game development project. In their paper, the authors discuss the generative power and usability of the tool in the context of creating NPC dialogue for their experimental game *Hammurabi*. *Hammurabi* is based on the resource management video game *Hamurabi* [91], in which the player takes on the role of a king of Babylon, supported by a steward, who must manage three resources (people, land, grain) to keep his people from starving. In each game turn, the steward reports on the state of the country, and players can choose between four actions: feeding people grain, sowing land with grain, buying land with grain and selling land for grain. Figure 2.2 shows part of a *Hamurabi* playthrough. The experimental *Hammurabi* expands on the game from the 1960s: the steward is substituted for three viziers, each with their own personality and interests, and public opinion is an extra resource that must be managed.

Some authors do not investigate procedural NPC dialogue in general, but focus on very specific elements of dialogue. Schlünder and Klabunde [233] described a system for greetings generation based on a formal model of politeness. Their computational model of greeting exchange is based on social variables such as ‘social distance’, ‘relative authority and power’ and ‘imposition of face threatening action’. The authors argue that even something as fundamental as greetings generation can still be improved in video games.

2.6 Games and coherence

Naturally, game content should fit the game in which it is included. Game content should clearly belong in the game and its narrative setting. We generally don’t expect ninjas in a Star Wars game, or laser guns in a strategy game with a medieval setting.² Moreover, as many games want to offer a polished experience

²Although some games explicitly play with the expectations of their audience by combining unrelated narratives tropes in a new and surprising way.

```

HAMURABI:  I BEG TO REPORT TO YOU,
IN YEAR 4 , 2 PEOPLE STARVED, 12 CAME TO THE CITY,
POPULATION IS NOW 62
THE CITY NOW OWNS 1010 ACRES.
YOU HARVESTED 1 BUSHELS PER ACRE.
RATS ATE 0 BUSHELS.
YOU NOW HAVE 319 BUSHELS IN STORE.

LAND IS TRADING AT 23 BUSHELS PER ACRE.
HOW MANY ACRES DO YOU WISH TO BUY? 0
HOW MANY ACRES DO YOU WISH TO SELL? 500

HOW MANY BUSHELS DO YOU WISH TO FEED YOUR PEOPLE? 500

HOW MANY ACRES DO YOU WISH TO PLANT WITH SEED? 230

YOU STARVED 37 PEOPLE IN ONE YEAR!!!
DUE TO THIS EXTREME MISHMANAGEMENT YOU HAVE NOT ONLY
BEEN IMPEACHED AND THROWN OUT OF OFFICE BUT YOU HAVE
ALSO BEEN DECLARED NATIONAL FINK!!!!

SO LONG FOR NOW.

```

Figure 2.2: Playthrough of Hamurabi (1968) from the book *BASIC computer games* [3]

to their players, manually authored and procedurally generated content should blend seamlessly with each other. Similarly, game assets should not feel like unrelated, isolated instances, but instead make up a coherent whole. As a result, naively generated content might be easy to spot.

Blending procedural content is an on-going challenge for creators of PCG systems. How can we make generated content indistinguishable from manually created content? Can we think of ways to hide it in plain sight? If the generator has created a flawed piece of content, can we trick the player into thinking it was deliberate? Various authors have criticized games for their lack of cohesion between game assets. In his description of qualitative procedural generation, Johnson [142] observed that cohesion, sense and shared internal logic are rarely found in most game spaces. Malysheva [179] names the discrepancy between game world and story world as the motivation for their quest generation research. Similarly, Kybartas and Verbrugge [155] criticize game narratives where the outcomes of quests (and related player choices) do not influence the narrative in the long run.

Of course, increasing the interconnectedness between game assets poses a risk for games that use procedural content generation. If all game assets are intertwined, accidentally inserting a defective generated game asset can have a large negative impact on the playability of the game. Game developers that want to use procedural content generation search for a balance between game asset independence and coherence.

In academic research, we can find the following examples of procedural con-

tent generation that tries to incorporate the game context into new assets. Kybartas and Verbrugge [155] incorporate the game world into their procedural generator by representing the game world as a graph, and applying graph rewriting steps to create new narratives that reflect and modify the game world state in an interesting way. This results in quests which can have an observable impact on the game world. Additionally, their automatic assessment for narrative quality of quests incorporates metrics for measuring the long-term influence of player actions.

Another method for generating coherent game assets is first generating a narrative basis which in turn influences all other game assets. This is for example used in game space generation, such as in Dwarf Fortress [2], where the historical simulation at the start of every game makes sure that the generated world is realistic. Similarly, Johnson [142] uses qualitative procedural generation to create socio-cultural concepts that act as this narrative basis. Johnson [142] argues that this will lend depth and meaning to the content that the player encounters in the game world.

Schlünder and Klabunde [233] argue that context-related language generation should be feasible in video games, as the necessary information about context is already present as game data. They experiment with a specific example, namely context-appropriate greetings for NPCs. Their algorithm incorporates physical and social context, such as line of sight between characters, character behavior (are they paying attention, are they trying to hide?), the appropriate physical distance, discourse history, and social standing. In contrast, Skyrim's greeting system is far more basic, as it incorporates fewer properties in greeting generation: character gender, character distance and faction memberships.

Taking game context into account during runtime is one of the important features of text generation tool Expressionist, according to Lessard et al. [164]. Expressionist incorporates arbitrary meta-data in its grammars, which can be used during runtime generation. Normally, context-free grammars are stateless, causing NPCs to seem stuck in an eternal present. In other words, if we use context-free grammars for game dialogue generation, this can lead to isolated sentences that do not show intertextual coherence. Lessard et al. [164] use meta-data to simulate 'memory' in their vizier NPCs. Expressionist can use this meta-data to create mini-narratives in the vizier dialogue. Viziers remember what was said over multiple game turns, which means they can comment on their past recommendations and previous player actions.

Another method for increasing the perceived coherence of generated assets is actively inviting the player to search for meaning, clues and relations. This is used in various mystery games, such as Cluegen [255], Condition Unknown [73] and DATA Agent [117], where the search for clues is part of the gameplay. Some games do this more implicitly. Johnson [142] describes a procedural world which contains cultures, all of which can be distinguished by their own aesthetic. Even without explicitly telling the player that two game objects, e.g. household

items, buildings, clothing items, and NPCs, are part of the same culture, the aesthetic properties should provide enough information for the player to link them to specific cultures. “The responsibility lies with the player to connect items they discover to other items they’ve seen before, and gradually build up a comprehensive picture of the world’s generated cultures, and use this information to inform their decisions and actions within the game world.” [142, p. 5]

Even when connecting various game assets is not demanded of the player, players will generally see patterns or design intention in generated game assets, even when there are none. Stockdale [255] also observed this behavior during playtesting for the murder mystery game *ClueGen*: “A surprising finding from playtesting was that once players were impressed with the NPCs’ abilities to lie, they were willing to attribute unusual behaviour to this deviousness. For instance, when one player realised that several innocent characters had unwittingly moved the murder weapon as far from the scene of the crime as possible, he assumed that they were conspiring with the murderer to hinder the investigation. (...) If the AI’s capabilities are demonstrated but not overtly explained, players are able to fill in the gaps and embellish on the plots presented them.” [255, p. 6] This tendency is similar to the Eliza effect described by Hofstadter [129], who noticed that humans will attribute intelligence or empathy to (text-producing) computer systems. It is also related to the ‘charity of interpretation’ effect studied by Veale, who found that “readers will generously infer the presence of meaning in texts that are well-formed and seemingly the product of an intelligent entity, even if this entity is not intelligent and the meaning not intentional.” [276, p. 3] Veale observed that if humans see a text in a well-known form (or *container*), they are disposed to attribute more meaning to the text than it actually contains.

This effect can be exploited by designers and generators. For example, the creators of *DATA Agent* [117] used the narrative of the game to explain accidental inconsistencies present in the narrative. “The player takes the role of a detective working for the Detective Agency of Time Anomalies (DATA). In the game’s universe, criminals can go back in time and murder famous people, altering the time line and creating inconsistencies. To prevent this, DATA sends agents to the past to catch the killer and prevent the murder. Since the very act of going back in time messes up time lines, DATA has incomplete information about the past, and can only tell the player who the suspects are.” [117, p. 3] This strategically chosen context is communicated in a cut scene at the beginning of the game. This small amount of backstory makes the generated game content more believable: even if the player encounters contradictory or missing information created by the generator, this can be plausibly explained by the game universe sketched in the backstory.

The game developers of *Caves of Qud* [119] use a similar approach for generating in-game biographies of sultan NPCs. The authors achieve a sense of coherence within the biographies by exploiting the human tendency for ‘apophenia’,

i.e. human players tend to rationalize the events presented to them, especially if the game only presents partial information. Instead of creating fictional historical accounts using a simulation or logical reasoning, the historical accounts are created in reverse order: first the event is generated, and then the reasoning behind it. The developers increase the perceived coherence of the biographies even further by assigning every NPC an underlying theme or domain, which determines the details of the biography events. For example, a sultan with the domain ‘ice’ gets assigned biography events that incorporate the color lightblue, snow, crystals, etc.

In this section, we explored the relation between procedural text for games and coherence. We looked at research papers and existing games, and discussed how these create a sense of coherence in generated game assets. In the discussed examples, we can find the following two design goals for procedural content generators:

1. Game assets should cohere with the game and with each other; to achieve this, procedural generators need to strike a balance between stand-alone game assets and tightly interwoven game assets. The former is more robust but might be boring, the latter is more interesting and engaging, but might break the game if part of the generation goes wrong.
2. Game assets should cohere with player actions. Ideally, player choices and actions change the game world in a meaningful and interesting way. However, changing the game world will also influence procedurally generated assets, which might not be robust enough to deal with arbitrary changes to the game world.

From both research literature and existing video games that use procedural text generation, we can distill the following two radically different approaches for incorporating coherence in procedural text:

1. We can increase coherence by grounding procedural assets in the game world, by incorporating information about the game world and game state during generation, and working around manually authored content;
2. Or we can focus on increasing the perceived coherence of generated game assets by capitalizing on player expectations, and actively encouraging players to search for patterns in and connections between game assets – even when there are none.

2.7 Conclusion

Procedural content generation is the research field that investigates how we can create large volumes of game content through algorithms. When we generate

textual assets, such as character dialogue or game narratives, we can draw on both general-purpose, content-agnostic PCG approaches and natural language generation techniques. When using PCG techniques in a video game, we need to ensure that generated assets are not easily distinguishable from manually created content, and that they do not feel like isolated, meaningless game assets. Therefore, procedural content generators should incorporate some form of coherence with the underlying game. We can increase the perceived coherence of generated assets either by grounding them in the game world, or by evoking the feeling of coherence by exploiting the human tendency to see patterns in unrelated data. Evoking a sense of coherence will play an important role in later chapters, especially Chapter 4 and Chapter 5.

Natural language generation

In this chapter, we discuss the background on natural language generation that is relevant to the work in this thesis. The chapter starts with a high-level overview of natural language generation. We list applications, definitions of key concepts and the most important high-level approaches. We then describe NLG methods that are particularly relevant to procedural text generation for video games. First, we discuss templates and formal grammars, two simple methods that are often used in procedural text generation. We then discuss a new development in NLG: the Transformer architecture, and focus on GPT-2, a specific type of language model that is built on this architecture. Section 3.4 describes story generation, a subfield of NLG that is particularly relevant to PCG for narrative video games. Finally, we discuss evaluation of NLG systems in Section 3.5.

The reader should note that natural language generation is not the focal point of this dissertation, but a tool that we can use to obtain our goal of generating coherent game texts. Instead of focusing on the underlying theory of natural language generation, we investigate NLG techniques to find out whether and how they can be used for game development. For a more in-depth overview of natural language generation research, we refer the reader to the recent survey by Gatt and Krahmer [101].

3.1 Definition and applications

Natural language generation (NLG) is the task of generating text or speech from non-linguistic input [101]. NLG is part of natural language processing, a multi-disciplinary research area that combines linguistics, computer science and artificial intelligence.

Natural language generation systems have been designed and implemented across many tasks and domains. As an example, take the following non-exhaustive list of applications of NLG systems:

- report generation, such as weather reports [114, 112], soccer reports [265, 160], financial reports [152, 195] and medical reports [137, 111]
- expert systems and decision-making support systems, for example for medicine [102, 234, 132] or finance [47]
- automated journalism or robot journalism [171, 163]
- dialogue generation [57, 217, 167, 136], for example for conversational agents and characters in digital games
- generation of creative text, such as stories [109, 153], poetry and lyrics [198], and procedural narrative for games [241]
- tailoring texts to a specific audience [160] and generating persuasive texts [214, 52, 46]

3.2 Approaches

Although there is consensus that an NLG system should produce natural language text, the inputs of NLG system vary greatly.

Text-to-text generation is generating text using natural language text as input. Examples of text-to-text generation are machine translation (e.g. English text is transformed to Dutch text), summarization (i.e. a long text is transformed to a shorter text), text simplification, spelling and grammar correction, question generation (given a text, generate a set of questions about that text) and question answering (given a text and a question about that text, formulate an answer to the question).

Data-to-text generation, on the other hand, is generating text using data (or rather, data that is not exclusively text) as input. Examples of data that can be used as input for data-to-text systems are numerical data, time-series data, semantic representations, graphs, sensor data, images, video and sound. Gatt and Krahmer [101] note that the lines between text-to-text and data-to-text are slowly blurring, as most modern approaches to text-to-text generation might also incorporate additional non-linguistic data, and vice versa. Indeed, we will see in this thesis, which mostly builds on text-to-text generation, that additional data sources are often used in practice. The previous chapter showed that text generation for games can be vastly improved by drawing on runtime information from the game world and game state, which is often non-linguistic in nature.

Gatt and Krahmer [101, p. 83] classify NLG systems in two orthogonal ways, namely based on their design and on the methods adopted in their development. Design-wise, we can distinguish between modular, planning-based and global (or end-to-end) approaches. *Modular* NLG systems divide the natural language generation in a series of steps.

In their classic paper on data-to-text NLG [213], Reiter and Dale describe these steps for natural language generation:

1. Content determination: what is the content, i.e. what do we want to say?
2. Text structuring: in what order should the content be presented?
3. Sentence aggregation: deciding which information to present in individual sentences,
4. Lexicalisation: finding the right words and phrases to express information,
5. Referring expression generation: selecting the words and phrases to identify domain objects,
6. Linguistic realisation: combining all words and phrases into well-formed sentences.

The authors divide these six steps into a three-staged pipeline architecture, with stages for text planning, sentence planning and linguistic realisation.

However, not all modular systems use the above tasks and stages. Some systems use a subset of these tasks, or add extra steps. For example, Reiter's data-to-text architecture [211] adds signal analysis and data interpretation steps. This design choice is dependent on many variables, including the intended application and users of the NLG systems, the inputs of the NLG system, the intermediate representation used for each task and stage, and the knowledge required for developing each module. If a modular NLG system uses the above steps or a variation thereof, it is commonly referred to as a (classic) NLG pipeline.

Planning-based approaches draw from research in logical reasoning and automated planning. These approaches view the task of natural language generation as a planning problem, where multiple computational actions should lead to successfully achieving a communicative goal. Koller and Petrick [150] note that these approaches have not been as popular as the classic pipeline and end-to-end approaches. However, planning-based approaches have been used for applications such as story generation (e.g. [216]) and dialogue generation (e.g. [203]).

Global, integrated, unified, or *end-to-end* approaches, do not split the generation task into multiple subtasks, but instead transform the input data to the output in one unified go. These approaches typically rely on data-driven, statistical methods.

In terms of NLG methods, we can broadly distinguish rule-based, statistical, machine learning and deep learning approaches to natural language generation.

Examples of rule-based approaches are templates and grammars. Template-based NLG is generating natural language text by filling gaps in a pre-written template. This method can be used stand-alone, as a simple, direct way of linguistic realisation [212], or as one step in a larger, more complicated NLG system [81]. Puzikov and Gurevych [206] developed both a neural and template-based

NLG system for the E2E NLG challenge in 2018. The E2E NLG Challenge is a shared task, in which participants build NLG systems that can generate restaurant descriptions from lists of restaurant properties. Puzikov and Gurevych compared the performance of the two systems, and found that in some cases “the costs of developing complex data-driven models are not justified and one is better off approaching the problem with simpler techniques.” [206, p. 467]

Formal grammars, e.g. context-free grammars, can be used for defining recursive templates. Grammars are very powerful. They can be used to express a wide range of rules, and there are many types of formal grammars, which means a grammar type can be chosen depending on the application. For example, rules of context-free grammars can be expanded with probabilities or data types, or with user-defined annotations such as in authoring tool Expressionist [223]. Grammars and their generative space can also be relatively easy understood by non-experts, which makes them accessible tools for creating natural language generation systems. Formal grammars (or similar combinatorial rewriting systems) are already frequently used in game development, for procedural generation of both textual (e.g. Tracery [70], Expressionist [223]) and non-textual game assets. Originally, the templates, grammars and rules used by rule-based NLG system were written by hand. This is labor intensive, and for large rulesets, hard to analyze and maintain. However, with the increased interest in data-driven methods, researchers have started to investigate how we can automatically learn rules for NLG from corpora, thus combining statistical and rule-based methods for NLG.

Statistical methods, including those that use machine learning or deep learning, typically rely on datasets of example text, or *corpora*. In some cases, methods need corpora of *aligned* texts, i.e. pairs of corresponding inputs (data) and outputs (text), to learn the desired generative space.

NLG with machine learning and deep learning typically use sequence-to-sequence (seq2seq) models and recurrent neural networks (RNNs) [90]. The encoder-decoder architecture is a prevalent architecture in neural NLG. In this architecture, an encoder neural network translates an input sequence to a vector of a fixed length. A decoder neural network subsequently decodes the resulting vector into an output sequence. The problem with this architecture is that every input has to be translated to a fixed length vector, which might lead to problems for inputs that are long, or longer than the sequences in the training set [63]. Bahdanau, Cho, and Bengio [14] addressed this problem by adding attention mechanisms, an extension to the encoder-decoder architecture that lets the neural network direct its attention to specific parts of the input sequence. Instead of encoding the entire input sequence into a single vector of fixed length, the neural network encodes the input sequence into a sequence of vectors. During decoding, the system dynamically chooses a subset of these vectors. In other words, the decoder focuses only on those parts of the input sequence that are relevant for the current output. More recently, Transformers [275], a type of neural lan-

guage model, have gained in popularity. Transformers perform well in a variety of natural language processing tasks, and are now increasingly used for creative language generation.

Whether statistical methods can be used depends on the availability of corpora for a given domain or task. There are many instances in which there is no example data available for a particular application domain, task, or language. A lack of available corpora can pose a challenge to the development of NLG for specific application domains or tasks, as example data is necessary for extrapolating rules and training machine learning models.

Researchers have been trying to mitigate these challenges in various ways, for example by aligning existing datasets automatically, creating new datasets using crowd-sourcing techniques, investigating methods that require fewer datapoints to begin with, and investigating statistical methods that do not require aligned data.

3.3 Popular NLG methods for PCG

Rule-based techniques for procedural generation are already commonly used in game development practice, probably because those are more accessible to people without a background in computer science than more complicated techniques. Consequentially, the work in this thesis mainly draws on rule-based techniques, especially templates and grammars. In addition to rule-based methods, we will explore Transformers, a recent architecture for sequence processing based on deep learning that performs particularly well in creative applications.

3.3.1 Templates

Template-based NLG is generating natural language text by filling gaps in a pre-written template. It is one of the simplest methods for text generation. Figure 3.1 shows an example of an email template. When combined with a database of customer information of a webshop, this template could be used to automatically create a large volume of emails.

3.3.2 Formal grammars

Formal grammars can be used for text-generation. In fact, formal grammars can be used for defining recursive NLG templates. Context-free grammars are the type of grammars that are most used for text generation. They were originally defined by Chomsky [64] in the 1950s, but similar mathematical formalisms were used long before that.

Context-free grammars are formal grammars consisting of an alphabet V of non-terminal symbols, a finite set of terminal strings Σ , a start symbol $S \in V$, and a finite set of production rules $R \in V \times (V \cup \Sigma)$ that specify how grammar

Dear FIRST_NAME LAST_NAME,

Thank you for buying our product PRODUCT in WEBSHOP.
 Could we interest you in SECOND_PRODUCT?
 We can offer you a code for a discount of DISCOUNT_PERCENTAGE:

CLIENT_DISCOUNT_CODE

Kind regards,

COMPANY_NAME
 COMPANY_ADDRESS

Figure 3.1: Example of a text-generation template for a marketing email. The variable names (in all-caps) represent gaps that can be filled in based on spreadsheet data to create a set of emails.

$$\begin{aligned}
 V &= \{D, G, N\} \\
 \Sigma &= \{\text{"Hello"}, \text{"Nice to see you"}, \text{"I've heard all about you"}, \\
 &\quad \text{"hero"}, \text{"high one"}, \text{"champion"}\} \\
 R &= \{D \rightarrow GN, \\
 &\quad G \rightarrow \text{"Hello"} | \text{"Nice to see you"} | \text{"I've heard all about you"}, \\
 &\quad N \rightarrow \text{"hero"} | \text{"high one"} | \text{"champion"}\} \\
 S &= D
 \end{aligned}$$

Figure 3.2: Context-free grammar for generating NPC greetings in a fictional game. The pipe symbol | in production rules denotes a choice.

symbols can be rewritten. Grammars are an application-neutral formalism. If we want to use them for natural language generation, the terminals of the grammar consist of natural language phrases. Figure 3.2 shows a context-free grammar that could be used for generating non-player character (NPC) dialogue lines in a video game.

This grammar can generate dialogue lines such as “I’ve heard all about you, hero.” and “Nice to see you, champion.” by rewriting starting symbol D using the production rules, until no non-terminals are left.

Context-free grammars are only one specific type of grammars, and many other types of grammar (typed grammars, context-sensitive grammars, L-systems) exist. Grammars have considerable expressive power, which means they can be

adapted according to various tasks and domains. For example, Markov chains, simple statistical language models, can be fully expressed as formal grammar. As an added bonus, context-free grammars are relatively easy to understand for non-experts, which makes them an accessible tool for creating natural language generation systems. Formal grammars or similar combinatorial rewriting systems are commonly used in game development, for procedural generation of both textual and non-textual game assets.

3.3.3 Transformers

The Transformer [275] is a neural architecture for sequence transformation and sequence modeling. As natural language text is *sequential*, i.e. a sequence of letters, words and sentences, Transformers are suitable for natural language processing tasks. The design follows the encoder-decoder architecture for machine learning. Researchers noticed that the performance of these models could be further improved by adding attention mechanisms. In contrast to typical encoder-decoder models, which use recurrent or convolutional neural networks as building blocks, Transformers are based solely on attention mechanisms. This has two advantages. Attention mechanisms are better at modeling dependencies between inputs and outputs that are far away from each other in the sequence. Unlike recurrent neural networks, attention mechanisms do not have to perform their computations sequentially. As a result, attention mechanisms can be used with parallelization, which allows for faster computation.

Vaswani et al. [275] evaluated the new architecture on English-to-German and English-to-French machine translation. The results were automatically scored with BLEU [202], a similarity metric for scoring machine translation. The authors found that their Transformer model outperformed the best previously reported models (including ensembles) for English-to-German. The model also outperformed all of the previously published single models for English-to-French translation, at less than 25% of the training cost of the previous state-of-the-art model. To measure the performance of the different components of the Transformer architecture, the authors varied their base model in different ways. Among other things, they found that bigger models perform better than smaller models.

After the publication of the Transformer architecture, various research groups started building upon this idea and training their own language models. Google Research created BERT [85], which in turn inspired various other language models and architectures, such as RoBERTa [174] and DistilBERT [231]. OpenAI created GPT [208], GPT-2 [209] and GPT-3 [49].

The original models model the English language only, as their training corpora consisted mainly of English language text. However, the models are not necessarily limited to applications in English. Transformer models can be fine-tuned on additional training data, including training data in other languages. GPT-2 is known to reproduce or generate programming code and XML, including

correct syntax and balanced tags. New BERT models have been trained for both single languages, including languages other than English [278], and multiple languages [221].

3.3.4 GPT-2

GPT-2 [209] is a set of pre-trained language models by OpenAI. The largest language model has 1.5 billion parameters.¹ Its predecessor is GPT [208], a pre-trained model built on the Transformer architecture. Vaswani et al. [275] observed that increasing the size of their Transformer model leads to an increase in performance, so OpenAI trained a series of language models with increasing size, both in terms of training set size and parameters.

The developers of GPT-2 trained the model to predict the next word in a sentence in an unsupervised way on a large corpus of scraped web text. Even though GPT-2 has not been finetuned on datasets for specific tasks, the model performs well on a variety of NLP tasks. This is because many NLP tasks can be encoded as natural language sequences: tuples of the input, output and task. “For example, a translation training example can be written as the sequence (translate to french, english text, french text). Likewise, a reading comprehension training example can be written as (answer the question, document, question, answer).” [209, p. 2] The creators of GPT-2 hypothesized that a model for predicting the next word in a sentence (sequence) should be able to infer tasks and then perform them if they are demonstrated in natural language sequences, as long as the model has sufficient capacity, and the training data is large and diverse enough.

In terms of capacity, GPT-2 is already much larger than the first OpenAI model GPT. GPT-2 has been released in four sizes, ranging from 124 million parameters to 1.5 billion parameters. All sizes pre-trained models are freely available. The release of the biggest GPT-2 model was at first postponed for fear of abuse. The pre-trained models can be finetuned with smaller datasets for specific language processing or language generation tasks.

GPT-2 has an even larger successor, GPT-3, with a similar architecture but even bigger model capacity. However, GPT-3 is, as of April 2021, still in ‘private beta’, which means only a select group of users can use the model. GPT-3 is only available via a commercial API, so the language model cannot be inspected fully. The creators of GPT-3 have written an open access paper about GPT-3 [49] and shared some sample data, but the full-size models and their underlying training data are not available.

3.3.4.1 Training corpus

The training corpus of GPT-2 is called WebText. This corpus consists of 40 GB of text data, or approximately 8 million webpages, scraped from the internet. In

¹https://github.com/openai/gpt-2/blob/master/model_card.md

order to obtain high quality text data, the researchers only scraped webpages that could be reached via a Reddit post that received at least three karma from Reddit users. Reddit is a social media platform where users can share links or text. Other users can vote for the best submissions using a point-system called karma, which is essentially a crowd-sourced quality ranking of posts and comments. This led to a higher data quality than can be found in similar webcorpora, such as CommonCrawl.

Unfortunately, WebText is not publicly available. The open-source community is trying to create an opensource version of WebText, such as the OpenWebText Corpus by Brown University researchers² and a similar-named³ project by Joshua Peterson. Other researchers are trying to build similar datasets, but for other languages. Various initiatives exist to create other large, high-quality corpora, such as OSCAR [201], a massive multilingual corpus based on a cleaned and shuffled version of CommonCrawl⁴. However, even if datasets of WebText's size are available, the sheer size of the language model makes training a challenge for common computer systems.

3.3.4.2 Usability

GPT-2 is relatively easy to use, even for non-experts. The model has been integrated in various open source projects, such as HuggingFace's Transformers⁵ library, and it is easy to find blogposts, tutorials, code examples and interactive demonstrations online. Some developers, such as Max Woolf, have written extensions or wrappers for GPT-2 that make it easier to use, either locally⁶ or deployed in a cloud environment⁷. The resulting accessibility of GPT-2 has contributed to a wide-spread adoption for creative language generation projects, such as games and Twitter bots.

3.3.4.3 Challenges

Despite their enthusiastic adoption and wide-spread media-attention, GPT-2, and similar large language models, have various problems and limitations that still need to be addressed. Some of the problems are inherent in large language models, and some are specific to GPT-2. Some problems can have significant societal impact, especially when the prevalence of large language models increases. Besides their many beneficial applications, large language models can be used for various malicious purposes, such as fake news [290, 24], online abuse, and building spambots [244, 98].

²<https://skylion007.github.io/OpenWebTextCorpus/>

³<https://github.com/jcpeterson/openwebtext>

⁴Common Crawl, <https://commoncrawl.org/>

⁵Transformers by HuggingFace, <https://huggingface.co/transformers/>

⁶gpt-2-simple by Max Woolf, <https://github.com/minimaxir/gpt-2-simple>

⁷gpt-2-cloud-run by Max Woolf, <https://github.com/minimaxir/gpt-2-cloud-run>

Large language models like GPT-2 can reflect various biases, such as biases pertaining to race, gender, sexual orientation [238, 244] and given names [242]. Researchers found that specific prompts can trigger GPT-2 in generating racist or offensive text [106, 281]. The biases that we can find in GPT-2, which was trained on web text, are reflective of the biases we can find in English online text. Consequently, some researchers are now arguing for stricter selecting and curating of training data for large scale language models, to reduce the latent bias in these models, e.g. [25].

GPT-2 has problems generating factual text. An evaluation of arguments generated by GPT-2 showed that it tends to generate a mix of opinion and facts [118]. In neural text generation, neural networks sometimes predict outputs that are factually incorrect given the input [220, 62]. An example of this can be found in image caption generation, when neural networks generate captions about objects that cannot be found in the image. This effect is called hallucination [220]. GPT-2 and derivative language models are also susceptible to generating factually incorrect text. The fictitious newspaper article about the discovery of Ovid Unicorns, which was included in the original GPT-2 paper [209, p. 20], is a good example of this, although we should keep in mind that the paragraphs of the newspaper article were cherrypicked from multiple generation rounds.

GPT-2 might generate text that loses its coherence over time. Sometimes the generator seems to lose track of the textual structure, or the topic under discussion. Holtzman et al. [130] observed that some sampling methods for GPT-2 may lead to text degeneration, e.g. generating text that is bland, incoherent or repetitive. The authors propose nucleus sampling to overcome the degeneration problem. Radford et al. [209] already observed that the smallest GPT-2 model tends to converge on repetition.

Finally, GPT-2 has a language bias. GPT-2 is strongly biased towards English, which stems from the English language training data that was used to create the pre-trained model. To mitigate the strong bias towards English language text, researchers have investigated whether GPT-2 can be tuned towards other language, such as Dutch and Italian [279]. Interestingly, OpenAI observed that the language bias was less prominent in the largest GPT-2 model with 1.15 billion parameters [244].

3.4 Story generation

Although most NLG research focuses on producing non-fiction text, there are various applications for NLG in the creative text domain. We will focus on story generation, as this is the most relevant topic in the context of text generation for narrative video games.

We can also find other text generation techniques that are useful for PCG for video games, for example in research for generating poetry, lyrics, jokes, advertising slogans, catchy titles, curiosity-inducing news headlines, and movie

scripts. Research related to these topics will not be discussed here, but in the background sections of relevant chapters.

3.4.1 Definition and key concepts

A problem with story generation is that it is not easy to formally define the problem that story generator systems should solve. In his 2012 survey of Story Generation Algorithms (SGAs), Gervás [109] observes that for story generation systems, it is not clear what the input should be, nor what the characteristics of the output should be. This makes it hard to evaluate the performance of a single system, and compare multiple systems.

Gervás focuses on computational procedures resulting in an artifact that can be considered a story. The definition of story that Gervás uses is purely functional, not aesthetic. For Gervás, the quality of the surface realisation, i.e. whether the story consists of “readable and appealing text” is of lesser importance. The survey mentions systems that at least produce a *fabula*, or a sequence of events that is an abstract representation of a story.

In their 2017 survey on story generation techniques, Kybartas and Bidarra [153] use a formal model of narrative based on structuralist narrative theory. The authors use this framework to identify which elements of a story are most commonly generated.

The authors provide the following definitions of core concepts in computational narrative research, based on earlier computational narrative surveys and structuralist narrative theory.

1. *Narrative* is defined as having a *story*, the main content, and a *discourse*, which is the particular telling of a story.
2. *Story* is the main content of the narrative; this includes what happens in the narrative, the plot, and the space in which the narrative occurs.
3. *Plot* is a set of events with an overall structure which represents both the temporal ordering, and the causal relations between the events. Events typically consist of one or more low-level actions, instigated by and/or affecting a number of entities in the space.
4. *Space* includes *existents*, or the characters, settings, props, and anything which is present either physically or abstractly in the space of the narrative. Because existents change and evolve over time, the space also consists of an initial state which contains the set of all existent states as they exist before the start of the plot. Space is also called world or game world, depending on the context of the research.
5. *Discourse* is the particular telling of a story. This may include the style of the space, the ordering and duration of the events in the plot, etc.

Similarly to the survey by , the authors focus mainly on story generation (the events being told) instead of discourse generation (the telling).

The definitions employed in the surveys by Gervás and Kybartas and Bidarra are very similar. However, we can find large differences in the definition of ‘story’ and the task of story generation in existing research. Mostafazadeh et al. [191] define a *story* or *narrative* as “anything which is told in the form of a causally (logically) linked set of events involving some shared characters.” In contrast, the definition of story by Kybartas and Bidarra [153] does not mention characters. Yao et al. [289] include both fabula generation and surface text realisation in their definition of story generation.

The survey by Gervás [109] describes systems that produce static stories. However, there also exist *interactive* storytelling applications, which allow a user to influence the events in the story. As examples, Gervás mentions interactive fiction (IF) and 3D simulated worlds similar to video games. There are also systems that focus on creating the surface realisation of stories from a fabula, such as STORYBOOK [51], the Narrator [266] and Curveship [186], and those that focus on various aspects of the telling of stories, for example stylistic variation [187], affective language [256] and personality [177, 280].

3.4.2 Story generation systems

Gervás describes the first storytelling system on record, the Novel Writer system by Klein et al. (1973), which is a system that generates murder mysteries. We saw something similar in the procedural content generation field, i.e. see Section 2.5.2, where murder mystery games are a common genre for generative systems. Novel Writer takes a description of the story world and the story characters as input. Stories are generated using two different sets of rules: one set determining the possible changes to the story world, and one set defining a logical sequence of events. This type of rule-based story generation is still used in practice. One of the drawbacks of this method is that the ruleset severely limits the type of story that can be generated, so this type of story generators is typically not generalizable across genres.

TALESPIN [185] is a story generator that can create stories about woodland creatures. In contrast to the Novel Writer system, it is planning-based. Characters in the story world have goals, and the story generator uses character goals to drive the actions in the story. Talespin introduced the idea that a story can use multiple characters with different (and sometimes conflicting) goals. Talespin also models character personality traits and relations between characters, which in turn influence character actions in the story.

AUTHOR [82] is a story generator that experimented with modelling author goals (or meta-goals) instead of character goals. The rationale behind this is that story events are determined by author goals, such as creating a coherent or believable story, or grabbing the reader’s attention.

Lebowitz [158] created UNIVERSE, a story generation system for soap opera scripts. The system focuses on creating never-ending stories for a large cast of characters with interweaving stories. This is different from earlier story generators, which focus on stories with a beginning and an end. Lebowitz also addresses the question whether the story world should be created first and the plot second, or whether the story world should be generated on the fly in service of the plot, for example by adding characters, items and locations as needed.

This is reminiscent of a similar debate in procedural content generation. In procedural generation, which approach can be chosen depends on many factors, including what other game assets are used. For example, if a story generator is used in conjunction with human-created game content, the generator should work on top of the manual content, and change the game world in only a minimal way to prevent breaking the game. On the other hand, the generator should create meaningful new content, which might require changing existing assets. Kybartas and Verbrugge [155] discuss this trade-off well in their work on quest generator ReGen.

In his story generation survey, Gervás also discusses the various algorithm types of the mentioned story generation systems. Existing story generators combine one or multiple of these types. In planning based approaches, the generator takes a formal description of an initial world state, and a desired goal, and produces a sequence of actions that will lead to the realisation of that goal in the story world. Planning based approaches can work on different levels of goals, such as author goals and character goals. TALESPIN also combined two “directions” of planning, which Gervás calls forward-chaining, i.e. first events and then their outcomes, and backward-chaining, i.e. first goals and then events that will lead to the outcome of the goal. Planning approaches can also reason about multiple aspects of the story at the same time, as demonstrated by FABULIST [215] which simultaneously reasons about causality, character intentionality and motivation. Another approach is using “a set of resources that abstract key elements of story structure”, such as story grammars and templates. Finally, Gervás mentions systems that “mine a set of previous stories to obtain material they can reuse in building new ones”, i.e. statistical approaches and data-driven approaches.

Kybartas and Bidarra [153] categorise and discuss relevant research based on two elements of stories, i.e. plot and space, and the degree of automation of those elements. They place relevant research in the abstract space of computational narrative authoring, of which the two axes are plot automation and space automation. The authors divide the space in four quadrants: manual story authoring, space generation (partially automated space but manual plot authoring), plot generation (vice versa), and story generation. True story generation, for Kybartas and Bidarra, are instances of mixed-initiative story creation where both the plot and space creation are done at least partially automatic. The authors discuss the entire range in story creation tools: from authoring tools for both manual plot and space creation, such as tools for writing hypertext, educa-

tion and therapy, to full story generation systems where both plot and space are fully automated.

Kybartas and Bidarra [153] identified five degrees of plot automation: *manual*, where the story creation is mostly done by a human author; *structure*, where the computer provides a structure to fill in but does not contribute to the content; *template*, where the computer creates a sequence of ordered events, *constrained*, where the computer generates a full plot according to a human-defined specification; and *automated*, where author involvement is minimized.

Similarly, they identify five degrees of space automation: *manual*; *modification*, where the computer modified parts of a human-written space; *simulation*, where the computer generates new content for a space using simulated interactions between existing content of that space; *constrained*, similar to constrained plot automation, and *automated*, where the human author contributes as little as possible.

3.5 Evaluation

Currently, one of the biggest open problems in NLG is how we can evaluate NLG systems. In NLG, evaluation methods vary greatly from research to research, which makes it hard to compare multiple systems, and thus draw conclusions about the efficacy of different approaches. This large variation in evaluation methods is partially caused by the lack of definition of an NLG system, and the resulting variety in their inputs, outputs and applications.

Spärck Jones and Galliers [245] describe various ways of approaching evaluation of NLP systems. Since NLG systems are a subset of NLP systems, their description applies to NLG systems as well. They distinguish two types of evaluation criteria: intrinsic and extrinsic criteria. *Intrinsic* criteria are properties that pertain to the task that the NLP system should solve, whereas *extrinsic* criteria are about the larger goal of the environment in which an NLP system is used. In other words, intrinsic properties are about the ‘what’ (the task) and extrinsic properties more about the ‘why’ (the reason we want to solve the task). Spärck Jones and Galliers [245] note that there is a difference between evaluation of the NLP system itself, and the larger environment in which the system is used, called the *setup*. Both the NLP system and the setup as a whole can be evaluated intrinsically and extrinsically.

Gatt and Krahmer [101] expand on the categorization by Spärck Jones and Galliers, and describe the dominant methods for both intrinsic and extrinsic evaluations of NLG systems [101, p. 123–130]. In intrinsic evaluations of NLG systems, outputs are rated on a set of properties. The ratings are either given by human judges or computed using statistical metrics and reference corpora of example texts. Intrinsic studies often measure a proxy of the intended effect of a system, instead of the effect itself. Intrinsic evaluations involving human judgements are normally done in an artificial context, i.e. a laboratory setting.

However, we can also ask users to interact with the NLG system in a real-world context, and afterwards ask them to rate the system on a set of properties, to lend more ecological validity to the intrinsic evaluation.

In extrinsic evaluations of NLG systems, we measure effectiveness in achieving a desired goal. The definition of effectiveness is dependent on the application domain and the task that the NLG system should solve. In the context of NLG, extrinsic evaluations assess a system's effect by measuring its effect on a user's performance of a task in a real-world setting. Extrinsic studies are relatively costly in terms of time and expense. They also require a large enough group of suitable research participants to evaluate the system in a real-world context in a statistically significant way. For example, for an extrinsic evaluating of an NLG system that produces game text, we would need a large enough group of players or game developers that want to use the system in practice.

Evaluation is often a balancing act that depends on the goals of the research project and the weight given to proper evaluation by the researcher and the reviewers. It also depends on the properties of the NLG system under evaluation and their complexity. If we want to evaluate the grammaticality of textual outputs, an automated metric or a survey could suffice. On the other hand, if we want to evaluate more abstract or harder to operationalize properties such as groundedness or usefulness, we might need a more extensive and ecologically valid evaluation. If we want to evaluate the effect of the text generator as a whole, evaluating it in a real-world context might be more appropriate.

Gatt and Krahmer [101] point out that according to recent studies, the different evaluation methods often do not converge on verdicts. This is the case for ratings of both single NLG systems and performance rankings between multiple systems. In fact, meta-evaluation literature might contradict each other. For a recent overview of the debate surrounding NLG evaluation, we refer the reader to Gatt and Krahmer [101, p. 121].

Spärck Jones and Galliers [245] also distinguish between *black box* and *glass box evaluation*. Black box evaluation only considers system input-output relations, whereas glass box evaluation also considers the mechanisms linking input and output, i.e. the underlying processing steps [245, p. 26]. In a glass box evaluation, the contribution of the separate components or parameters are evaluated. For this we need an evaluation setup where the contribution of each components can be measured separately and subsequently judged uniformly, for example by disabling one component at a time and measuring the differences in the generated results. However, as Gatt and Krahmer [101] observe, exhaustive component-wise comparisons are sometimes difficult to make and may result in a combinatorial explosion of configurations. An example of the glass box evaluation approach can be found in the paper describing the Transformer architecture [275], in which the authors discuss the contribution of the attention mechanisms and various model parameters by showing their influence on the performance on machine translation tasks.

3.5.1 Automated metrics

Metrics-based NLG evaluations score an NLG system by comparing its outputs to a corpus of gold-standard reference texts. The metric is a score that tells us how similar the NLG system's outputs are to the texts in the reference corpus. Examples of commonly used metrics are BLEU [202], METEOR [157], ROUGE [169] and CIDER [277]. There are also many variations on these metrics for specific languages, application domains or tasks. Most of these metrics originate from research fields where text similarity is an important evaluation property for generated texts, such as machine translation and summarization. Consequentially, most metrics will favor outputs that are very similar to the reference corpus. However, if we want to measure textual properties that are not related to document similarity, then opting for metric-based evaluation is not the right approach.

Even though automatic metrics have a number of caveats, they continue to be popular in NLG research. Metrics based evaluation is faster and less cost-intensive than evaluations involving human judgements, allowing the evaluation method to scale if the number of outputs or systems increases. This is an advantage when multiple systems need to be evaluated relatively quickly, for example in the context of a shared task. Dušek, Novikova, and Rieser [90] used metrics to automatically evaluate 62 submissions to the E2E NLG challenge, a shared task on end-to-end (E2E) natural language generation (NLG) in spoken dialogue systems.

Some researchers use automated metrics as a measure of negative performance. Wang and Cho [283] published the first paper that used language model BERT for text generation. They used SELF-BLUE [293] to measure the diversity of texts generated by BERT. A higher SELF-BLUE means less diversity. Geissler et al. [107] used ROUGE-5 to detect and discard outputs that were very similar to the original training corpus in a computational creativity project.

3.5.2 Collecting human ratings

Instead of using automatic metrics, we can ask human judges to rate outputs on a set of properties. The advantage of this approach is that humans can evaluate texts using experience, grounded knowledge and logical thinking, something that automatic metrics cannot. Discrete ratings, such as 5-point or 7-point Likert-scales, are common. We can ask judges for absolute and relative judgements. In the first case, each output is judged independently of the others. In the second case, judges are presented with a collection of outputs, which should be ranked according to some property. The evaluation might be in the form of a survey. If the evaluation takes place online, the research participants are often recruited from crowd-source platforms such as Appen⁸ or Amazon Mechanical Turk.

⁸Crowdsourcing platform Crowdfunder was renamed to Figure Eight in 2018, and acquired by Appen in 2019.

A slightly more involved approach for collecting human judgements is embedding the generated texts in the context for which they are generated, instead of presenting them in isolation. In this approach, participants are asked to interact with the NLG system to perform a task in (a laboratory version of) a real-world context. Afterwards, we ask the research participants questions about the artifact and the context. However, this approach has some additional challenges. Not all domains are suitable for this evaluation method, as not all real-world contexts can easily be transferred to a research setting, such as NLG for the medical domain.

The properties on which outputs are evaluated depend on the type of output, and the intended application of the NLG system under evaluation. As most NLG research focuses on informative NLG, typical evaluation properties are grammaticality, language quality, correctness, groundedness, fluency, usefulness and clarity. For creative language generation, typical properties are linked to enjoyment and aesthetics, such as creativity, humor, surprise, unexpectedness and the probability that the output was written by a human instead of by a computer.

Human judgements are subjective, even more so if the properties themselves are hard to define. Operationalizing properties like creativity and humor remains difficult, but researchers from the same research field tend to use the same established definition. For example, Boden's breakdown of creativity [41, p. 1] is commonly used in computational creativity research. According to Boden, creativity is the ability to come up with ideas or artefacts that are new, surprising and valuable. Subsequently, the properties 'novelty', 'surprise' and 'value' are often used as evaluation properties in computational creativity evaluations. One of the challenges of evaluating with human ratings is that, even when researchers provide a clear description of the evaluation properties, research participants might differ in their interpretations of core concepts, which can lead to large variations in ratings between participants.

3.5.3 Measuring task-based performance in a real-world context

Finally, the most involved but more robust approach to NLG evaluation is to put the generator in a real-world context and let research participants perform the task for which the generator was built. Instead of asking participants their opinion about the system, we measure the effect of the NLG system on the actual performance of the task.

This type of evaluation is the most rigorous, but also the most complex in terms of experimental design. Is the intended effect of the NLG system, such as better decision making, measureable? How do we measure the intended effect of the system? What can we use as a baseline? Interpreting the results of such an holistic approach is also complex, as the performance is not solely determined by the NLG system and the efficacy of the underlying techniques. Properties of the setup that are independent of the NLG system, such as the quality of the user

interface, might also influence the performance of the task by the user.

In some cases, ethical considerations prevent us from doing real-world task-based evaluations, because the mistakes in the NLG system might harm or pose a risk to research participants. In these cases, researchers must take measures to prevent this from happening. In some domains, such as crisis management or medicine, this evaluation approach might not be possible at all.

3.6 Conclusion

Natural language generation is the research field that focuses on transforming data into natural language text. As we saw in Chapter 2, existing text generators for games use templates or formal grammars for text generation. In this chapter, we described templates, grammars, and GPT-2, a neural language model built on the Transformer architecture. Later chapters will feature flavor text generators based on both templates and GPT-2. We also saw that GPT-2 can be fine-tuned on additional data, to learn specific tasks or adopt to a specific domain. In Chapter 8, we will present multiple video game text corpora that can be used for natural language generation in the video games domain. This chapter also provided an overview of story generation, which shares some characteristics with quest generation for video games. In the next chapter, we will study story generators in the wild to gather additional inspiration for procedural text generation for video games.

Part II

Flavor text generation

Low-cost story generation in the wild

In this chapter, we start investigating potential approaches for generating coherent flavor text by looking at text generators for creating works of fiction. We take this somewhat round-about approach as a way to find low-cost generation methods that are usable in practice by non-experts in computational linguistics. We analyze the code and outputs of 60 text generators built for online programming challenge NaNoGenMo 2018 which can generate works of fiction of at least 50,000 words. Because these methods are typically not documented in research literature, they might have escaped notice of the research community.

The main contributions of this chapter are:

- A qualitative analysis of the code, approach and output of 60 text generators for generating fiction
- A description of four common design patterns used in these generators: high level structure definition, using modular building blocks, emergent narrative and evoking coherence.
- An analysis of the suitability of these design patterns for coherent flavor text generation for video games

This chapter is based on the following peer-reviewed publication:

- Judith van Stegeren and Mariët Theune. “Narrative Generation in the Wild: Methods from NaNoGenMo”. In: *Proceedings of the Second Workshop on Storytelling*. Florence, Italy: Association for Computational Linguistics, Aug. 2019, pp. 65–74. URL: <https://www.aclweb.org/anthology/W19-3407>

4.1 Introduction

Coherence is generally considered to be a property of a good story. For a story to be coherent, “all the parts of the story must be structured so that the entire sequence of events is interrelated in a meaningful way” [236, p. 960]. For generated stories, in particular those generated using neural models, coherence tends to decrease rapidly as the output length increases. For this reason, generating long stories is still a challenge [148].

To gain more insight in how generation of long stories is done ‘in the wild’, we review a collection of story generation projects that were created as part of the online challenge NaNoGenMo.

NaNoGenMo, or National Novel Generation Month, is a yearly online event¹ that challenges participants to create a novel using text generation. Participants have one month (November) to develop a text generator and use it to procedurally generate the text of their novel. GitHub is used to register for the event and share participants’ progress throughout the month. To qualify as a NaNoGenMo winner, participants have to share their code and a generated text of at least 50,000 words.

Since NaNoGenMo takes place online, we can use it to study practical approaches to text generation and story generation. Participants do not necessarily use state-of-the-art techniques from story generation research. Instead, the NaNoGenMo entries offer us a look into practical novel generation methods used in a (mostly) non-academic context. NaNoGenMo provides an accessible repository of story generation projects (including both code and output) that is incomparable to any academic generation challenge in terms of diversity and scale. What makes NaNoGenMo extra interesting is that it focuses on the generation of texts with a much longer length than addressed in most scientific research.

We analysed the work of participants in NaNoGenMo 2018, as this was the most recent edition of NaNoGenMo at the time of writing. We start with categorising the projects by their output type, focusing on projects that generate text with a novel-like structure. We then list the main methods for text generation used by participants in Section 4.4, since text generation methods influence the coherence of the output text. In Section 4.5, we discuss projects that generate text with a coherent narrative structure. We list the different approaches that were used to achieve this narrative structure, and link them to scientific literature. Finally, we provide some recommendations on when to use which approach.

¹<https://www.github.com/nanogenmo>

4.2 Related work

In addition to scientific literature about story generation, which we discussed in Section 3.4, we now describe work related to NaNoGenMo and measuring and ensuring narrative coherence of generated texts.

4.2.1 NaNoGenMo

NaNoGenMo was invented in 2013 by Darius Kazemi. His inspiration was NaNoWriMo, or National Novel Writing Month, an online event in November where participants are challenged to write a 50,000 word novel in 30 days.

The first attempt to create a survey of text generation methods used by NaNoGenMo participants was a blog post in Russian by Shevchenko [239]. The author discussed projects of NaNoGenMo 2013–2015, and categorised them by generation technique, such as Markov chains, recycling existing works of fiction, simulation, high-level plot generation, and neural networks. Inspired by this blog post, the NaNoGenMo community conducted their own survey² of methods (2016) and programming languages (2014–2017) as part of the event.

There is some cross-pollination between the NaNoGenMo community and academia. Participants sometimes refer to research articles, either for their own projects or to help other participants. Additionally, NaNoGenMo has been mentioned in scientific literature in fields that have a close connection to the goal of the event: procedural generation for games [145], story generation [188, 134] and computational creativity [184, 75, 70].

Cook and Colton [75] discuss the NaNoGenMo community in detail in their paper on online communities in computational creativity. Although they review some of the projects from NaNoGenMo 2016, the focus of their article was not the methods or quality of the projects, but rather the community of NaNoGenMo itself. Montfort [188] developed a novel generator called World Clock, as entry for NaNoGenMo 2013. Interestingly, most of the researchers citing NaNoGenMo have participated themselves in the past.

4.2.2 Narrative coherence

The generation of long stories, such as the 50,000 word novels of NaNoGenMo, places strong demands on coherence: the set of events in the story need to be linked, and preferably also fit into some overarching dramatic structure.

One way of achieving coherence in generated stories is by imposing a specific structure on the output text. Researchers have investigated the structure inherent in existing stories to find out how humans do this. Propp’s model of the structure of Russian folktales has been used in various story generation systems [108].

²The surveys can be found by searching for issues labeled ‘admin’ in the GitHub repositories for those respective years.

Alternative narrative structures that have been used to guide story generation are Booker's seven basic plots [126], the Hero's journey or Monomyth [99] and the Fool's journey from tarot cards [257].

In neural text generation, it is less easy to impose a narrative structure on the generated texts – unless the task is split into two steps, planning and realisation, such as in the work of Yao et al. [289]. Another way to improve the global coherence of texts generated with recurring neural networks was proposed by Holtzman et al. [131], who used a set of discriminative models that encode various aspects of proper writing.

Another way of achieving coherence is through emergent narrative [11]. This is a type of narrative (at the fabula level) that emerges from simulating simple behaviours that, when interacting, create a complex whole. The simulation gives rise to a sequence of causally linked events which give coherence to the story. The coherence in emergent narrative tends to be mostly local in nature: although the events are linked through their immediate causes and consequences, it is difficult to impose a global dramatic arc on them. Examples of generation systems that use the emergent narrative approach are FearNot! [12], the Virtual Storyteller [262] and the simulation framework from Talk of the Town [225].

Simulation-based narratives are particularly suitable for game-based story generation, since games often already have a world-state, characters, objects and a set of rules that describe valid changes to the game state. The rule system of role-playing game Dungeons & Dragons [124] is the most well-known of its kind. Various story and quest generation systems [180, 263, 155] have been built upon this and other related rule systems.

4.3 Data

NaNoGenMo uses GitHub's built-in issue tracker to keep track of all user submissions. Every issue corresponds to one NaNoGenMo project. In the issue thread, participants can post comments, interact with other users, share their development process and publish previews of the generated novels.

We downloaded all issues from the NaNoGenMo 2018 repository³ as JSON data using the GitHub API. We took issues into account that were opened between the start of NaNoGenMo 2018 and March 2019, that were labeled as 'completed' and not labeled as 'admin'. The label 'completed' means that both the generator code and a 50,000 word output are publicly available. All 61 issues⁴ were manually reviewed, by looking at the programming code, the output, the tools and used datasets. For practical reasons, we ignored projects in other languages than English.

³<https://github.com/NaNoGenMo/2018>

⁴Throughout this chapter we will reference each project by its issue number on GitHub. The details of each project can be found on the corresponding issue page on GitHub, i.e. <https://github.com/NaNoGenMo/2018/issues/{issuenumber}>.

We have made the data⁵ for our analysis in this chapter available online. We also communicated⁶ this research to the participants of NaNoGenMo 2018 in the official Github project, and asked for their feedback on the collected dataset prior to publication of the related research paper. We did not receive comments or feedback from participants on the published dataset. A few participants reacted positively when our research paper was published in 2019.

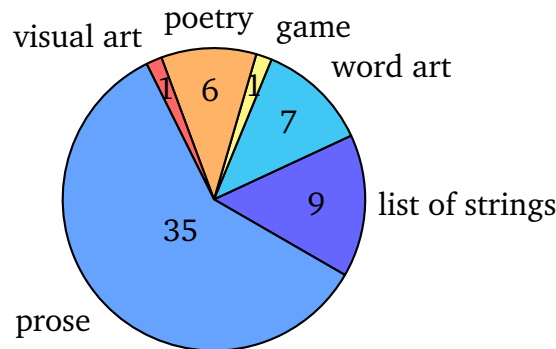


Figure 4.1: Output type of completed NaNoGenMo 2018 projects.

NaNoGenMo uses a loose definition of ‘novel’: any text of more than 50,000 words qualifies as an acceptable output. There are no rules dictating the format, style, grammaticality, subject or content of the text. As a result, the outputs vary greatly from one another. See Figure 4.1 for a categorisation of NaNoGenMo projects according to their output type. Most projects generate a novel-like text, with a form that resembles sentences, paragraphs and chapters. One participant (project 72) created a generator for an Interactive Fiction game. Other projects generated word art, e.g. repetitions of one word, ASCII art or text without meaning, poems, graphs or lists. In the rest of this chapter, we will limit our discussion to the 35 projects that generate novel-like text.

For an overview of the programming languages used in the projects, see Figure 4.2. Some projects used multiple languages. The availability of good NLP and NLG resources in a particular language has probably contributed to people choosing those languages. Consequently, the choice for a particular programming language may have influenced the chosen text generation and narrative generation approach, and vice versa. Both Python and Javascript, the two most popular programming languages with NaNoGenMo participants, have accessible libraries for text processing and text generation. Participants that programmed in Python mainly used Markovify, SpaCy and NLTK; Javascript projects used mostly Tracery [70], a Javascript library for text generation with context-free grammars. The developers of Tracery specifically mention the NaNoGenMo community as the target audience for Tracery, which could explain the wide adoption of Trac-

⁵<https://github.com/jd7h/narrative-gen-nanogenmo18>

⁶<https://github.com/NaNoGenMo/2018/issues/2#issuecomment-496199766>

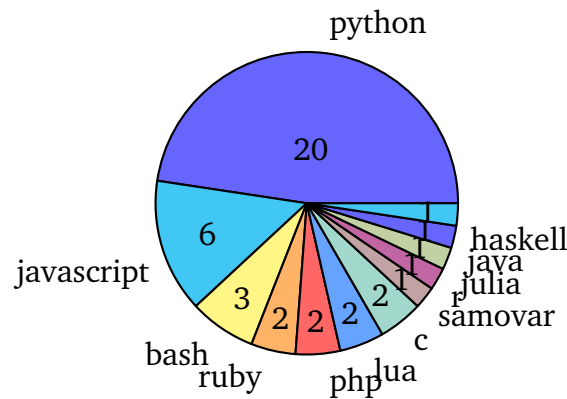


Figure 4.2: Programming languages used in NaNoGenMo projects. Projects that use more than one language are counted multiple times. We counted Bash as a language in the instances where a Bash-script was used to generate part of the output text.

ery within the NaNoGenMo community, as well as the large number of projects in Javascript, a programming language that is not typically used for text generation or text processing.

In addition to NLP libraries and tools, most participants use externally sourced text data. Public domain books from Project Gutenberg⁷ and The Internet Archive⁸ were very popular with NaNoGenMo participants, as was Darius Kazemi’s Corpora⁹ repository, which is a collection of word lists organized by subject, such as games, medicine and religion. Some participants created their own corpora from online resources, such as databases of subtitles, marathon reports, horror stories and reports of personal experiences with psycho-active drugs.

4.4 Text generation methods

The 35 novel generation projects of NaNoGenMo 2018 use a variety of text generation methods to create the surface text of their novel. In this section, we provide a survey of the various approaches we have seen.

4.4.1 Templating

More than 10 projects use some form of *templating*. Libraries such as Tracery offer a fast way to implement this in Javascript and Python. Most text templates were hard-coded in the generator, which is time-consuming and requires manual effort. An alternative approach used in some projects (projects 64, 101 and 104)

⁷www.gutenberg.org

⁸www.archive.org

⁹<https://github.com/dariusk/corpora>

was to create templates automatically, e.g. by running all sentences from a corpus through a part-of-speech (POS) tagger and creating sentence templates from the POS-tags.

The popularity of templating is not surprising, as templates offer a strong form of control over the surface text. However, using templates does not guarantee a good quality output. If templates are filled with randomly chosen phrases, as was done in some projects, the quality of the generated text may be worse than that of a text generated with Markov chains (discussed next).

4.4.2 Markov chains

At least 8 projects used *Markov chains* for text generation. Markov chains are statistical language models, which can be created fully automatically from corpora. They can be used for text generation by choosing a start token and using the probabilities in the model to choose the next token. Using Markov chains is an accessible approach to text generation, as it does not require coding the content of the output. Markovify¹⁰, a Python library for working with Markov chains, was used by the majority of users that used Markov chains for generation. We believe that Markovify has contributed to the popularity of the Markov chain approach under NaNoGenMo participants.

Not your average ultra (project 89) creatively mixes the outputs of two Markov chains. One Markov chain was trained on a collection of marathon reports, the other on a dataset of reports of personal experiences with psychoactive drugs. As the generator produced more text, the influence of the second Markov chain on the generator grew stronger, which resulted in output in the form of a race journal that becomes progressively delirious over time.

Although the outputs from a Markov chain are often less coherent than those produced by templates, the advantage of Markov chains is that they often yield surprising or interesting results. For participants that value creativity over coherence, Markov chains are a suitable technique for text generation. As we will see in Section 4.5, the resulting lack of coherence is not always a problem.

4.4.3 Remixing

Remixing external sources, such as text from existing novels, was also a popular approach with participants. More than half of the projects use some form of remixing to create their output. One example of remixing is creating a new text by taking a source text and substituting words from the text according to specific rules. A hilarious example of this is *Textillating* (project 96), where Dickens' *Great Expectations* is 'improved' by increasing the number of exclamation marks and substituting each adjective in the text with its most extreme synonym.

¹⁰<https://github.com/jsvine/markovify>

Some participants collected external sources and composed their novel by cutting-and-pasting sentences from these. For example, *Doctor, doctor!* (project 86) used the corpus of Yahoo! health questions and answers to generate a dialogue between a doctor and a patient. Another participant scraped sentences from GoogleBooks about a set of topic words, and created an original text by cutting-and-pasting snippets from Google Books preview files. In some cases, remixing was paired with statistical modeling. The author of *Angela's claustrum* (project 28) transformed a manually-written NaNoWriMo novel draft into an outline and remixed this into a new novel by using a stochastic model of Gutenberg texts.

With this category of methods, either the output text is very similar to the source text (and similarly coherent), or the output is completely new but loses some coherence in the process, often because developers chose to introduce random words into existing sentences in their word substitution.

4.4.4 Machine Translation

There were various generators that used *machine translation* techniques for creating a novel. Project 22 created a new text by mapping every sentence of *Northanger Abbey* by Jane Austen to a sentence written by Sir Arthur Conan Doyle, using sentence embeddings.

Project 61 used machine translation to transform the text of one of L. Frank Baum's Oz books. All dialogue from the book was translated to the "language" of *The Muppets' Swedish Chef*, and all other text was translated to Valleyspeak.¹¹

One participant (project 33) used a public domain movie as the basis for their novel. They turned the movie into a collection of screenshots and fed this to Microsoft Cognitive services to generate captions for the screenshots. The captions were then transformed into novel text. This can be seen as a form of machine translation. Instead of translating between different languages, this project translates between different modalities (video to image, image to text).

Machine translation within NaNoGenMo can be seen as a form of remixing, and the drawbacks are indeed very similar. Either the output text shows a strong resemblance to the original text, or it is more creative but ends up incoherent.

4.4.5 Deep learning

Finally, there were three projects that used deep learning to create their novel. Two projects, project 73 and project 76, used Torch¹² to create an LSTM architecture trained on an external dataset. Project 73 trained the LSTM on a crowd-sourced collection¹³ of *Dungeons & Dragons* character biographies, and project

¹¹Valleyspeak is an American social dialect that originates from the San Fernando Valley in Southern California.

¹²<http://torch.ch/>

¹³https://github.com/janelleshane/DnD_bios

76 used user-written horror stories scraped from CreepyPasta.¹⁴ Both projects have output that is neither coherent nor grammatical. However, the LSTM does manage to convey the typical writing style of RPG biographies and horror stories. Finally, project 99 used machine learning to investigate whether a neural network trained on the text of Moby Dick could successfully reconstruct the original text, by predicting the sequence of sentences.

4.5 Methods for narrative coherence

NaNoGenMo output is at least 50,000 words, or roughly 75 pages of text. This is a much greater length than is usually produced by story generation systems. In computational creativity and creative NLG, typical outputs range from tweets (140-280 characters) to stories of one or two pages, with exceptions such as Curveship [186], UNIVERSE [159] and World clock [188].

To see how NaNoGenMo participants generate coherent novel-length narratives, we performed an informal qualitative analysis of the outputs of the 35 text generation projects, specifically focusing on coherence and the presence of narrative structure. Out of the 61 projects of NaNoGenMo, only 14 projects had a narrative structure, that is, they exhibited coherence as discussed in Section 4.2.2. Below we give an overview of the four approaches used to achieve this.

4.5.1 High-level specification

Some projects achieve coherence by hard-coding a narrative structure in their input. *The League of Extraordinarily Dull Gentlemen* (project 6) defines that narrative structure in a specification written in Samovar, a PROLOG-like domain-specific language for world-modeling using propositions. The specification is a high-level description of the story, with its representation level a mix of a fabula and surface text: it is not just a sequence of events, but also includes dialogue and narrative exposition. The surface text for its output was generated by running the specification through Samovar's assertion-retraction engine¹⁵, taking the resulting sequence of events and realising those into sentences with a Python script. This approach is similar to that of other story generation systems that use logic programming to generate stories or fabulas, such as Ceptre by Martens [180], Robertson and Young [218]'s General Mediation Engine, and the work of García-Ortega et al. [99].

Hard-coding a narrative arc in a specification can be seen as high-level templating. It also has similar advantages as templating: because the author specifies the narrative arc by hand, they have tight control over the surface text, which results in an output that looks like it was written by a human. However, this approach places an authorial burden on the developer of the generator. The story

¹⁴<https://www.creepypasta.com/>

¹⁵<https://catseye.tc/article/Languages.md#samovar>

of project 6 of 50,000 words was generated in 930 lines of Samovar. We expect that the effort of writing such a specification could be reduced with code generation. Another disadvantage is that one story specification defines exactly one surface text. The surface text of project 6 includes little variation. The book consists of scenes where multiple characters perform the same action in sequence. Repeating patterns are clearly visible in the output text, making for a dull read – hence the title of the project. However, the output of project 6 sets itself apart from other generated novels by having grammatical surface text and maintaining a clear traditional narrative arc throughout the entire story with a beginning, an incident, a climax and a problem resolution.

For authors that want to generate a story out of a high-level story description, using a domain specific language like Samovar might be a suitable solution. The code for this NaNoGenMo project is very readable and could serve as an introduction to this approach. As this approach requires the user to write part of the story, it is less suitable for projects where the author also wants the generator to create the contents of the fabula, or requires a lower cost in terms of writing the code and specification.

4.5.2 Hard-coded narrative elements

Instead of hard-coding the narrative structure of the entire story in the generator, it can be hard-coded only in specific places. An example of this approach from outside NaNoGenMo is described by Reed [210], who used a grammar to generate ‘satellite sentences’ that can be inserted in a larger human-authored narrative for an interactive fiction game. Satellite sentences are sentences that moderate pacing and reestablish context within dialogue scenes [210], such as “She coughed”, “The clock was ticking” and “It was getting late”.

There were several NaNoGenMo projects where the generated text itself had no structure at all, but where the developer still created a narrative by providing a hard-coded section at the beginning and/or ending of the book. Having a fixed beginning and ending can tie otherwise incoherent pieces of generated text together, as it gives readers a context in which they can interpret the generated text. Even text generation techniques that normally do not lead to coherent output, such as Markov chains and random generation, can still be ‘saved’ by using this technique.

An example is *Not your average ultra* (project 89), which successfully frames the (in)coherence of text generated by Markov chains by naming the specific setting of the novel at the beginning and end: an ultramarathon.

Similarly, *The Defeat at Procyon V* (project 83) contains 50,000 words of dialogue between a science fiction Fleet Commander and their Super Admiral. The lines of dialogue are randomly generated from a grammar of science fiction techno-babble, occasionally interspersed with exposition sentences, similar to the satellite sentences from Reed [210]. Because the beginning and ending of

the novel are fixed, the reader has a context in which to interpret the conversation: the conversation is about the various weapons and technologies that were deployed in the defense of Procyon V.

With this approach, the problem of generating a coherent narrative is transformed into writing narrative elements that frame the generated text in such a way that the reader perceives a narrative in the entire text. It is particularly useful in instances where developers prefer straight-forward text generation techniques over narrative generation techniques, and for developers that want to write as few lines of code as possible.

4.5.3 Simulation

There were various projects (projects 11, 18, 39, 60 and 100) that used simulation as the basis for their narrative. The projects with simulation-based narratives had two things in common.

Firstly, most projects used rule systems that are similar to those of well-known role-playing games. For example, *The Longest Corridor* (project 18) uses a combat system that closely resembles that of tabletop game *Dungeons & Dragons* [124]. The project generates stories about a mythical corridor filled with monsters and treasure. For each chapter of the novel, the system generates a hero who has to fight their way through the corridor. If the hero is defeated by the inhabitants, the hero's remains will stay in the corridor for later heroes to find. If the hero reaches the end of the corridor and finds the treasure, they install themselves as the new master of the corridor, waiting for new adventurers to come and challenge them. This continuity, where characters of previous chapters (old world state) can interact with characters from current chapters (current world state), is what moves this project from a straight-forward simulation into the realm of narrative. Similarly, *Of Ork, Fae, Elf and Goblin* (project 39) generates a fabula of a group of creatures that fight each other with procedurally generated weapons in different rooms.

Another roleplaying-game inspired project is *High Fantasy with Language Generator* (project 60). Instead of having one global text-level simulation that tracks the world state and governs the entire narrative, it uses multiple low-level simulations that each govern one type of event. The project follows a group of adventurers on their quest. During their travels, the characters encounter monsters, visit local taverns and play dice games with strangers. For each of these scenes, the generator uses a separate simulation.

A second property of simulation-based novels is that they often have a journal-like format. The world state of the simulation gives rise to the surface text of the story. Since the world state is updated with each clock tick, it is intuitive to let the novel's sections (chapters, paragraphs) correspond to one clock tick. Consequently, simulation-based narratives are particularly suitable for generating journals or logbooks, in which each section corresponds to one unit of time. *The*

Pilgrimage (project 11) is an example of a project that follows a journal format.

A weakness of some of the simulation-based projects in NaNoGenMo is that they generate events that are not linked to each other. An example is *Wheel of Fortune* (project 100), which simulates characters who slowly grow old and die, all the while experiencing events that are generated from randomly drawn tarot cards. The resulting sequence of events looks like a fabula. However, the events are not related to each other and do not influence each other: the characters' actions happen completely in a vacuum. This does invite the reader to imagine their own narrative, but this requires a lot of effort on part of the reader. Still, symbolism from tarot cards can be used successfully to shape a narrative when combined with other methods, such as high-level specification of narrative structure (see Section 4.5.1). A story generator from outside NaNoGenMo that also used the tropes from tarot was developed by Sullivan, Eladhari, and Cook [257]. However, Sullivan, Eladhari, and Cook used the tarot cards to generate movie-like story synopses, with a plot structure based on Booker's seven basic plots and screenwriting principles.

4.5.4 Evoking a narrative

Some of the project outputs *evoke* a narrative in the mind of the reader, even though there is no narrative structure explicitly present in the text. This can even be the case for output texts that are not grammatical. Incoherent texts that still have a recognizable novel form force the reader to guess the meaning of the author. This subjective interpretation might lead the reader to perceive the generated novel as a coherent whole.

As Veale [276] notes in his paper on tweet generation, form can be more important than content. Veale calls this effect 'charity of interpretation': if humans see a text in a well-known form (or *container*), they are disposed to attribute more meaning to the text than it actually has. NaNoGenMo participants achieved this effect in three ways.

If the text of the novel is limited to a specific subject, readers will try to fill in the gaps in the structure with their own knowledge and expectations. An example of a project that limits its topic to instill a sense of coherence is *Doctor, doctor!* (project 86). The output text has the form of a dialogue, consisting of randomly chosen questions and answers from a dataset of Yahoo! questions from the health domain. The questions and answers have no logical connection whatsoever, but the vocabulary and writing style will be recognizable to readers who are familiar with medical discussions on the internet. Even though the answers of the doctor make no sense in the context of the respective questions, readers will infer that this novel is about a dialogue between a doctor and their hypochondriac patient.

Another technique for evoking a narrative is by connecting unrelated random elements with each other to improve the perceived coherence. *Out of Nowhere*

(project 57) simulates an interaction between its characters by connecting interactions at the word level, which we explain below. Out of Nowhere produces the script for a play, based on lines of English text from public-domain phrase books. These phrase books consist of collections of sentences in two languages, for example for language learners or travelers in a foreign country. The characters represent different nationalities, and their dialogue lines are based on the text of phrase books for their respective languages. The character dialogue is generated by choosing lines from each character's phrase book. Most dialogue lines are chosen randomly, but the generator increases the coherence of the output with a few tricks. Both the location and the interactions are influenced by the words that occur in previous lines. For example, if the previous line contains the word 'waiter', the generator will include a restaurant or cafe in the scene. Similarly, if one of the previous lines contains a question mark and an interrogative word ("what", "who", etc.), the generator will assign a higher probability to lines that would constitute a logical answer. For example, if previous lines contain the phrase "Where is ...?" the generator favors sentences like "In Timbuktu" or "At my house". This approach is similar to the one used by Reed [210], in which the text generator takes different types of context into account, such as dialogue progression, location and time of day. The difference is that Reed tagged his text with locations for the satellite sentences, whereas project 57 generates all sentences and their connections on the fly. The result is a script that has similar quality as the generators that use the simulation approach, even though there is no underlying world state for this play. All the coherence comes from word-level choices.

Besides limiting the topic of a text, using the right style can increase the perceived coherence of a text as well. If a reader recognizes a particular style from a particular type of narrative, the reader might infer meaning where there is none. A project that adapts this idea in an original way is *Velvet black skies* (project 65), which uses statistical modeling to find the most cliché sentences in a corpus of science fiction writing. The developers defined clichés as "n-grams that occur in the texts of more than 30 authors." The generator creates a new text from these clichés by clustering them by topic and by remixing them into a chapter for each topic. Readers of science fiction classics will immediately recognize the particular style of vintage science fiction.

The above techniques ask something extra of the reader during the interpretation of the text. As such, they are suitable for situations where the writer wants to highlight the subjective experience of the reader in ascribing meaning to a text.

4.6 Discussion

In the previous sections, we described the most prevalent text generation methods from NaNoGenMo 2018 and their respective advantages and disadvantages. We discussed four different approaches that were used to achieve coherence (or

the semblance of it) in novel-length texts, highlighting some of the most creative projects.

This chapter described the methods for text generation used by NaNoGenMo 2018 participants, and divided these methods into four categories: hardcoding a high-level narrative, hardcoding contextualizing beginnings or endings, emergent narrative, and faking coherence.

If there is already a high-level story arc thought out for the surface text, using a high-level specification to define this story arc is a good approach. Hard-coding the high-level narrative arc in a specification can reduce the authorial burden of manually writing the full text significantly. However, the approach is not suitable for projects where the generator should generate the fabula in addition to the surface text. We saw only one project in NaNoGenMo with this approach. Although the surface text was high-quality compared to other projects, it was also repetitive and would not be mistaken for a human-written story.

If the generator is also in charge of generating the events that underlie the surface text, a simulation-based or emergent narrative approach is a good choice. Emergent narrative has been applied in various story generation systems already, most notably for the games domain, because of the overlap in functionality between simulations for narratives and rule systems for games. A weakness of simulation approaches is that, if the generated events are not interrelated, the sequence of events generated by a simulation lacks narrative coherence.

However, even text generation methods that do not create coherent text can be turned into a narrative, either by hardcoding narrative elements, such as a contextualising beginning or ending, or by evoking a narrative by exploiting readers' charity of interpretation.

4.6.1 Limitations

A limitation of this overview is that it is only a snapshot of the NaNoGenMo challenge. Since the work from this chapter was published in 2019, natural language generation techniques have developed further, including new language model architectures such as the Transformer [275]. Pre-trained language models like GPT-2 [209] and BERT [85] have gained in popularity and are currently being used for various creative generative projects. Later editions of NaNoGenMo reflect these developments.

The definition of coherence that was used for the informal analysis of contributions was fairly loose. We were only interested in obtaining inspiration for approaches for low-cost, relatively coherent text generation. However, it is clear that a more theoretically grounded and in-depth analysis of the coherence of each project would have been beneficial in assessing the generative power of the four identified approaches. It would also have been interesting to rank the contributions according to narrative quality, and treat the best performing narratives more in-depth.

This chapter considers all NaNoGenMo contributions in isolation. Comparing multiple projects with each other might enable us to draw conclusions about the efficacy of the four high-level techniques for generating coherent novels. Unfortunately, manually comparing 35 novel-length texts would be too labor-intensive. Automatic evaluation of NaNoGenMo contributions, for example on the coherence property, could be a solution to this problem. The research conducted by Scheuter [232], on scoring long fictional texts on semantic and syntactic coherence, is a first step in this direction.

4.6.2 Automatic evaluation of NaNoGenMo contributions

To automatically evaluate NaNoGenMo contributions, we could use metrics for assessing coherence in generic natural language text. As Gervás [109] mentions in his survey on story generation, evaluating and comparing story generators is intrinsically a hard problem, because of the variation in definitions of what makes up a story. Evaluating and comparing the narrative quality and coherence of stories is similarly elusive.

We could use existing metrics for assessing coherence in generic natural language text. However, NLP research focuses mostly on non-fiction text, which is typically more topically coherent than fiction. Existing metrics might therefore not be suitable for generated fiction. As far as we know, no automatic metrics for measuring coherence in fiction currently exist, although Kybartas and Verbrugge [155] have proposed a collection of metrics for assessing the quality of quests in video games. Narrative quality metrics could also be used to quickly sift through the large number of NaNoGenMo contributions, to find the most promising projects in each year's challenge. This could save time for researchers who want to continue studying NLG in the wild.

Alternatively, we could use other quantitative properties of the Github issues as a proxy for narrative quality or code quality, such as Github *stars*, i.e. favorites and bookmarks, or *forks*, i.e. a copy or modification of an existing project. This has been used in other research, including research for assessing popularity [272], research software impact [89] and communication efficacy in open-source communities [48]. However, in 2018 the NaNoGenMo community was still fairly small. Furthermore, the amount of interaction with Github projects is not solely determined by the content of a project, but also by issue activity during NaNoGenMo, number of Github followers of the project author, social media presence of the project author, and other factors [44]. Consequentially, this method would probably not yield enough information to make a representative ranking of all finished projects.

4.6.3 From novels to the video games domain

How do the techniques from this chapter translate to the video games domain? Since many video games contain stories, the four approaches for coherent narra-

tive generation from this chapter could also be applied to video game narratives instead of novels. However, video games often contain interactive narratives, whereas novels generally consist of static text. This limits the applicability of the described techniques for the video games domain. However, game narratives range from fully static to fully interactive narratives. If the video game narrative is at the ‘static’ end of this range, the methods might be usable for creating a video game narrative. Game narratives also differ in structure, compared to novels: games narratives can consist of a monolithic, single story, or they can be made up of various smaller, interrelated or independent narratives. Depending on the level of interactivity in a game, the described techniques could be used, either for generating a single, coherent narrative, such as a main quest line for an RPG, or parts of a narrative, such as a part of a quest or a piece of flavour text.

The first approach, hardcoding a narrative arc in a domain-specific language, could be suitable for games that contain a single, static narrative. We find this type of narrative in single-player games with one fixed storyline, such as *DevOps: The Line* [287] or *Wolfenstein: The New Order* [178]. If a game has an interactive story, i.e. player choices could influence the narrative, using live generation with logical programming might be too computationally expensive. Since a specification of a narrative arc is suitable for one (type of) narrative, this method does not scale well across different games or even different stories. This would suggest this method is most suitable for linear games with a largely manually authored narrative.

Hardcoding narrative elements is already common practice in video games that use procedural content generation. Rule-based procedural generators often create a specific type of assets, or assets that incorporate a central theme. Manually authored content can contextualize that theme for the player, and make them experience the generated content as more coherent. A concrete example of this is *Skyrim*’s Radiant quest system [30], which creates new side quests for the player on the fly. All sidequests fit into the manually authored backstory of *Skyrim*, which centers on a civil war between the local Nord populace and The Empire which rules *Skyrim* as a province. The civil war and resulting unrest can function as a backdrop, narrative context or even directly, as the underlying cause for the objective of generated subquests.

Simulation or emergent narrative is an approach to narrative creation that is very common in video games. Many simulation and survival games, such as *The Sims* [264], *Kerbal Space Program* [248] and *Oxygen Not Included* [149], feature no hand-written narratives but contain extensive rule systems. Examples of rule systems are systems that govern in-game physics (*Kerbal Space Program*), construction rules for combining in-game resources to create new game elements (*Kerbal Space Program*, *Oxygen Not Included*), or social relationships between in-game characters (*The Sims*, *Oxygen Not Included*). These narratives are created during gameplay from how game elements interact with each other, while following the underlying rule systems. However, most of these narratives will be

implicit in the sense that the narrative is only acted out by the game elements, instead of appearing as explicit surface text in the game. Some recent games do realise surface text for their emergent narrative, such as survival game *Rimworld* [176], which features an “AI narrator” that summarizes the emergent narrative events.

The final approach that we discussed is evoking a coherent narrative, or faking coherence by tricking the reader. This approach is also used in games, although it is less widespread than mixing procedural content with manually authored content (the first two approaches) and emergent narrative. We already discussed a few examples in Section 2.5.2. Rogue-like game *Caves of Qud* [95] uses the ‘faking coherence’ approach [119] for generating biographies for Sultan NPCs. Instead of generating the life events of the NPC first, and the biography second, the generator works backwards. The generator starts with generating a set of random historical events, and finishes by linking all these events together. Randomly generated historical events are framed in a new context to form a coherent biography. Backwards generation is also common in procedural murder mystery and information games, such as *DATA Agent* [117] and *ClueGen* [255]. In these games, the generator has to make sure it creates *solvable* murders. To enforce this generative goal, first the murder is generated, and then a plausible motive, location and character relations.

The evoking coherence approach is much broader than reverse generation to ensure solvability. The projects discussed in this chapter show that by playing with the expectations of the reader, for example by generating output with the right form, topic or style, procedurally generated game assets can suggest coherence to readers/players, sometimes at very low cost.

We encourage the PCG and NLG research communities to keep an eye on the online creative communities, such as *NaNoGenMo*, as these typically contain art and programming enthusiasts that are early adopters of new NLG techniques in the wild. Collaborating with and borrowing techniques from these communities might yield interesting new insights for all involved. Given the popularity of tools such as *Tracery* [70], *Markovify*, and *gpt-2-simple* (in later *NaNoGenMo* editions), online communities can give us insight in how we can make generative tools more usable and accessible for non-NLG-experts.

4.7 Conclusion

In this chapter we gave a high-level overview of the different approaches for generating coherent narratives from *NaNoGenMo* 2018: hardcoding a high-level narrative, hardcoding narrative elements, emergent narrative and evoking coherence. We briefly discussed example projects for each approach, and connected them to existing research literature. Finally, we discussed the applicability of each approach for the video games domain. We saw that some approaches are already used in video game development. The fourth approach, evoking a co-

herent narrative by playing with the reader's expectations, seems worthwhile to investigate further, as it is not yet widely used in game development. This approach is be a low-cost method for creating procedurally generated game assets that are *perceived* as being coherent. In the next chapter, we will build upon this method of evoking coherence when we create flavour text headlines for video games.

Generating flavor text headlines

In the previous chapter, we found four approaches for generating coherent fiction in the contributions to NaNoGenMo 2018. One of these approaches is ‘faking coherence’, which exploits the human tendency to see patterns in unconnected data. This approach seems particularly useful for generating flavor text with a low-cost approach. The faking coherence method is very promising for text generation for games, because it requires relatively little effort in terms of authoring, and few computational resources.

The NaNoGenMo projects that used this approach found a variety of ways to achieve increase perceived coherence: playing with the reader’s expectations by following a certain form, structure or style, or using specific words, themes or topics. The projects have in common that they contextualize generated text in such a way that it is perceived as coherent, even if the text is unstructured, unrelated, incoherent or random. The reader is invited to come up with a logical explanation for the contents of the text. This interpretative action might help in creating the illusion that the perceived relations in the text were introduced deliberately by the text generator (or a human writer). We can find a similar concept in the work of Tony Veale, who has investigated metaphor generation and computational humor in the computational creativity field. Veale describes how the form of generated textual outputs, and not just the content, determines how readers interpret the text.

This chapter describes an NLG prototype that uses low-cost techniques for creating coherent flavor text, namely template-based generation and text modification. To increase the perceived coherence of generated text, we use two strategies from the previous chapter. Firstly, text generated by the prototype has a familiar form: flavor text in the form of headlines. Secondly, the text generator generates headlines based on an input text. The input text acts as starting point and context for the output. The prototype reuses words from the input text, synonyms and related concepts. This should increase the perceived intertextual coherence of the input text and output text.

The main contributions of this chapter are:

- a design for a flavor text generator with a modular architecture that creates flavor text in the form of headlines
- an implementation of the flavor text generator that uses pre-existing data and open-source libraries
- an evaluation of the flavor text generator with human participants that shows that the approach for evoking coherence in the reader works in practice

This chapter is based on the following peer-reviewed publications:

- Judith van Stegeren and Mariët Theune. “Churnalist: Fictional Headline Generation for Context-appropriate Flavor Text”. In: *Proceedings of the Tenth International Conference on Computational Creativity*. ICC3 ’19. Charlotte, NC, USA: Association for Computational Creativity (ACC), 2019, pp. 65–72
- Judith van Stegeren and Mariët Theune. “Remixing Headlines for Context-Appropriate Flavor Text”. In: *Proceedings of the IEEE Conference On Games*. London, UK: IEEE, Aug. 2019

5.1 Introduction

In this chapter, we describe *Churnalist*, an interactive system for generating newspaper headlines for a given context. Our system is meant for generating fictional headlines that can be used in games. Most headline generators take a newspaper article as input and summarize it in one sentence. In contrast to these systems, *Churnalist* accepts free text as input and generates headlines based on nouns extracted from the input text. By reusing nouns from the input text in the generated headlines, we aim to make the headlines context-appropriate, by which we mean that readers will believe that the headlines are related to the input text. We want to exploit the human tendency to see connections between texts (input text and headlines) where there are none.

There are various games that use fictional news (in the form of headlines or newspaper articles) to provide narrative context to the player. For example, in city simulation game *SimCity 2000* [182], the player has access to newspaper articles with information about important city issues, disasters and new technologies. Similarly, *Cities Skylines* [68] features a fictional social media website called ‘Chirpy’, where virtual citizens of the player’s city express their (dis)satisfaction with the player’s performance as mayor and city planner. Chirpy messages are a mix of random flavor text messages that do not contain any information about the current game state, and messages that only occur if there is an

underlying problem with the player's city. In *Deus Ex: Human Revolution* [92], the player can find ebooks and newspapers that refer to the social unrest that is driving the game's main storyline. Idle game *Cookie Clicker* [267] has a news ticker with randomly generated headlines that reflect the player's progress in the game. For example, once the player has bought one or more units of 'Grandma', i.e. an upgrade that creates extra cookies, the headlines start to include messages from "grandma". If the player decides to sell one of the grandma upgrades, the ticker might respond with "News : cookie manufacturer downsizes, sells own grandmother!"

These fictional newspaper articles and headlines are an example of *flavor text*, i.e. text that is not essential to the main game narrative, but creates a feeling of immersion for the player. Immersion is especially important for role-playing games and simulation games, as it gives the impression that the virtual world the player is interacting with is a living and breathing world. In some cases, the flavor text changes depending on the game state, such as with *Chirpy* and the *CookieClicker* headlines.

Writing flavor text is a time-consuming task for game writers, and text generation can be a solution to this problem. Most games that incorporate text generation use simple templates or canned text. More complex NLG techniques rely on linguistic models, which often take considerable effort to create and require linguistic expertise. Statistical linguistic models can be created automatically from a dataset of texts. However, generators with underlying statistical models offer less fine-grained control over the output. Canned text and simple templates offer balance between control over the output and ease of use. One drawback of this approach is that players will figure out the underlying templates after playing the same game for a while, or after replaying the game [13]. We think that natural language generation techniques other than canned text and simple templates are worth investigating in the context of game development, especially data-driven approaches to text generation, as these can overcome the need for expensive, handcrafted language models. We propose a system that can generate fictional headlines in order to support game writers in the task of writing flavor text.

In Section 5.2, we will discuss related work. In Section 5.3 and 5.4 we will present *Churnalist*: we describe the system goal, the architecture and the generation steps in detail, using a running example. We then describe our evaluation experiments and discuss our results in Section 5.5.

5.2 Related work

In this section, we discuss work related to headline generation, text generation for games and generative systems that take context into account.

5.2.1 Headline generation

Headline generation for non-fictional headlines is often seen as a document summarization task, where headline generators take a full article text as input and return a headline that describes the most salient theme or the main event of the text. The literature distinguishes between extractive summarization and abstractive summarization approaches. Contrary to extractive systems, the output of an abstractive system does not have to correspond to a sentence from the input text. Possible approaches for abstractive headline generation systems are rule-based [88], statistics-based [17] and with machine learning [67, 237], with the latter winning in popularity in recent years.

Headlines [104] is an example of a headline generation system that focuses on the creative side of writing headlines. It can be used to support editors in their task of writing catchy news headlines. Given a newspaper article text as input, it extracts the most important words from the text and uses these as seed words for generating a large set of variations on well-known lines, such as movie names and song lyrics. This research is a good example of combining natural language generation with techniques from computational creativity.

5.2.2 Text generation for games

Text generation for games is a form of *procedural content generation* (PCG). Procedural content generation, which refers to the creation of content automatically through algorithmic means, is a relatively new addition to the field of artificial intelligence. PCG for games studies the algorithmic creation of *game contents*, defined by citetyannakakis2011experiencedrivenpcg as all aspects of a game that affect gameplay, excluding non-player character behaviour and the game engine. Examples of game content are maps, levels, dialogues, quests, music, objects and characters. Text generation techniques can be used for generating dialogue, stories, quests and flavor text for games. Although including generated game text in video games is winning in popularity, these texts are often generated with simple NLG techniques, such as canned text and simple templates.

Within the natural language generation field, there are various publications that list game text as a possible application [233, 256, 177]. However, there are few cases where the implemented system is actively used in a games context. One example is Caves of Qud [95], which combines techniques from procedural content generation and natural language generation to create a unique game world for every playthrough. Notably, the game generates fictional biographies for mythical non-player characters called sultans [119]. The biographies consist of fictional events, or *gospels*, from the life of the sultan, such as starting a war, acquiring a mythical weapon or forging an alliance. The game creators wanted to infuse a sense of coherence in these biographies, even though the events were randomly generated. They achieved this coherence by assigning a domain, such as ‘glass’, ‘jewels’, ‘ice’ or ‘scholarship’, to each sultan. The biography generator

incorporates domain-specific elements in each life event.

Players of Caves of Qud will interpret the randomly generated biographies as coherent narratives, thereby creating their own logical explanation for the overarching theme in each biography. The developers call this human tendency to perceive patterns ‘apophenia’. It is related to the ‘charity of interpretation’ effect studied by Veale [276], who found that “readers will generously infer the presence of meaning in texts that are well-formed and seemingly the product of an intelligent entity, even if this entity is not intelligent and the meaning not intentional.” If humans see a text in a well-known form (or *container*), they are disposed to attribute more meaning to the text than it actually contains. A similar effect is the Eliza effect described by Hofstadter [129], who noticed that humans will attribute intelligence or empathy to (text-producing) computer systems. With Churnalist, we want to exploit this effect too: by incorporating words from the input text in the output, we hope that readers will perceive the generated headlines as coherent with the input.

The biography generator of Caves of Qud picks the domain-specific elements for each gospel from a knowledge base, that links each domain to a set of words and phrases. The developers of Cave of Qud have hand-written this knowledge base, which gives them a great deal of control over the quality of the output. In this research, we used a knowledge base for a similar purpose: to link seed words to a set of related words. Instead of creating it manually, we used word embeddings as the basis for our knowledge base.

5.2.3 Computational creativity systems and context

For Churnalist, we were inspired by how computational creativity systems create text for a certain context, especially two computational creativity systems by Gonalo Oliveira.

O Poeta Artificial 2.0 [115] is a bot that tweets poems that are generated for trending hashtags on Twitter. The bot is based on PoeTryMe [116], a poem generation framework for Portuguese. According to Gonalo Oliveira, (generated) poetry should follow three rules: convey a meaningful message, follow grammatical and lexical rules, and exhibit poetic features. PoeTryMe creates poems that fulfill these requirements by filling a poem template with generated sentences. The sentences are generated by the Sentence Generator based on a set of seed words. To build sentences, the Sentence Generator uses a semantic graph of relational triplets and a generation grammar. It uses the seed words to select a subgraph of the semantic graph. Then, it selects a rule from the generation grammar that matches one of the semantic relations in the subgraph. This rule is used to generate a new sentence. PoeTryMe’s generation strategies determine which generated sentences are used to fill the poem template. The PoeTryMe framework uses external data sources to enrich the output of the system, such as a database of Portuguese poems, a semantic graph and lexical datasets. These data

sources can be changed depending on the needs of the user, which makes PoeT-ryMe suitable for different applications within the poetry generation domain.

O Poeta Artificial uses trending hashtags on Twitter as topical seed words, to generate poems that fit the hashtag. Initially, the Twitter bot had difficulty incorporating the topic, i.e. the trending hashtag, into its output. Trending hashtags are often unique words or abbreviations, which means the hashtag is not present in Poeta’s semantic graph. Consequently, the internal Sentence Generator could not create sentences that contained the hashtag. This meant that the trending topic itself was rarely included in the output, which made the link between the generated poem and the hashtag unclear to readers. To solve this problem, O Poeta Artificial 2.0 was extended with new functionalities to strengthen the connection between the poem and the hashtag, such as remixing fragments of human-written tweets and using template sentences for introducing the topic of the poem.

Another source of inspiration for Churnalist’s design was the Twitter bot TwoHeadlines¹ by Darius Kazemi, which creates new headlines by remixing the topics of two existing headlines.

5

5.3 Description of Churnalist

Churnalist is a system for generating fictional headlines that are context-appropriate for the textual input. In this section, we discuss the goal of the system and the requirements for the output. We elaborate on the technical design of the system and provide a running example.

5.3.1 System goal

We mentioned in the introduction that Churnalist is meant for generating flavor text for video games, in the form of headlines. Instead of taking newspaper article texts as input, as is common practice for headline generators, Churnalist accepts user-supplied free text as input, in the form of English sentences from a game. For example, see the one-sentence input in Figure 5.1.

“Mario must save Princess Peach from Bowser’s castle.”

Figure 5.1: An example of valid input text. The names and noun phrases that Churnalist will incorporate in the output headlines are underlined.

Churnalist extracts a set of seed words from the input and creates new headlines by doing word-substitution on headlines from a database. The seed words consist of words from the input. We expand the set of seed words by querying a vector space of word embeddings for vectors close to the words from the input.

¹Twitter bot Two Headlines, <https://twitter.com/twoheadlines>

Mario apologises to mother involved in car crash
 Mario injured after Sicily volcano triggers earthquake
 Mario says Arsenal return vs Qarabag was 'emotional'
 Mario: 'My marriage is over because I voted to leave the EU'
 Princess Peach unveils world's first Chromebook with AMD processors
 Bowser's castle retains Border-Gavaskar trophy after cleaning up Australia on day five

Figure 5.2: Example output of an early version of Churnalist, generated from the input text in Figure 5.1.

By using words that have a link with the input text, or *context words*, we generate headlines that fit the context that is represented by the input. By inserting context words in the headlines from the database, we hope to exploit the Eliza effect [129], apophenia [119] and the charity of interpretation [276] in readers: readers should think that the headlines are related to the context. Churnalist's output is a set of fictional headlines, see Figure 5.2.

Game writers can use Churnalist for flavor text generation by using prewritten core game text as input. The words from the game text will end up in the generated headlines, with which we want to evoke a feeling of coherence with the core game text. A longer example of game text as input is provided below, in Figure 5.4.

Using free text input makes Churnalist useable for different games and different topics. Regardless of the content or the type of game, as long as the input text contains content nouns and noun phrases, Churnalist will be able to extract these from the input and use them as seed words to generate headlines. Churnalist was developed using publicly available datasets, open source libraries and only simple text modification techniques, so that no linguistic expertise is required.

For Churnalist's output, we adopt similar requirements as Gonalo Oliveira [116]:

1. The output texts must look like headlines. We are not generating news article texts. The content of the headlines does not have to be realistic or ground in reality. On the contrary: we aim for fictional output, as well as output that is not literally present in the database of headlines (for copyright reasons).
2. Headlines must be grammatical.
3. Headlines must feel context-appropriate (coherent, meaningful, relevant) for the input text to a not-too-discerning, not overly critical reader.

5.3.2 Architecture

We have implemented a prototype system with the architecture shown in Figure 5.3. Churnalist has a modular design so that every subtask can be implemented or modified according to the requirements of the user, to make the system as flexible as possible.

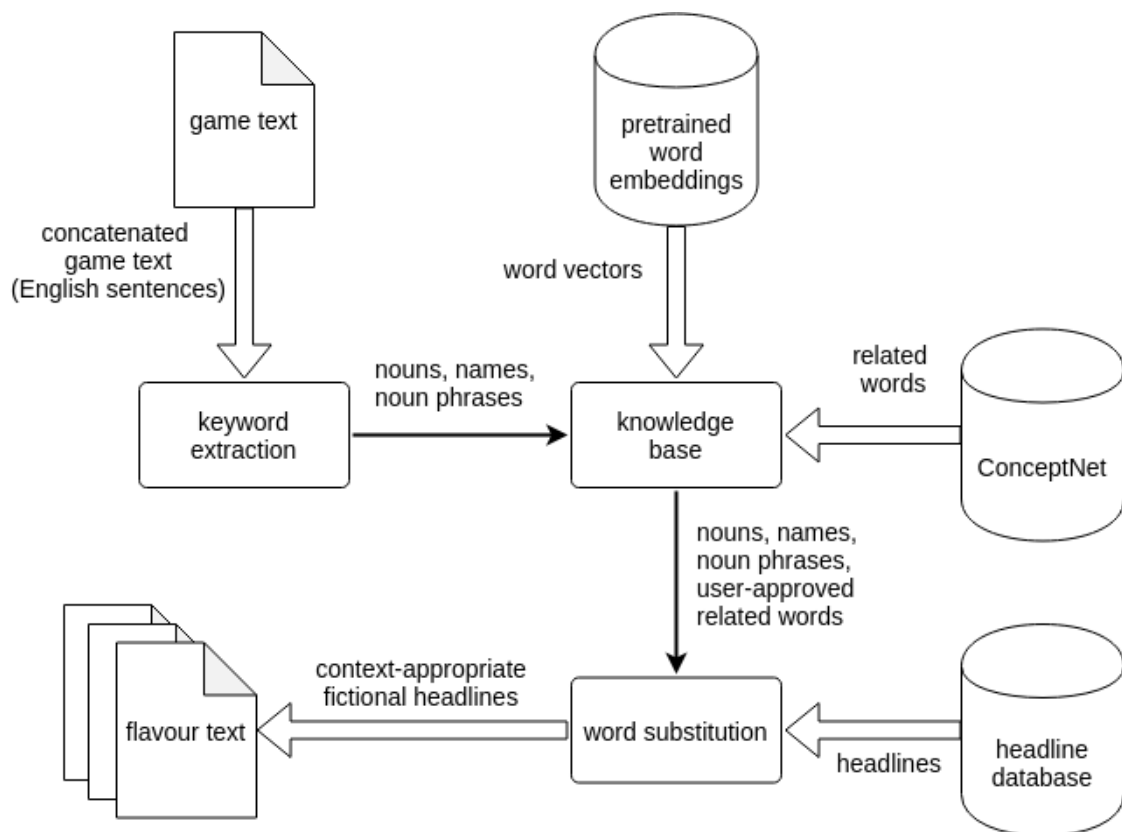


Figure 5.3: Churnalist: system architecture

Churnalist’s pipeline consists of three modules, one for every step in the generation process. At the end of every step in the generation pipeline, the user of Churnalist can filter the output of the system, thus fine-tuning the nouns and noun phrases that are used in later generation steps.

We have chosen this human-in-the-loop approach for multiple reasons. Generated outputs would have to be manually curated by a human user anyway, for two reasons. Firstly, the method of substitution is not very sophisticated and can easily lead to malformed outputs. Furthermore, Churnalist works with a corpus of headlines sourced from real-world news outlets, such as the New York Times, which might contain words or themes that are not suitable for specific audiences or inclusion in a game in general. By allowing the user to influence the generator after every processing step, the system provides insight and rapid feedback to the user about how it perceives and uses the input. Errors in intermediate outputs

“You are the system administrator of SuperSecure ltd, a hosting company. At four o’clock in the afternoon, your manager storms in. Apparently, there has been a break-in in your computer network. The CEO has been receiving anonymous emails from a hacker that demands a payment of \$100,000 before midnight. If SuperSecure does not pay, they threaten to publish sensitive company documents online. The manager is worried, since the hacker claims to possess important intellectual property.

Manager: Can you find out how the hackers got into our systems?

CSIRT: We recognize this mode of operation. We will share some relevant IOCs with your company. Can you contact us if you have finished your forensical analysis?

Security officer: There has been a nation-wide increase in phishing attacks in the past few days.

System administrator: I can’t find any traces of active malware on our Windows server. I will check the network log files for malicious activity.”

Figure 5.4: Representative input text for Churnalist: example game text for a hypothetical dilemma-based serious game, consisting of a description and a few lines of NPC text. Names and noun phrases are underlined.

can be filtered, which trickles down to later processing steps to create a larger amount of usable final outputs.

The first module, the *keyword extractor*, reads the input text and extracts the most important words. These words are the *seed words*. The second module takes the list of seed words and expands this with a set of loosely related words, gathered from the *knowledge base*. The seed words and the related words form the set of *context words*. The *substitution module* takes a random headline from a headline database, runs it through a dependency parser and substitutes parts of the sentence with context words. The resulting new headline is the output of the system. Users can generate multiple headlines from one input text; the number of possible results is determined by (1) the number of seed words in the input text, (2) the size of the headline database and (3) the size of the set of user-approved context words.

We describe these three steps in more detail and provide an example below. Figure 5.4 shows an example which features both a situational description and some lines of NPC text. The rest of this paper will feature examples that were generated with this input text.

5.3.3 Keyword extractor

The start of the pipeline is the keyword extractor. We assume that the input text consists of grammatical sentences, so that it can be parsed by a sentence tokenizer and a dependency parser.

The keyword extractor runs the input text through the NLTK sentence tokenizer² and the spaCy³ part-of-speech-tagger and dependency parser trained on spaCy's default corpus⁴ for English. It uses spaCy's noun phrase extraction to extract all English noun phrases from the input text. The keyword extractor saves all noun phrases that occur in the input text, together with the head of each noun phrase. Churnalist uses the head nouns as seed words and saves the noun phrase itself so it can be reused later, during the word substitution phase. See Figure 5.5 for an example of seed words and the corresponding noun phrases as extracted from the input text in Figure 5.4.

5.3.4 Knowledge base

In order to get more variety in our output, and not limit the words used in our output to nouns extracted from the input text, we extend the list of seed words with related words. Note that we mean 'semantically related' in a broader sense than 'similar semantics' or even 'synonyms'. Semantically similar means that two words are close in meaning, or that they can be used in the same semantic context. Semantic relatedness indicated any type of relationship between words [15]. The word 'car' might be *semantically similar* to 'bus' and 'bike', but is *semantically related* to the words 'road', 'driving', 'steering wheel' and 'mechanic', all of which are not semantically similar to 'car'.

To obtain related words, Churnalist queries the knowledge base for words similar to the seed words. We took this idea from the procedurally generated biographies in Caves of Qud [119], which evoked a feeling of coherence because of the related domain words that were put into biography. For example, for the seed word 'ice' the words 'lightblue', 'frost', 'cold' or 'winter' would all be suitable related words. The word lists for the various domains in Caves of Qud were written manually by the game developers. However, we do not want to build large content-models by hand. Instead, we want to focus on generating text from data that can be obtained automatically. We explored two data sources as knowledge base: pretrained word embeddings (FastText) and a knowledge graph (ConceptNet). The advantages and disadvantages of each option are discussed below.

The set of seed words together with the set of related words from the knowledge base forms the set of *context words*. Figure 5.7 shows the final set of context words for the seed words from Figure 5.5.

Word embeddings as knowledge base

In the first version of Churnalist, we used the English dataset of FastText's pretrained word embeddings [42] as a knowledge base, similar to the external se-

²NLTK 3.3, <https://www.nltk.org>

³spaCy 2.0.16, <https://www.spacy.io>

⁴Language model en_core_web_sm 2.0.0

Head noun	Noun phrase
administrator	system administrator
company	hosting company, company
network	computer network
CEO	CEO
emails	anonymous emails
documents	sensitive company documents
manager	manager
property	important intellectual property
hackers	hackers
IOCs	relevant IOCs
analysis	forensical analysis
officer	Security officer
traces	traces
malware	active malware
server	Windows server
files	network log files
activity	malicious activity

Figure 5.5: Noun phrases and their head noun extracted from the input text from Figure 5.4. Some phrases have a high probability of creating malformed outputs, for example because they are unsuitable for forming a headline's object or subject. These phrases, e.g. 'days', 'o'clock' and 'afternoon', were filtered out manually.

mantic datasets used by the PoeTryMe framework [116].

Word embeddings are a method for encoding words as their context, based on a corpus. The FastText dataset is based on a corpus of Wikipedia articles. It contains words represented as vectors that encode the context of these words. Words (vectors) that are close to each other in the resulting vector space, are words that occur in similar contexts.

The FastText dataset contains word embeddings that encode subword information: the vector of a word is created from the vectors of its subwords of length n . As a consequence, the dataset can be used to obtain vectors for out-of-vocabulary words: we only need to create a vector for them by looking at the vectors of their subwords. This allows us to deal with words that are not present in the semantic resources being used. Consequently, we bypass a problem similar to O Poeta Artificial's out-of-vocabulary hashtags [115].

The knowledge base tries to assign a vector to each extracted seedword and find its closest neighbours. If the seed word is an out-of-vocabulary word, the system calculates a new vector for the word based on the word embeddings of its subwords, and uses this new vector to find related words. The user can set a minimum distance for suggestions from the knowledge base and select which suggested words should be passed on to the substitution step. For an example of

Word	Distance
companiess	0.7817
subsidiary	0.6847
telecompany	0.6821
companywide	0.6773
ecompany	0.6668
webcompany	0.6496
corporation	0.6315
firm	0.6218

Figure 5.6: Suggested similar words from the FastText knowledge base for seed word ‘company’, and their distance to the seed word in descending order. Not all suggestions by the knowledge base are shown.

the results of the knowledge base, see Figure 5.6.

Since FastText was trained on Wikipedia dumps, there are words in the model that are unsuitable for inclusion in Churnalist’s output, such as words with crowd-sourced typographical errors. For example, the words closest to ‘company’ are ‘companiess’, ‘companythe’ and ‘companyx’, which result from typographical errors (and possibly pre-processing errors) in the Wikipedia dataset. Additionally, some words are very similar to one of the seed words but have no connection to the way that seed word is used in the input text. Take the compound noun ‘security officer’, which means someone who defines and enforces the information security policy in a company. Its head noun is ‘officer’, for which FastText will list ‘sergeant’, ‘quartermaster’ and ‘sublieutenant’ as related words. However, these words have little connection with the term ‘security officer’ and should not be used in the output. The user of Churnalist should filter the suggestions for related words from the knowledge base.

The FastText word embeddings had a few caveats, the most important being that the vector space contained many misspellings of common words, which would then come up as ‘closely related’ to seed words in the input text. As a result, the user of Churnalist would have to manually exclude all these misspellings from the list of context words, or use a spellchecker to filter out misspelled word automatically. Furthermore, the pre-trained FastText word embeddings are quite large, a few gigabytes per language, which makes Churnalist harder to deploy.

Knowledge graph as knowledge base

In the second version of Churnalist, we used ConceptNet [246] as source of related words. ConceptNet is a multilingual knowledge graph that represents relations between words and phrases. ConceptNet was originally created by Liu and Singh [172] as a parsed representation of Open Mind Common Sense, a crowd-sourced knowledge project. ConceptNet is a graph that represents rela-

Seed word	Approved suggestions	Rejected suggestions
administrator	-	administrators, administrator
company	subsidiary, webcompany	companies, companythe, companynew
CEO	executive, shareholder, entrepreneur	CFO, chairman, COO, CTO
network	-	networky, networkx, networknbc
emails	-	emailings, voicemail, emailers
documents	documentation, memos, archives	documentations, document, documents
manager	teammanager	managership, imanager, managerin
property	-	poperty, propert, propertyless
hackers	hacktivists, cybercriminals, scammers	hackings, blizzhackers, hackery
IOCs	-	ligtvoet, zeijst, lennaert
analysis	-	analyses, analyseses, analysist
officer	-	underofficer, officerer, commander
malware	spamware, botnet, vulnerabilities	spyware, malwarebytes, antivirus
server	-	iserver, vserver, pvserver
files	folders, fileserver	fileset, filesmy, filespace
activity	-	activitiy, activism, activin, reactivities

Figure 5.7: Seed words and examples of knowledge base suggestions for related words. Not all words suggested by the knowledge base are shown. The results were approved and rejected manually by the author. Words in the ‘approved’ column are added to the set of context words.

tions between words and phrases in various languages. The words and phrases form the nodes of the graphs, whereas the labelled and weighted edges represent assertions. Examples of assertions are *IsA* (a hacker is a person), *PartOf* (a wheel is part of a car), *UsedFor* (a keyboard is used for typing). Figure 5.8 shows ConceptNet’s knowledge about the word ‘hacker’. The graph contains translations (hakkeri), synonyms (black hat), hypernyms (person) and hyponyms (bot herder), and related words (dox). Besides relations between words, ConceptNet also represents links between knowledge resources. Besides its own knowledge

en hacker

An English term in ConceptNet 5.8

Sources: Open Mind Common Sense contributors, DBPedia 2015, JMDict 1.07, OpenCyc 2012, English Wiktionary, French Wiktionary, and Open Multilingual WordNet
View this term in the API

Documentation



Figure 5.8: ConceptNet's browsable interface (conceptnet.io) shows facts about the English word “hacker”.

about words, it might contain links to related resources, such as pages on WordNet, Wiktionary, OpenCyc, and DBPedia.

In contrast to other lexical resources, such as Cyc, DBPedia, and The Google Knowledge Graph, ConceptNet does not focus on named entities, but on the common-sense meaning of words. ConceptNet is free and accessible. In contrast to the FastText word embeddings, ConceptNet can be accessed online. A simple HTTP request to the ConceptNet API suffices to obtain a list of related words. This is also its weakness: using ConceptNet requires an Internet connection, so the prototype no longer works in an offline environment.

In the research paper describing ConceptNet 5.5 [246], the authors show that ConceptNet can also be combined with distributional semantics to create a new semantic space: ConceptNet Numberbatch. This hybrid semantic space performs better than any existing systems on word relatedness. However, Churnalist uses ConceptNet as a standalone resource.

Depending on the situation, either FastText or ConceptNet is the appropriate choice as knowledge base for Churnalist. FastText requires more storage space, but once downloaded, the data is local. Loading the model and computing the distance between a seed word and words in the vector space can be computationally costly. ConceptNet does not take up any storage space, but requires an internet connection. Both resources support multiple languages. ConceptNet can switch between languages instantaneously, whereas FastText would require downloading an additional model for every new language.

5.3.5 Substitution module

The substitution module receives the list of context words from the knowledge base module and produces new headlines that contain one or more context words. It creates new headlines by substituting the subject of an existing headline from the headline database. This approach is similar to that of Headlines [104], which inserts keywords from a newspaper article into existing sentences.

The substitution module starts by picking a random headline from the headline database. This headline is used as the starting point for one new headline. The headline is run through spaCy's part-of-speech-tagger and dependency parser, trained on spaCy's default English corpus. From the information of the parser, Churnalist tries to find the subject of the sentence. This is the substitution target. If the parser cannot determine what the subject of the sentence is, a different headline is drawn randomly from the headline database.

Next, Churnalist chooses a random seed word. Each seed word has a set of context words associated with it: the seed word itself, noun phrases from the input, and the user-approved related words from the knowledge base. Churnalist randomly chooses one of these as a substitution candidate. If the substitution target is of a different number than the substitution candidate, Churnalist converts the candidate to the right number (singular to plural or vice versa). Finally, the target is substituted by the candidate and the new headline is presented to the user.

Headlines from News API

In the first version of Churnalist, the external dataset of headlines consisted of headlines scraped with the API from News API.⁵ This API returns headlines and article excerpts from several large news websites. We collected 3629 headlines from media from the UK and the US in December 2018 and January 2019.

In practice, the headlines scraped from NewsAPI are suboptimal as source data for word substitution. They are not formatted consistently, and might contain special characters, which makes them difficult to parse with SpaCy. We also noticed that many articles have 'clickbait' titles that do not contain many content words, which means we do not have many content words to use in word substitution. And many of the scraped headlines contain references to the publication in which they originally appeared. We do not want to generate fictional headlines with references to real-world publications.

Headlines from GigaWord

To mitigate the issues with our scraped headlines, the second version of Churnalist uses the headlines present in the annotated Gigaword corpus [194]. Anno-

⁵News API, <https://newsapi.org>

tated Gigaword consists of English Gigaword⁶, extended with tokenized and segmented sentences, Treebank-style constituent parse trees, syntactic dependency trees, named entities, and in-document coreference chains. We extracted the headlines, without the annotations. Since Gigaword is sourced from international newswire organisations, such as Bloomberg and the New York Times, the quality of the data is much higher than headlines from News API. The size of the corpus is considerably larger as well. Even if we filter out all headlines with non-alphanumeric characters, to make dependency parsing as easy as possible for spaCy, we are left with over 1.5 million headlines.

5.4 Text generation with Churnalist

5.4.1 Outputs

Figure 5.9 shows examples of generated headlines, together with the original headline and seed word.

According to the requirements from Section 5.3.1, generated headlines should have an appropriate form, should be grammatical and should be context-appropriate for the input text.

Firstly, applying text modification to the headlines will lead to texts that again look like headlines. Informal inspection of the headlines generated suggests that this requirement is fulfilled sufficiently.

The headlines are often grammatical, but not always. In some cases, the dependency parser has trouble selecting the full noun phrase in both the input text and headlines from the headline database, which leads to only partially substituted objects and subjects. Since Churnalist is meant for supporting game writers, we rely on the user to filter and discard ungrammatical output. Although we would like to improve the quality of the output to reduce the effort for game writers even more, we do not consider this a problem.

Since seed words and headlines are selected and matched at random, many of the generated headlines would probably not yet be considered context-appropriate. For example, readers might not relate headlines mentioning ‘company documents’ to the stolen company documents from the game text.

Some seed words have a stronger connection to the story and will evoke a stronger sense of coherence than others. For example, we expect that headlines that mention ‘hackers’ will have a stronger link for the reader with the story than headlines that mention ‘managers’. Most companies have managers; few companies have problems with malicious attacks from hackers. We expect that incorporating a stronger filter for seed words will lead to headlines with stronger link to the game story from the input text. For example, we could rank seed words based on their term frequency-inverse document frequency (tf-idf). This

⁶English Gigaword Fifth Edition by Linguistic Data Consortium, <https://catalog.ldc.upenn.edu/LDC2011T07>

Seedword	system administrator
Headline	Revealed: £500k number plate conman is a convicted people smuggler
Output	Revealed: system administrator is a convicted people smuggler
Seedword	hosting company
Headline	Pelosi has edge over Trump on budget negotiations, CBS News poll shows
Output	Hosting company has edge over Trump on budget negotiations, CBS News poll shows
Seedword	computer network
Headline	Met Office issues ice warning as snow hits UK
Output	Computer network issues ice warning as snow hits UK
Seedword	hacker
Headline	Uber loses latest legal bid over driver rights
Output	Hacker loses latest legal bid over driver rights
Seedword	sensitive company documents
Headline	Investigators revise cause of escape room fire that killed 5 girls
Output	Sensitive company documents revise cause of escape room fire that killed 5 girls
Seedword	forensical analysis
Headline	MPs' threat to block government's tax without second brexit referendum
Output	MPs' threat to block forensical analysis without second brexit referendum

Figure 5.9: Generated headlines for the input text in Figure 5.4.

would take into account that seed words that occur frequently in general are probably less representative for the input text than seed words that occur rarely in other English texts. For now, we leave the task of filtering the seed words and generated headlines to the user of Churnalist.

Choosing a random headline from the database for substitution is a mixed blessing. On the one hand, combining a context word with the randomized headline can lead to surprising and creative outputs. On the other hand, sometimes the link with the context word that was chosen for substitution is far-fetched or even downright ridiculous.

The application domain of Churnalist is supporting game writers in their creative task. Since Churnalist requires no linguistic knowledge, it is an accessible tool. Instead of relying on hand-written linguistic models, it requires external datasets for its text modification functionality. By using News API for collecting headlines and using FastText or ConceptNet as knowledge base, Churnalist can run fully on publicly available data. Similarly to PoeTryMe, users could choose to

use different datasets for their particular application, for example for a different language than English.

However, using external data for NLG has some caveats. Reusing headlines has as advantage that we do not have to write templates. The disadvantage is that quality of the output headlines will never be better than the quality of the headlines from the headline database. Using headlines from low-quality news outlets with click-bait headlines, the output headlines will show similar clickbait properties.

We developed Churnalist with open source libraries such as NLTK and spaCy. This has a few drawbacks. As long as we want to generate headlines in English, there are enough English language resources available. However, were we to generate text for a different language (such as Dutch), we have fewer options in programming libraries for processing natural language. For example, NLTK's noun phrase extractor is not implemented for the Dutch language.

5.4.2 Improvements

There are various ways to improve Churnalist as a tool.

Firstly, we can improve parts of the implementation, by using better and more appropriate tools and resources. We used an open source dependency parser for Churnalist that was trained on a standard corpus of English. Training the parser on a set of headlines could improve the accuracy of the parser, which might lead to better quality text modifications.

We could also expand Churnalist's approach to generation. Churnalist could use a generate-and-test strategy, like PoeTryMe, where multiple candidate headlines are generated and a fitness function determines the best candidate headline from this set as output. Instead of applying text modification to randomly chosen headlines, Churnalist could scan existing headlines for content words that already show a strong link with the input text, for example using a database of n-grams, and use these as a basis for modification. We could use sentiment analysis to filter out too negative headlines, or use a blacklist of words that represent sensitive topics (e.g. war, rape, suicide). Finally, Churnalist could extract names from the input text (such as "SuperSecure ltd" from our examples) and incorporate these in the headlines as well. Applying named entity recognition techniques on the input text would be a step in this direction.

5.5 Evaluation

We evaluated Churnalist via research platform Appen (previously Figure Eight, Crowdflower), using the same evaluation method as Veale [276], Alnajjar, Lepänen, Toivonen, et al. [4] and Gatti et al. [104].

We evaluated two aspects of Churnalist's outputs, in two experiments:

1. Form: Does Churnalist produce acceptable quality headlines?

2. Function: Does Churnalist produce headlines that are perceived as appropriate to the context?

We will discuss the first experiment ‘Form’ in Section 5.5.1, and the second experiment ‘Function’ in Section 5.5.2. The evaluation took place via an on-line survey in which participants were asked to rate each headline on a series of statements. To make sure the survey was not too long, and to compute a fair compensation for research participants, we tested a pen & paper version of the survey before starting each experiment. In both experiments, the order of presented texts was randomized for each participant to reduce order bias in our results. We obtained permission from the Ethics Committee of the faculty of EEMCS at the University of Twente prior to starting the experiments.

5.5.1 Form

The first evaluation experiment measures whether Churnalist generates plausible headlines. Does Churnalist’s text modification create acceptable quality headlines, or does it make human-written headlines worse?

The first experiment was conducted in December 2019, on crowdsource platform Figure Eight (later acquired by Appen). Participants were shown batches of 10 texts. Each participant could rate as many batches of 10 texts as they wished. The batches were a random sample of the following collection of 144 texts:

- 48 human written headlines, sourced from Gigaword
- 48 Churnalist headlines
- 48 human-written sentences from the body of newspaper articles, sourced from Gigaword

The texts were presented without any context.

Churnalist’s input was taken from four New York Times newspapers articles published in 2010. We used newspaper articles as input for Churnalist instead of text from video games, because these headlines were just used to compare Churnalist output text with human-written headlines. We chose older articles, so that when readers were reading and interpreting the texts, they would not be not biased by their knowledge of recent events. We used Churnalist as a (game) writer would: as an authoring aid where the user can influence every step in the generation process. Churnalist’s demo interface generates batches of 40 headlines. We generated one batch per NYT news article and we cherry-picked the best outputs until we had a total of 48 headlines.

We used 48 randomly picked GigaWord headlines as gold standard of headline quality. We also picked 48 random prose sentences (not headlines) from the New York Times articles as baseline.

We asked participants to rate each text on a 5-point Likert scale, using the following statements (Likert items):

1. This text looks like a headline
2. This text makes sense
3. This text is grammatically and syntactically correct
4. I think this text was written by a computer

In the following section, we will use the term *headline-ness* when referring to the property measured by Statement 1; *sense* for the property measured by Statement 2; *grammaticality* for the property measured by Statement 3; and *computer-generated* for the property measured with Statement 4.

We collected 2-5 judgements per text from 167 participants. On platform Figure Eight, participants can choose how many batches of 10 texts they want to rate. Each participant rated between 10 and 50 texts. Participants were paid 60 dollar cents for rating one batch.

To check whether our participants were proficient enough in English to properly rate the language quality of the texts, we also included a set of validation headlines in our experiment, all of which contained grammatical and spelling errors. Each participant rated these validation headlines prior to rating the normal evaluation texts. If they rated the validation headlines as highly grammatical, we excluded their ratings from our results.

Results for evaluation of headline form

Since we used a Likert scale, participant ratings are discrete. Statistically analysing Likert scales is problematic because of their discrete nature, and participants might differ in their interpretation of Likert scale items. How to interpret Likert scale results has been an object of discussion in various fields since 1940. However, we will follow the common methodology in NLG research. We assume that our Likert scale values are equally spaced ordinal values, e.g. we will treat the space between “Strongly disagree” and “Agree” equally as the space between “Agree” and “Neutral”.

We used the Shapiro-Wilk normality test to determine whether our ratings are normally distributed. The results of this test determine whether we can use parametric statistical tests like ANOVA. Figure 5.10 shows the results of the Shapiro-Wilk normality test. The ratings for all four properties are significantly non-normal.

Figure 5.11 and Figure 5.12 visualize the ratings for the four evaluation properties. The boxplots in Figure 5.11 confirm that our ratings are not normally distributed.

Since our data is not normally distributed, this means the prerequisites for using parametric tests are not fulfilled. We use a non-parametric test to find differences between the three systems. We applied a Kruskal-Wallis rank sum test for each of the four evaluation properties. The Kruskal-Wallis test H ranks

Property	W	p	Significance
Grammaticality	0.81807	$< 2.2e - 16$	*
Sense	0.79565	$< 2.2e - 16$	*
Headline-ness	0.77484	$< 2.2e - 16$	*
Computer-generated	0.78445	$< 2.2e - 16$	*

Figure 5.10: Results of the Shapiro-Wilk test on the ratings for four evaluation properties grammaticality, sense, headline-ness and computer-generated for all evaluation texts. The ratings for all four properties are significantly non-normal.

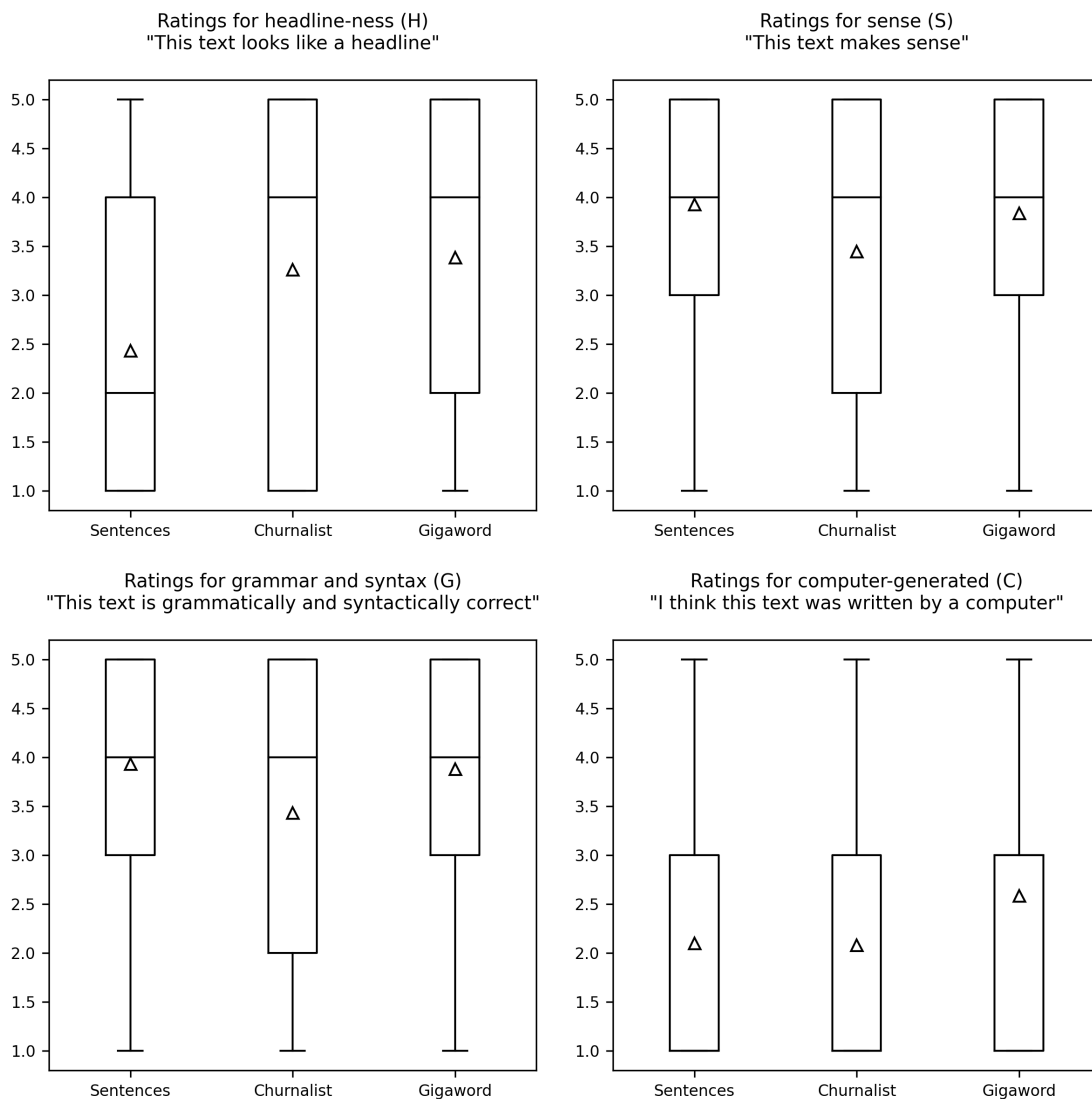


Figure 5.11: Box plots for the ratings of headline-ness, sense, grammaticality and likelihood that the text is computer-generated. Texts were rated on a 5-point Likert scale, with 1 "Strongly disagree" to 5 "Strongly agree". The triangle marks the mean.

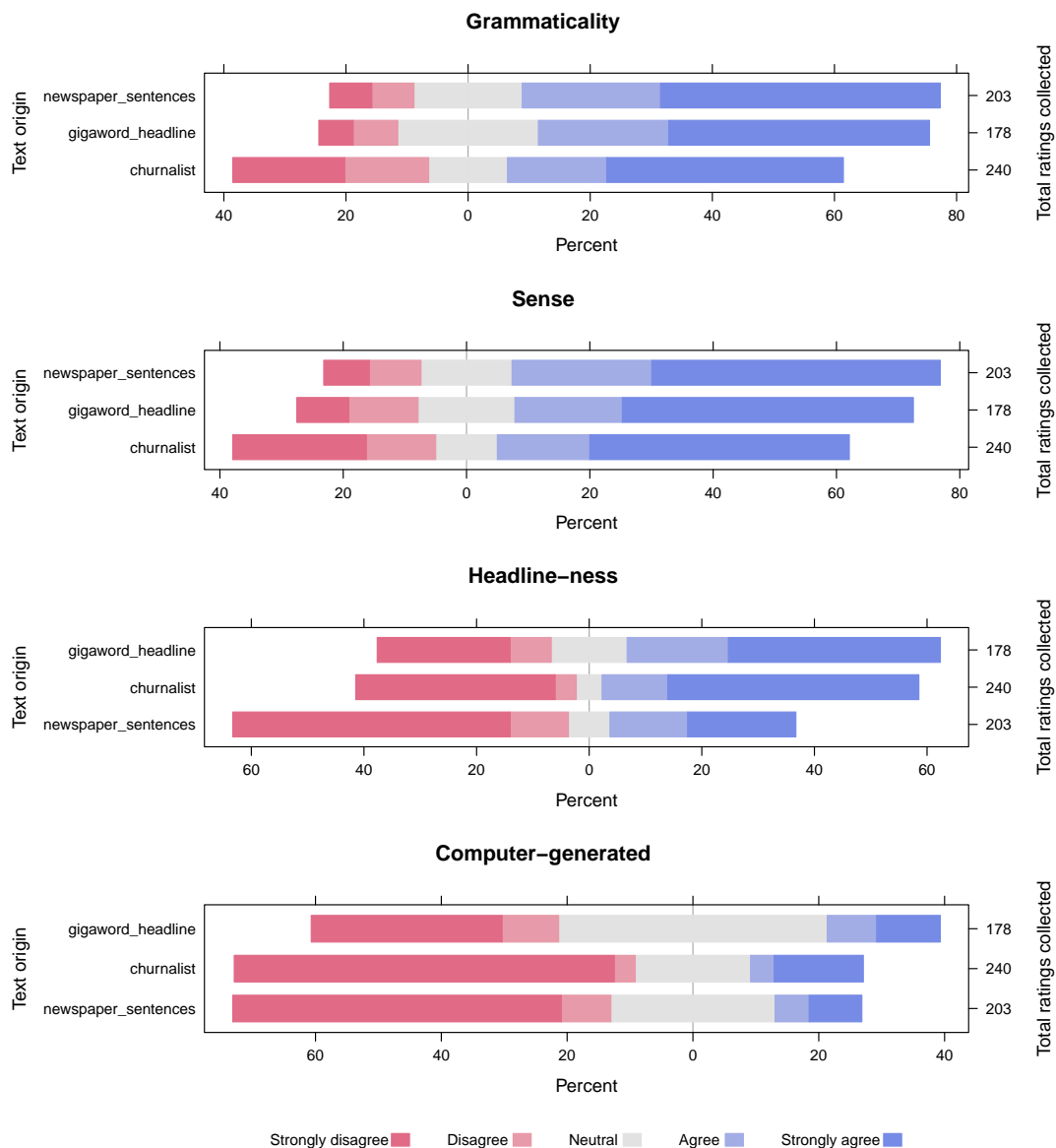


Figure 5.12: Diverging stacked bar charts for the ratings of headline-ness, sense, grammaticality and likelihood that the text is computer-generated.

all the ratings, and computes the mean rank for each of our three systems. We collected a total of 621 ratings, so the rank is a number between 1 and 621. Figure 5.13 reports the results of this test, and the mean rank for each system. The test results show that for each property significant differences exist between the systems.

To investigate the differences between the three systems, we do focused post-hoc tests using multiple comparison test after Kruskal-Wallis in R. In our post-hoc tests, we compared the mean rank of Churnalist to the mean rank of the two other systems using a two-tailed test. Figure 5.14 shows the results of the post-hoc tests.

Property	$H(2)$	p	Mean rank Churnalist	Mean rank Gigaword	Mean rank sentences	Significant ($p < 0.01$)
Sense	8.6564	0.01319	286.0042	322.7163	330.2783	*
Grammaticality	11.582	0.003055	281.9354	324.2135	333.7759	*
Headline-ness	35.063	2.433e-08	335.3854	344.3034	252.968	*
Computer-generated	23.594	7.529e-06	285.7271	361.6011	296.5099	*

Figure 5.13: Kruskal-Wallis rank sum test for each of the four evaluation properties.

Property	Systems	Observed difference	Critical difference	Significant ($p < 0.05$)
Sense	Churnalist vs Gigaword	36.71213	39.77799	*
	Churnalist vs sentences	44.27416	38.34586	
Grammaticality	Churnalist vs Gigaword	42.27807	39.77799	*
	Churnalist vs sentences	51.84045	38.34586	
Headline-ness	Churnalist vs Gigaword	8.917954	39.77799	*
	Churnalist vs sentences	82.417436	38.34586	
Computer-generated	Churnalist vs Gigaword	75.87404	39.77799	*
	Churnalist vs sentences	10.78277	38.34586	

Figure 5.14: Two-tailed multiple comparison test after Kruskal-Wallis. We compared Churnalist with Gigaword and sentences on all four properties.

From the post-hoc tests, we can conclude that participants rated Churnalist not significantly different from Gigaword on the property of sense. If we look at the rating visualization and the mean rank, participants thought the sentences made the most sense, and the Gigaword and Churnalist headlines less so.

In terms of grammaticality, Churnalist is rated significantly different from both Gigaword and newspaper sentences. This affirms the problems we noticed in Churnalist’s output, most of which are caused by Churnalist’s random substitution.

Churnalist is not significantly different from Gigaword headlines in terms of headline-ness, which is good news. The difference between Churnalist and newspaper sentences is statistically significant. From this we can conclude that readers think that Churnalist’s output are clearly headlines and not regular prose sentences.

The results for the computer-generated property are surprising. From the post-hoc tests, we can conclude that Churnalist differs significantly from Gigaword on this property, but not from sentences. If we look at the mean rank for this property, it turns out that participants thought it most likely that Gigaword headlines were generated by a computer. The bar charts from Figure 5.12 paint a more nuanced picture. Participants often rated Gigaword ‘neutral’ on this property, whereas they were divided on whether Churnalist’s outputs were computer-generated. Participants chose the two extremes, i.e. “strongly disagree” and “strongly agree”. more often than ‘neutral’ for Churnalist headlines, which leads to a lower mean rank on the computer-generated property.

5.5.2 Function of output

In the second evaluation experiment, we wanted to find out whether Churnalist generates headlines that fit the input context. With a survey, we tried to measure whether readers interpret generated headlines as context-appropriate (and plausibly coherent) for a given video game.

We used Prolific⁷ to recruit 50 participants for this experiment. We specifically recruited adult participants with a good proficiency of the English language. Participants received £2.80 for participating in the study. The survey took 5–20 minutes to complete. We used Google Forms to create the survey and collect the data. Instead of rating texts in batches, as in experiment 1, the survey for experiment 2 contained all questions and headlines. As a result, every participant rated every headline on every property.

The survey consists of three parts, and each part of the survey is about one real-world video game. We chose three games of different genres that contain news headlines: CookieClicker [267], a browser-based idle game; Deux Ex: Human Revolution [92], a science fiction action game; and SimCity 3000 [183], a city-building game. In the survey, each game is introduced with a title, a one-

⁷Prolific, <https://www.prolific.co>

CookieClicker

CookieClicker is an browser-based idle game in which the player takes the role of a cookie magnate. By employing an army of cookie-baking grandmothers, building cookie-producing factories and farming chocolate, the player can become the number one cookie-producer in the world!

Deus Ex: Human Revolution

Science fiction role-playing game Deus Ex: Human Revolution is set in 2027, just as human augmentation begins to enter mainstream life. Augmentation, the use of cybernetics in order to improve or replace human body parts, allows people to assume superhuman abilities, albeit not without limitations. Players take the role of Adam Jensen, a security consultant employed by biotechnology company Sarif Industries to protect a research lab in Detroit.

SimCity 3000

In city-building game SimCity3000 , the player takes on the role of a mayor. The objective of the game is to create a city, develop residential and industrial areas, build infrastructure and collect taxes for further development of the city. Importance is put on increasing the standard of living of the population, maintaining a balance between the different sectors, and monitoring the region's environmental situation to prevent the settlement from declining and going bankrupt.

Figure 5.15: Game descriptions for three video games with headlines. These descriptions were used as context in the survey, and as input for Churnalist to create new flavor headlines for these games.

paragraph description and one or more images. This information should help survey participants understand the game, even if they are not familiar with it. We used short descriptions of the three games, paraphrased from their Wikipedia entries, to contextualize the headlines in the online survey. Figure 5.15 lists the three game descriptions. The descriptions also functioned as the input text for Churnalist, to create generated headlines.

We use four categories of headlines for this experiment.

1. original headlines from the game, as our gold standard;
2. randomly chosen headlines from Gigaword, as our baseline;
3. headlines from Gigaword that contain at least one seed words, and
4. outputs from Churnalist.

Since CookieClicker is a client-based JavaScript game, CookieClicker's headlines can be extracted directly from the game's source code. Headlines from SimCity 3000 and Deus Ex: Human Revolution were downloaded from fan websites. Outputs from Churnalist were generated using the title and descriptive

CookieClicker	
From game	Cookie factories linked to global warming
Gigaword random	Clinton takes aim at budget today
Gigaword with seedword	Girl Scouts witness how the cookie crumbles
Churnalist	European Commission criticizes cookie magnate on waste crisis
Deus Ex: Human Revolution	
From game	500 augmented workers head to Panchaea as Arctic Ocean Installation breaks new scientific ground
Gigaword random	Beijing private cars triple in five years
Gigaword with seedword	Witness in Detroit case denies any terror ties
Churnalist	Syrian Foreign Minister to visit augmented human
SimCity 3000	
From game	City Tourist Bureau Launches City Beautification Project
Gigaword random	Indonesia to Commemorate Global Hand Washing Day
Gigaword with seedword	Sharks Hire Sims as Coach
Churnalist	Sacked Minister Accuses Municipality of Accepting Favors

Figure 5.16: Example headlines for each game and every headline category: original headlines from the game (gold standard), randomly chosen headlines from Gigaword (baseline), headlines from Gigaword that contain at least one seed word, and outputs from Churnalist.

paragraphs from Figure 5.15 as input. Headlines from Gigaword were randomly picked using a Python script.

The four headline categories differ on three properties. Firstly, the first three categories are hand-written, whereas Churnalist outputs are generated. We can use this distinction to measure the difference between human-written and computer-generated headlines. Secondly, Gigaword with seed words and Churnalist contain seed words as reference to the input text, whereas the headlines in the random Gigaword category explicitly do not contain any of the words that occurred in the input text. We can use this difference to measure the difference between headlines with and without seed words. Headlines from the original game are a mixed bag for this category: 4 out of 5 original SimCity headlines, 4 out of 5 Deus Ex headlines and 3 out of 5 headlines of CookieClicker contain seed words. Finally, we included headlines from the original game as a gold standard. We can use this to measure differences in ratings between the headlines from the original game, and the other three categories. Figure 5.16 shows example headlines for each game and headline category.

Each headline was scored on a 5-point Likert scale on the following two statements:

1. This headline is meaningful given the game's description.

2. It is plausible that this headline is part of the game.

Our initial survey contained a third statement, namely ‘This headline fits the described game’. However, a pen and paper test round indicated that this confused participants, as Statement 2 and Statement 3 were interpreted as too similar.

In the following section, we will use the term *meaningfulness* when referring to the property measured by Statement 1, and *part of game* for the property measured by Statement 2.

Each part of the survey corresponds to one game, and for each game, participants rate 5 headlines of each type on two different properties. In total, the survey contained 3 games x 4 headline types x 5 headlines x 2 properties = 120 ratings.

Results for evaluation of headline function

Figure 5.17 visualizes the Likert ratings we collected for the statements “This headline is meaningful given the game’s description” and “It is plausible that this headline is part of the game.” We collected a total of 3060 headline ratings per property from 51 participants, with 60 ratings per participant. From the barcharts, we can see that the differences between Gigaword with seed words, Churnalist, and the original game texts are negligible, and that random Gigaword headlines score much worse than the former three categories, on both properties.

As was the case with the first evaluation experiments, both the ratings for meaningfulness, $W = 0.83381$, $p < 2.2e - 16$, and part of game, $W = 0.83859$, $p < 2.2e - 16$, were significantly non-normal. As the ratings are not normally distributed, we again used a non-parametric Kruskal-Wallis rank sum test to determine where the differences are in our results.

The Kruskal-Wallis test H ranks all the ratings, and computes the mean rank for each of our four headline types. Since we collected a total of 3060 ratings per property, the rank is a number between 1 and 3060. Figure 5.18 reports the results of this test, and the mean rank for each system. The test results show that for both properties significant differences exist between the four systems.

To investigate the differences between Churnalist headlines and the other three headline types, we did focused post-hoc tests using a multiple comparison test after Kruskal-Wallis. In our post-hoc tests, we compared the mean rank of Churnalist to the mean rank of the three other systems using a two-tailed test. Figure 5.19 shows the results of the post-hoc tests.

From the post-hoc tests, we can conclude that participants rated Churnalist as significantly different from the random Gigaword headlines on both properties. The ratings for Churnalist’s outputs do not differ significantly from the ratings of original game texts, which is our gold standard, or Gigaword headlines with seed words. The bar charts in Figure 5.17 subscribe these findings. In the chart, we

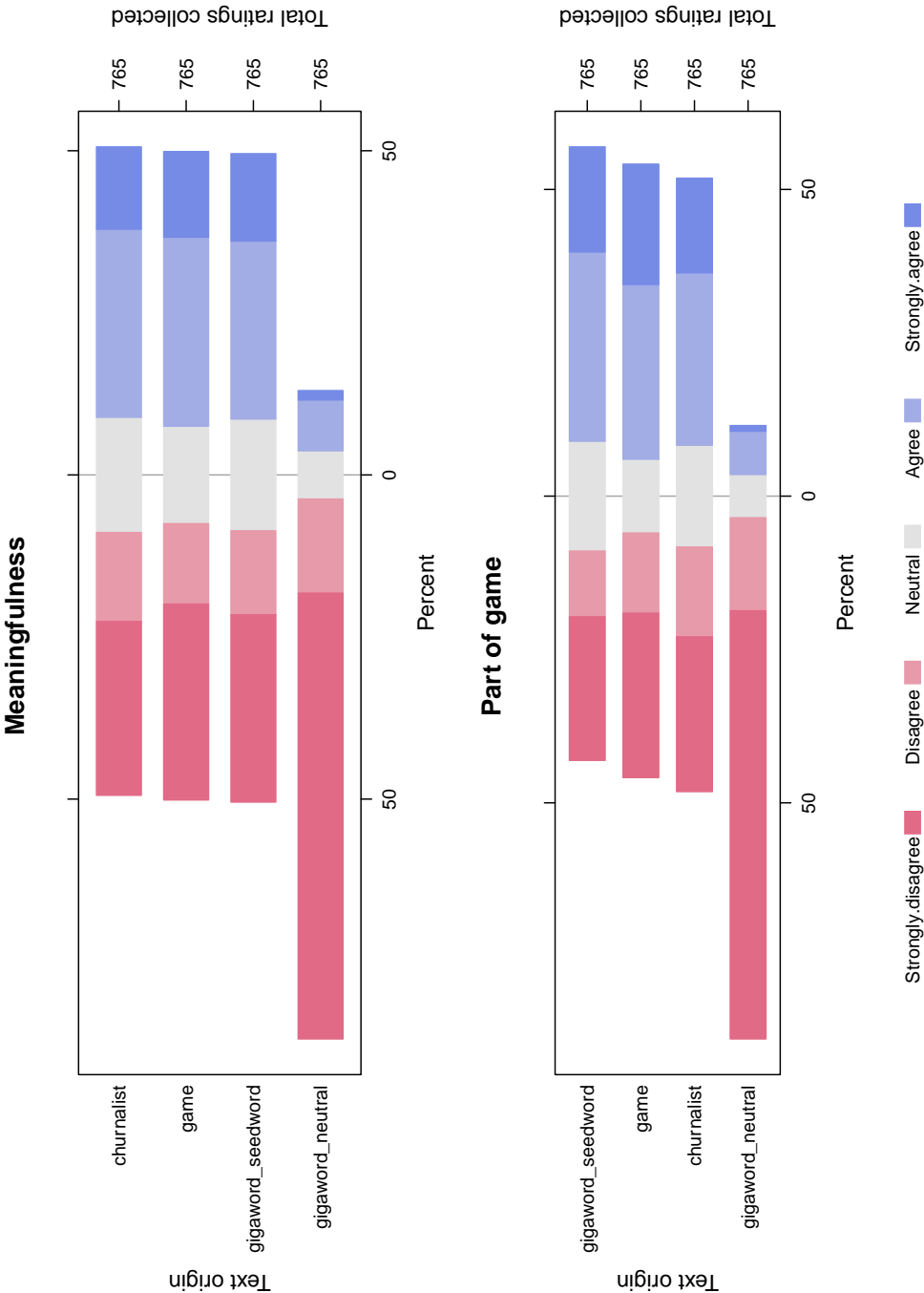


Figure 5.17: Diverging stacked bar charts for the ratings of meaningfulness and part of game. Results for all three games (CookieClicker, Deus Ex and SimCity) were combined.

Property	$H(3)$	p	Mean rank Churnalist	Mean rank Game	Mean rank Gigaword seed words	Mean rank Gigaword neutral	Significant ($p < 0.001$)
Meaningfulness	440.63	$< 2.2e - 16$	1733.761	1704.261	1711.044	972.934	*
Part of game	564.48	$< 2.2e - 16$	1704.839	1742.441	1777.108	897.6124	*

Figure 5.18: Kruskal-Wallis rank sum test for each of the two evaluation properties.

Property	Systems	Observed difference	Critical difference	Significant ($p < 0.05$)
Meaningfulness	Churnalist vs game	29.50000	108.145	
	Churnalist vs Gigaword with seed words	22.71634	108.145	
	Churnalist vs random Gigaword	760.82680	108.145	*
Part of game	Churnalist vs game	37.60196	108.145	
	Churnalist vs Gigaword with seed words	72.26993	108.145	
	Churnalist vs random Gigaword	807.22614	108.145	*

Figure 5.19: Two-tailed multiple comparison test after Kruskal-Wallis. We compared Churnalist with the original game texts, Gigaword with seed words and random Gigaword on both properties.

can see that the differences between the three highest rated categories are very small compared to the ratings for Gigaword neutral.

The differences between Churnalist, original game headlines, and Gigaword with seed words for the statement ‘this headline is meaningful given the game’s description’ are minimal. This is interesting, because it implies that participants rate a headline as meaningful if it reuses words from the game description. The lack of a significant difference between Churnalist and Gigaword headlines with seed words implies that we could just as well use Gigaword headlines as source of inspiration for game writers, instead of creating new headlines using Churnalist. Of course this leads to problems with copyright, as we can’t just blindly insert headlines from published news paper articles in a role-playing game. Note that we can only use unmodified Gigaword headlines if the game narrative does not contain too many made-up words. This was a challenge in the case of Deus Ex, which is a science fiction game. Most words that are important in the Deus Ex narrative, such as augmentation, LIMB clinic (augmentation clinics) and neuropozyne (a fictional drug), are not common words in day-to-day English. As a result, few headlines in the Gigaword dataset contain at least one seed word. Luckily, Deus Ex takes places in a future version of Detroit, so we could just use any headline that contains the word ‘Detroit’ for the survey.

If we look at the ratings for the statement ‘It is plausible that this headline is part of the game’, the ratings for headlines from Churnalist, the original game and Gigaword with seed words differ slightly, albeit still not significantly. The latter two categories are on average rated higher on ‘part of game’ than Churnalist headlines. We saw in the previous evaluation experiments that participants definitely notice the differences in language quality for Churnalist headlines and Gigaword headlines, so we estimate that this difference can be explained because both categories contain manually-written headlines.

5.6 Conclusion

We have presented Churnalist, a system for generating fictional headlines. The content of the headlines is determined by the input text, or rather the noun phrases present in the input text, which function as seed words. It expands the list of seed words by querying a knowledge base for related words and injecting these into existing headlines via word substitution.

We circumvented problems with out-of-vocabulary seed words by using word vectors based on subword information. For any out-of-vocabulary seed word, we can create a word vector and use it to find related words in the knowledge base. Alternatively, we can expand the list of seed words by querying ConceptNet for related words. The user can partially control Churnalist’s output by filtering the intermediary output of each step in the system pipeline.

Churnalist can be used by game writers, as an authoring aid for writing flavor text. We have provided example outputs for every step in the system pipeline,

using a representative game text as input. Our system was developed using publicly available datasets, open source libraries and only simple text modification techniques. It requires no linguistic expertise from its users.

Although Churnalist is currently implemented for English, the use of external datasets allows us to adapt the system to other languages and use cases with minimal effort. This makes Churnalist suitable for different languages and game types.

We conducted an evaluation of Churnalist's outputs on various properties. First, we evaluated the headlines on their form, i.e. whether they were grammatically correct, whether they made sense to readers, whether the text looked like a headline, and whether readers thought the headline was computer-generated. In our analysis of the results, we compared the ratings for Churnalist with those of a baseline, i.e. prose sentences from newspaper articles, and a gold standard, i.e. hand-written headlines from published articles. Churnalist's outputs scored significantly worse on grammar than both other categories. It scored slightly worse on sense compared to the newspaper sentences, although not significantly different from the gold standard headlines, which is encouraging. Participants rated the headline-ness of outputs not significantly different from the gold standard headlines. Interestingly enough, participants thought that the gold standard, human-written headlines were most likely to be generated by a computer. A possible explanation for this is the recent media attention for large-scale language models, such as GPT-2 and GPT-3, which can create multiple paragraphs of seemingly coherent English prose at the press of a button. Various journalists have used large language models to create (parts of) news articles, e.g. [204, 207, 121]. This might have caused participants to believe the prose sentences from our evaluation survey were created with a similar technique.

We also evaluated the generated headlines on their intended function as flavor text. We asked participants to rate headlines on their meaningfulness in the context of a game description, and on the plausibility that they are part of the relevant video game. For this evaluation, we compared Churnalist with random Gigaword headlines as baseline, Gigaword with seed words as high-quality and human-written equivalent of Churnalist's outputs, and original game headlines as gold standard. For both properties, Churnalist was rated significantly higher than the baseline. Churnalist's performance differed when compared to the two high-quality human-written categories. For meaningfulness, Churnalist headlines were rated highest, but the differences between the three categories are minimal. When asked how plausible it is that headlines are part of the video game, participants rated Churnalist's outputs lower than the two human-written categories. This might be explained by the clear difference in language quality between Churnalist outputs and human-written headlines, which we saw in the first evaluation experiment. However, for both properties, the differences between Churnalist and our gold standard were not statistically significant.

Rebuilding Churnalist with GPT-2

6.1 Introduction

In the previous chapter, we saw that we can use Churnalist to generate flavor text headlines. The evaluation of headline *functionality* showed that headlines that contain seedwords score well on meaningfulness in the context of a game. Of all headlines with seedwords, Churnalist's outputs scored lowest on the statement 'it is plausible that this headline is part of the game'. The evaluation of the *form* of headlines showed that Churnalist's outputs were significantly less grammatical and syntactically correct than human-written headlines, which could explain Churnalist's lower score on the 'part of game' property.

The subpar language quality of Churnalist's outputs is caused by two things. Firstly, the low accuracy of the various open source libraries on which Churnalist is built trickles down to the output. Churnalist is a sequential pipeline of various open source libraries for natural language processing, such as Pattern and SpaCy. Even though the accuracy of open source NLP libraries might be over 80 or 90 percent, small error rates in libraries used in sequence might lead to large accuracy errors downstream. For example, imagine a pipeline consisting of three NLP functions with each an accuracy of 80%. This pipeline produces outputs that are correct only 51.2% of the time: $0.8 \cdot 0.8 \cdot 0.8 = 0.512$. Additionally, since the NLP libraries are meant for processing English prose text, their accuracy on headline text might be even lower.

Secondly, Churnalist uses naive substitution based on random number generation. To construct a headline, a random headline is chosen from the Gigaword data and its object is substituted with a random seed word from the list. In many cases, the chosen seed word does not fit the verb or prepositions of the original headline, which results in an ungrammatical headline.

In this chapter, we describe Churnalist version 2, which has a word substitution module based on GPT-2. Because of its large training set and its resulting latent knowledge of the English language, GPT-2 is known to perform well on

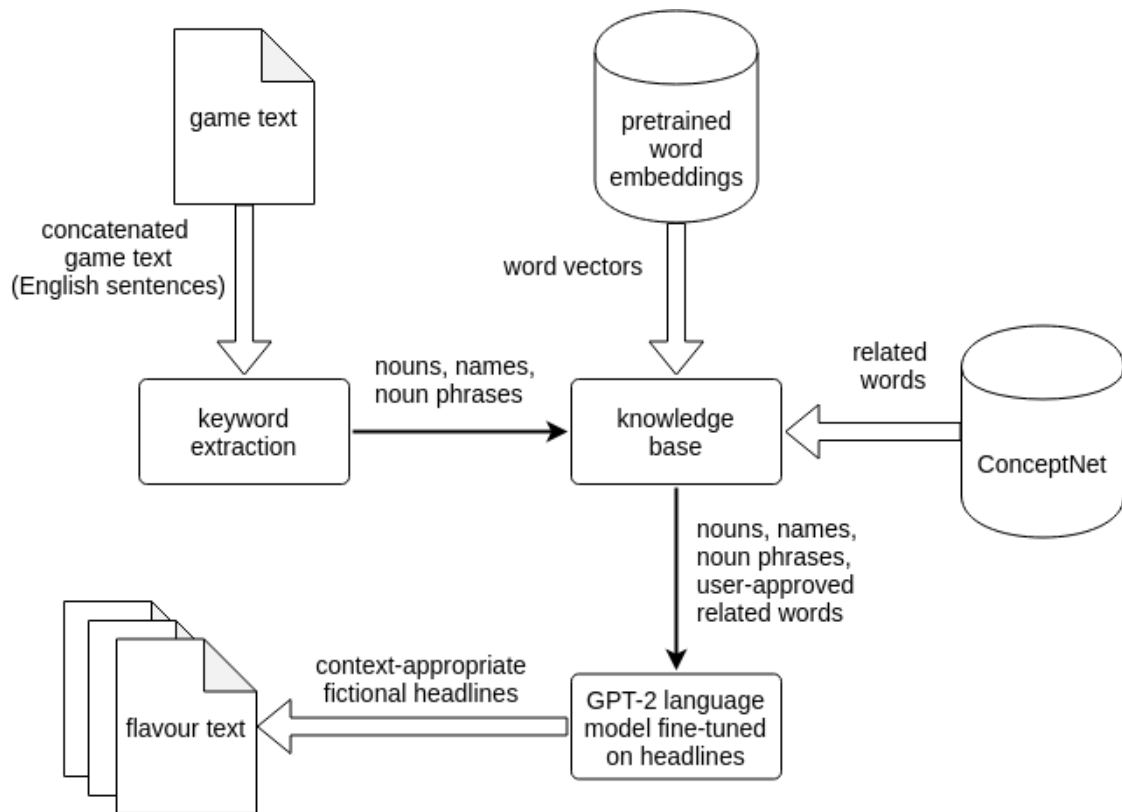


Figure 6.1: Architecture of Churnalist version 2. This version has a fine-tuned GPT-2 model instead of a database with headlines and a word substitution module.

coherent and grammatical text generation, even without additional fine-tuning. If we fine-tune GPT-2 on headlines extracted from GigaWord, the resulting language model can be used in place of Churnalist’s word substitution module. We hypothesize that substituting Churnalist’s rule-based substitution module with end-to-end generation based on GPT-2 could significantly increase the quality of Churnalist’s outputs. Increased language quality means that the outputs require less manual editing before they could be included in a game, which is a practical advantage should Churnalist be used for flavour text generation for a real-world game.

The related work for this chapter was already discussed in the background chapter on natural language generation, especially Section 3.3.3 and Section 3.3.4.

6.2 Method

We substitute Churnalist’s headlines database and the word substitution component with a fine-tuned GPT-2 model for English. Figure 6.1 shows the architecture of Churnalist version 2.

We want to use GPT-2 as a basis for a language model with the same functionality as Churnalist’s substitution module. That means a fine-tuned GPT-2 model should have the following two properties:

1. The GPT-2 model should be able to produce the subset of English that is specific to headlines
2. The GPT-2 model should be able to produce headlines that contain user-defined seed words

We performed two experiments to create a language model that has both properties. For the first property, we fine-tuned the smallest GPT-2 model (124M parameters) on a dataset of headlines sourced from GigaWord. This first experiment is described in Section 6.2.1. For the second property, we conditioned GPT-2 by fine-tuning the smallest GPT-2 model on a dataset of headlines, annotated with their object. We use object as a proxy for the topic of the headline. This experiment is described in Section 6.2.2. We use `gpt-2-simple`¹, an open-source library for working with GPT-2, as basis for our implementation. Unless otherwise specified, all texts are generated with the default generation parameters of `gpt-2-simple`. Note that the example results in this chapter were not cherry-picked. To avoid sampling bias, we either list all generated samples, or present a subset of generated texts randomly selected by a computer.

6.2.1 A GPT-2 language model for English headlines

Our first goal is to obtain a language model that can generate text in headline form. Headlines have a different structure from normal English text. Since GPT-2 is pre-trained on a large corpus of English webtext, it has not modeled headline-specific English yet. We created a GPT-2 model that has modeled ‘headlines’ by training the smallest GPT-2 model on a cleaned subset of the headlines extracted from the annotated Gigaword [194] dataset. This is the same dataset that was used as source material for Churnalist version 1. We filtered the GigaWord headlines for headlines that consist only of alphanumeric characters. This way, we end up with data that is as clean as possible, without going through the headlines by hand. If we would have used the unfiltered data, GPT-2 would be trained on headlines that contain mismatched apostrophes, quotes, currency symbols, and other noise. The cleaned dataset contains 1.5 million headlines in English. To denote the beginning and ending of headlines, we enclose them in special start and end tags: `<|startoftext|>` and `<|endoftext|>`.

We fine-tune a fresh GPT-2 model with 124 million parameters in 1000 training steps with the default fine-tuning parameters from `gpt-2-simple`. Training took 1 hour in an online Google Colab environment. After 400 steps, the language model had already learned the headline format and occasionally included

¹`Gpt-2-simple` by Max Woolf, <https://github.com/minimaxir/gpt-2-simple/>

-
- 1 Chinese firm to build major coal mine in Uganda
 - 2 Five patients die after being shot in California
 - 3 China adopts proposed new full-scale push-back on nuclear arms proliferation
 - 4 Indonesian gold-buyers end up with huge debt to be paid back in six years
 - 5 Bosnian Serb president arrives for talks with Serb insurgents
 - 6 Pegasus-style aircraft crash kills three
 - 7 Quarter-final spot for British soccer
 - 8 Hong Kong bans tobacco advertising in schools
 - 9 Chinese Embassy in Oman Attacked
 - 10 Sweden Set Free of Criminals
-

Figure 6.2: Headlines generated by the fine-tuned GPT-2 model from our first experiment. The prompt was `<|startoftext|>`. None of the generated headlines were present in the training set.

-
- 1 Video game writers see special power of Internet
 - 2 Video game writers to be put on trial in Sweden
 - 3 Video game writers to sue game companies over Afghanistan ties
 - 4 Video game writers want more cash
 - 5 Video game writers and researchers face charges as criticism of links
 - 6 Video game writers fail to check for plagiarism
 - 7 Video game writers say they were full of pot
 - 8 Video game writers banned for lifetime
 - 9 Video game writers found dead on Canadian island
 - 10 Video game writers hit back at Nintendo over non-payment-spending rumors
-

Figure 6.3: Headlines generated by the fine-tuned GPT-2 model from our first experiment. The prompt was `<|startoftext|>Video game writers`. None of the generated headlines were present in the training set.

reporter names in headlines. Figure 6.2 shows 10 outputs generated by the fine-tuned language model after 1000 training steps.

We can see from Figure 6.2 that our fine-tuned language model indeed creates text in ‘headlines’. At first glance, the overall quality of the headlines is also much higher than those of the first Churnalist. Churnalist version 1 often outputs headlines where the verbs or prepositions of the headline do not match the substituted seedword. In contrast, the headlines generated by GPT-2 in Figure 6.2 are grammatical, and the verbs match the lexicographical form of the subjects.

We can already use this model to generate headlines about a specific topic, by extending the prompt to include specific words. Figure 6.3 shows 10 headlines that were generated by the prompt `<|startoftext|>Video game writers`.

The generated headlines about video game writers show GPT-2's strength in generating coherent outputs. Various headlines contain references to concepts that are semantically related to video game writers, such as “game companies”, “plagiarism” and “Nintendo”. Only headline 5 is incoherent, as it is not clear what ‘as criticism of links’ means.

Using this language model for text generation has an important limitation compared to Churnalist version 1. We can only control the output of this language model by changing the prompt. If we want to include seed words in the output, they can only be included at the start of a headline, as demonstrated in Figure 6.3. In the next section, we will address this limitation.

6.2.2 Generating headlines with specific keywords

From the previous experiment we have seen that even the smallest GPT-2 model can learn the language specific to headlines relatively quickly, i.e. in only 1000 training steps. However, we also want to be able to guide the language model towards generating headlines that contain words chosen by Churnalist's user. The seed words should be able to occur anywhere in any position in the headline, and not just at the beginning. We trained a second language model that is conditioned on an extended version of the previous task. GPT-2 can be conditioned on specific NLP tasks using only natural language sequences from the training set. For example, to condition GPT-2 on translating English to French, training the model on datapoints of the form `translate, English text, French text` will lead to a fine-tuned model that can create French text from a prompt of the form `translate, English text`. Our language model should be able to perform the following task: given a seed word, create a text in headlines that contains that seed word.

In this experiment, we use the object of GigaWord headlines as a proxy for the topic of a headline. GPT-2 could just as easily have been fine-tuned on the task of substituting the subject instead, or both subject and object. The method for fine-tuning and generation would have been the same.

However, we plan to compare GPT-2's outputs with those of Churnalist version 1. Our earlier experiments showed that Churnalist version 1 created headlines with the best language quality if only the object of headlines was substituted. Substituting the object by a random phrase is less obvious to a reader than substituting the subject, because the chance that the verb matches a random object is higher than that it matches a random subject. For example, compare substituting the object and subject in the headline “US president decides on carbon emissions.” The object “carbon emissions” can be substituted with a whole range of phrases before the headline becomes ungrammatical. On the other hand, the majority of substitution candidates will fit the verb “decides on” as subject. Consequentially, the last version of rule-based Churnalist only substituted objects. We plan to compare the outputs of our fine-tuned GPT-2 models

with those of Churnalist version 1, which means both systems should perform the exact same task.

We created a language model that can fulfill this task by training GPT-2 on headlines that are annotated with the seed word, with each part of the input sequence surrounded by special tags, i.e. `<|startoftext|><|object|>seed word<|headline|>headline with seed word as object<|endoftext|>`. This way, the model is conditioned on this structure: it learns from the examples that if the prompt contains `<|object|>`, a seedword, and `<|headline|>`, the headline should contain the seed word as the object of that headline.

We used `<|object|>` as the tag to denote the topic of the headline. We can use any tag that we like, as long as that particular sequence of tokens is unique enough, given the training sets. Since GPT-2 has been trained on millions of pages of web text, this means that our tags should be unique enough for this training set, not just the training set we use for fine-tuning. For example, prompts that use tags that are common in programming languages, such as HTML tags, might accidentally lead to sampling completions that contain additional HTML, thus distracting the model from the task we want to teach it.

We used SpaCy's dependency parser to annotate all headlines in our dataset with their object. It took SpaCy about two hours to annotate the headlines. Headlines for which Spacy could not find an object were removed from the dataset, which left us with about 750,000 headlines. The resulting format of the training set can be seen in Figure 6.4. From datapoint 1 and 4 we can see that SpaCy's dependency parser does not have perfect accuracy. For these two headlines, the right objects are "funds for peacekeeping along Ethio-Eritrean border" and "new law on religious freedom". However, we estimated that the accuracy of the dependency parser is good enough for our purpose. Annotating the 1.5 million headlines by hand is not a feasible alternative.

We fine-tuned a fresh GPT-2 model of 124 million parameters for 5000 steps on the annotated dataset. Figure 6.5 shows examples of headlines that were generated by the resulting model. Generally speaking, the resulting model has indeed learned to incorporate the phrase between the object-tag and headline-tag.

Figure 6.6 shows 10 headlines that were generated from prompts that specify a seed word. From these examples, we can see that the GPT-2 model is not always successful in incorporating the word in the headline. However, even when the seed word does not literally occur in the headline, as in the case of headlines 3, 6 and 7, the headline still contains related words or substrings.

6.3 Error analysis

To assess whether incorporating GPT-2 improves the output of Churnalist, we performed an error analysis of headlines generated with Churnalist version 1 and version 2. We did not perform another evaluation with human participants,

1	< startoftext >< object >Funds< headline >OAU Seeks Funds for Peacekeeping Along Ethio-Eritrean Border< endoftext >
2	< startoftext >< object >Tibetan development< headline >Chinese Tibetologists introduce Tibetan development in Prague< endoftext >
3	< startoftext >< object >its air quality objectives< headline >Hong Kong to review its air quality objectives< endoftext >
4	< startoftext >< object >new law< headline >Spanish government to prepare new law on religious freedom< endoftext >
5	< startoftext >< object >likely ethics probe< headline >Bush congressional ally faces likely ethics probe< endoftext >

Figure 6.4: Datapoints in our second training set. We used SpaCy to annotate the headlines in our dataset with their objects.

because the evaluation of Churnalist version 1 (see Section 5.5) already showed that reusing words from the input in the output leads to a higher perceived coherence of generated texts.

We generated 150 headlines for each version of Churnalist, using the seed words from the 3 games used for the Churnalist evaluation in Chapter 5: CookieClicker [267], Deux Ex: Human Revolution [92] and Simcity 3000 [183]. The seed words are shown in Figure 6.7.

We evaluated the headlines by hand, manually annotating them with any errors we found. Figure 6.8 shows the various errors we found, together with an example from the 300 headlines.

Figure 6.9 shows the analysis results. The most frequently occurring problem of Churnalist version 1 is that SpaCy's dependency parser selects the wrong substring of a Gigaword headline as object. Another common problem is a mismatch between the verb of the original headline and the inserted seedword, which occurs 34% of the time. As we can see, the errors that plague Churnalist version 1 hardly occur in outputs of Churnalist version 2. In that sense, replacing the substitution module with a fine-tuned GPT-2 language model is an improvement. However, Churnalist version 2 introduced two new errors, which did not occur in headlines generated by the rule-based system: missing seed words and repetition. Generating repetitive language is something that occurs frequently with GPT-2, especially with smaller language models. Missing seed words occur when we ask Churnalist version 2 to generate a headline with a seed word that is a

	< object >	< headline >
1	questions	Finnish foreign minister questions <u>EU constitution</u>
2	bribery	Britain investigators probe bribery by British winners of <u>Olympic</u>
3	religious freedom	US urges religious freedom for Taiwan refugees
4	new product	China to introduce new product for water quality monitoring
5	Dogs	Chinese Children to Help Raise Dogs in Western Areas
6	France	UN chief Blames France for Taliban
7	a third	SKorea accepts a third of its EU aid
8	Italy	Spain asks Italy to clarify its stance on EU constitution
9	Moves	Kosovo Government Makes Moves to Keep Peace Alive
10	Karzai	URGENT Taliban accuse Karzai of lying in

Figure 6.5: Headlines generated by the fine-tuned GPT-2 model from our second experiment, without any prompt. Bold text denotes the object generated by GPT-2, underlined text is the actual object of the headline.

	< object >	< headline >
1	video games	Japanese TV stations told to stop using video games at work
2	natural language processing	Sri Lankan students practicing natural language processing
3	PhD students	School of Globalization calls on students to pursue PhD in HK
4	research project	China sets up research project for foreign students
5	Dutch language	Students protest Dutch language in university
6	dissertation	China to receive <u>paper version of World Heritage List</u>
7	academia	Landmine outrage hits <u>education in south China province</u>
8	universities	Uganda to teach universities to women and girls in poverty
9	grant proposals	Iraq continues grant proposals for sports development
10	graduation day	Vietnam marks graduation day of foreign diplomats

Figure 6.6: Headlines generated by the fine-tuned GPT-2 model from our second experiment, with a prompt that contains an object-tag and seedword. Bold text denotes the seed word, underlined text is the actual object of the headline.

CookieClicker	Deux Ex	SimCity 3000
cookie magnate	augmentation	mayor
grandmothers	cybernetics	residential areas
cookie-baking grandmothers	augmentations	city
factories	research lab	industrial areas
cookie-producing factories	Detroit	city development
chocolate	superhuman abilities	city infrastructure
cookie factory	Sarif industries	municipality
number one cookie-producer	biotechnology	local taxes
cookie	augmented human	municipal taxes
chocolate	human enhancement technologies	local population
grandmother		
sugar		

Figure 6.7: Seed words for three video games with headlines used as input for Churnalist to create new flavor headlines for these games.

proper name, a non-English word, or a word that is not frequently used in the English language. In some cases, GPT-2 incorporates a word that shares a substring with the seed word. For example, `<|object|>augmentation` resulted in headlines with the words ‘segmentation’, ‘segregation’ and ‘fragmentation’. In other cases, there was no clear relation between the seed word and the generated headline. Especially the seed words for the game Deus Ex, such as ‘augmentation’, ‘augmented human’ and ‘Sarif industries’, triggered this error. Made-up words are common in fantasy and science fiction narratives, so this limitation could be a problem in the context of generating texts for video games with a fantasy or science fiction setting.

6.4 Discussion

The results show that using GPT-2 instead of Churnalist’s substitution module improves the language quality of Churnalist version 1. Some issues, such as dependency parser errors, lexicographical errors and replacement errors, completely disappeared. Mismatches between verbs and prepositions and topics were mitigated in version 2, although some outputs still have these issues.

Unfortunately, using a neural language model introduces a few new issues in outputs that did not plague outputs in Churnalist version 1. In terms of output language quality, we find new issues such as repetitive language (a common problem in language generated by the smallest GPT-2 models) and ‘hallucination’, i.e. generating text about things that are not present in the input. In the case of our model, we mean that GPT-2 generates headlines that do not contain the seed word from the prompt at all. Instead, these faulty headlines contain words that share a substring with the seed word, such as ‘fragmentation’ instead

Error	Example
Wrong object selected by parser	Dow Jones sells [stake cookie] in AmericaE-conomia magazine
Wrong singular/plural form	Petrochemical Giant Raises [Funds farming chocolates] in Stock
Implementation error	Congressmen recomb[end city] end to baseball monopoly
Verb mismatch	Hong Kong Stocks Hit [New Low city infrastructure]
Preposition mismatch	Giuliani promises [advice mayor] by May on measures against Mexico City crime
Seed word not present in headline	Russia says it will not be forced to use aluminium fragmentation (seed word: augmentation)
Repetition	Australia to modernize and modernize armed forces

Figure 6.8: Examples for the various error types we found in the 300 headlines generated for the error analysis. The errors are denoted in bold. The pipe symbol | denotes substitution.

Error	Churnalist version 1		Churnalist version 2	
	Headl	Perc	Headl	Perc
Wrong object selected by parser	53	35.33%	0	0%
Wrong singular/plural form	14	9.33%	0	0%
Implementation error	2	1.33%	0	0%
Verb mismatch	51	34%	11	7%
Preposition mismatch	13	8.66%	2	1.33%
Seed word not present in headline	0	0%	18	12.66%
Repetition	0	0%	2	1.33%
Headlines without errors	55	36.66%	119	79.33%

Figure 6.9: Errors in 150 headlines generated with Churnalist version 1 and version 2. Multiple errors can occur in one headline.

of ‘augmentation’. It is as if GPT-2 misread the prompt. The latter often occurred when the prompt contained non-English or rare English words. Repetition and hallucination problems have been reported by other researchers as well. Literal object hallucination is a common problem in caption generation for images [220], and small GPT-2 models are known to converge on repetitive language [209, 130].

Our error analysis showed that overall, the language quality of Churnalist version 2 is significantly better than that of version 1. Of the headlines generated

with version 1, 36% can be used without any additional manual correction, for 79% of headlines generated with Churnalist version 2.

In determining which approach yields the best results, we should also consider other aspects of both versions of Churnalist. The rule-based version of Churnalist that was described in Chapter 5 is a white box system. In every generation step, the user can inspect the actions of the system and influence the generation process. Churnalist version 2 uses neural generation, which is not fully transparent. Recent research shows that neural language models are not fully opaque. With brute-force techniques and statistical analysis, it is possible to gain insight in latent information in GPT-2, such as biases. However, getting this information is significantly more complex than just inspecting the generative progress of a white-box system. Consequentially, if Churnalist version 2 creates bad quality outputs, such as outputs with offensive contents, it is hard to assess what caused this problem. In the case of Churnalist version 1, we have much more possibilities for inspection and quality control.

One of the limitations of Churnalist version 1 that was not addressed is that Churnalist works only for English language headlines. In theory, we could use the same approach to build a rule-based headlines generator for other languages, provided there is a large enough dataset of headlines available for that language, and a set of natural language processing libraries. The GPT-2 extension of Churnalist does not solve this problem. However, recent work by Vries and Nissim [279] suggests that GPT-2 can be ‘recycled’ to yield a language model for other languages, such as Italian or Dutch. Another alternative to copying GPT-2’s architecture is using one of the open-source alternatives to the GPT-2 model, such as GROVER [290], a method that was used to create ARAGPT2 [8], a large scale language model for Arabic. Combined with a dataset of headlines in that other language, our method could be adapted to other languages.

Another limitation of Churnalist version 1 is that we need a dataset with example headlines of sufficient size. The first version of Churnalist uses these headlines as substitution medium. Churnalist version 2 also requires this dataset for fine-tuning GPT-2 on the headline generation task. In both cases, the size of the dataset determines how well the system will work. We hypothesize that both versions of Churnalist have different dataset requirements in terms of size, but we cannot quantify these requirements yet. In Churnalist version 1, the generative space is determined by the number of seed words S that the user finally marks as acceptable, and the total number of headlines H in the example dataset: Churnalist version 1 can create a maximum of $S \cdot H$ headlines. For version 2, we need *sufficient* headlines to teach GPT-2 the two aspects of the task: the definition of headlines, and the latent task operationalized with the two tags `<|object|>` and `<|headline|>`.

By using GPT-2 we have mitigated the language quality problems of Churnalist version 1, but introduced new problems that are inherent in using large scale language models, such as the need for computation power for fine-tuning and

sampling the language model. The fine-tuned language model that we used is relatively small in terms of size, only 400 Mb. After the fine-tuning step is complete, sampling the model for generation is less computationally expensive. However, generating outputs with `gpt-2-simple` is still slow, about 1 minute per prompt, although this might be because the open source implementation is not optimized for computation speed.

The confusion of GPT-2 when we include non-English words in the prompt also points to another problem. If we want to use our approach for generating text that includes game-specific words, such as ‘augmentation’ in the case of *Deux Ex: Human Revolution*, we need to teach the GPT-2 model the semantics of these words, for example by providing it examples of how these words are used in English text. It would be useful to fine-tune GPT-2 on additional game text, so that the game-specific words are incorporated in the language model prior to generation.

6.5 Conclusion

This chapter described a fine-tuned GPT-2 language model as alternative for Churnalist’s substitution module. We performed an error analysis of headlines generated with the two versions of Churnalist, which showed that using GPT-2 solves or mitigates most of the problems in outputs created by Churnalist version 1. Only 36 percent of headlines generated by version 1 were completely free of errors, in contrast to 79 percent of headlines generated by Churnalist version 2. Additionally, training the GPT-2 model turned out to be much faster and more straightforward than building a rule-based module from a combination of open-source NLP libraries.

However, the GPT-2 language model introduced new problems which did not occur in the rule-based substitution module. The substitution module of Churnalist version 1 is a white box system, whereas GPT-2, which is a neural language model, is black box. This means that text generation with Churnalist version 2 is not fully transparent, which makes it harder for Churnalist’s user to analyze the generation process if the outputs are of sub-par quality. Additionally, version 2 outputs sometimes converge on repetitive language, and the system has problems generating headlines from prompts containing non-English or infrequently occurring words. This is a serious drawback, since game narratives might contain many non-English words, especially games with a science fiction or fantasy setting. A possible solution to this problem is performing additional fine-tuning of GPT-2 on texts extracted from the video game in question. The game-specific words and their implicit semantics then become part of the language model. However, datasets with video game text are scarce, so this work-around might not always be viable in practice.

In the next chapters, we will continue this line of investigation. In Chapter 7, we will fine-tune GPT-2 on text from roleplaying video games, and use the

fine-tuned language model to generate new flavor text. In Chapter 8, we will address the scarcity of video game text corpora. We will investigate where we can find new datasets of video game texts, and present three new corpora that can be used for fine-tuning GPT-2.

NPC dialogue generation from annotated video game quests

In Chapter 6 we investigated how GPT-2 can be fine-tuned on a task-specific dataset annotated with special tags. We used the resulting model to generate flavor text for video games. In our case, we used GigaWord headlines to fine-tune a GPT-2 model for creating headlines that can serve as flavor text in games. In this chapter we continue this exploration, this time working with video game data from a commercial role-playing game as source material.

This chapter is based on the following peer-reviewed publication:

- Judith van Stegeren and Jakub Myśliwiec. “Fine-tuning GPT-2 on annotated RPG quests for NPC dialogue generation”. In: *Proceedings of the 16th International Conference on the Foundations of Digital Games (FDG) 2021*. FDG ’21. Online: ACM, Aug. 2021

7.1 Introduction

In this chapter we investigate the efficacy of Transformer-based models, such as GPT-2 [209] and BERT [85], for dialogue generation for quests in role-playing video games. Role-playing games are a particularly suitable application domain for natural language generation techniques, as this genre leans heavily on narrative [23]. The importance of narrative often gives rise to a large amount of in-game text. In many RPGs, the main game narrative is supplemented with side-quests, i.e. minor narrative threads that may be pursued by players for game advantage, but which have no impact on the main story [43, p. 44]. Most side-quest have the form of non-obligatory small errands, e.g. speak to a specific NPC, kill 10 monsters, fetch 3 ingredients for a potion, or collect 7 pieces of a treasure map. They tend to follow a fixed structure, so they lend themselves well to procedural generation. We use the massive multiplayer online RPG (MMORPG) *World of Warcraft* [39] as a case study for our research.



Figure 7.1: A player character in World of Warcraft Classic starts the quest ‘Dwarven outfitters’ by talking to an NPC named ‘Sten Stoutarm’. The quest-giver NPC is identifiable by a yellow exclamation mark above his head. The dialogue box on the left shows the quest title, flavor text in the form of dialogue, quest objective and the quest rewards.

Figure 7.1 shows a screenshot of World of Warcraft Classic, where a player is talking to a quest-giver NPC. The NPC offers the quest ‘Dwarven outfitters’, one of the first quests for Dwarven player characters. Wowhead is a website that provides a searchable database, internet forum, strategy guides and player character services for World of Warcraft. Players can look up properties of World of Warcraft quests, such as the starting location and rewards, in the Wowhead’s quest database. Figure 7.2 shows the Wowhead quest database entry for this same quest.

From Figure 7.2, we can see that, among other properties, quests have a title and a quest objective, i.e. the assignment that the player should complete to earn the quest reward, such as experience points, items or money. Typically, quests are obtained by the player through special quest-giver NPCs. These NPCs introduce the quest to the player and contextualize it in the game world with a few lines of dialogue. The dialogue lines with the quest’s backstory are a form of flavor text, i.e. decorative text that is not essential to the gameplay. If we remove the dialogue lines from the game, the quest is still playable, but the quest is no longer explicitly linked to the game world and the game narrative. We have investigated whether GPT-2 can be taught the lore and writing style of World of Warcraft, and whether we can use this model to generate flavor text for quests, given a title and an objective. There are already many systems which can generate video game narratives and quests in certain structures [153].

Title	Dwarven Outfitters
Objective	Sten Stoutarm would like 8 pieces of Tough Wolf Meat.
Requirements	Tough Wolf Meat (8)
Description	<p>What do we have here? You look as though you might need something to keep your hands warm, hm?</p> <p>I'll tell you what would help: a pair of nice, warm gloves. And, being the kind soul that I am, I'd be more than happy to provide you with a suitable pair. I've one condition, however.</p> <p>I need you to go get me some wolf meat. Nice arrangement, hm? You bring me some wolf meat, and I'll make sure you don't lose any digits to frostbite. Well, what do you say?</p>
Progress	Wolves giving you a bit of trouble? You'd do well to avoid the fangs and claws and other sharp bits, yes?
Completion	Ah! Wonderful. The wolf meat should do nicely. Oh, don't worry, <name>, I wouldn't forget my part of the bargain. Here, one of these should fit you.
Rewards	<p>You will be able to choose one of these rewards:</p> <p>Rabbit Handler Gloves</p> <p>Wolf Handler Gloves</p> <p>Boar Handler Gloves</p>
Gains	<p>Upon completion of this quest you will gain:</p> <p>80 experience</p> <p>100 reputation with Ironforge</p> <p>100 reputation with Gnomeregan Exiles</p>

Figure 7.2: The entry for World of Warcraft Classic quest 'Dwarven outfitters' from the Wow-head quest database. Description, Progress and Completion show three dialogues for the quest-giver NPC. These lines are shown to the player upon starting the quest, during the quest and after quest completion, respectively.

However, most of these systems use rule-based approaches to generate surface text for those stories [155]. We explore whether GPT-2 is a viable alternative for rule-based approaches, especially for generating quest surface text.

Our contribution is two-fold. We have created a fine-tuned GPT-2 model based on a dataset of World of Warcraft quests. We then show that this model can be used to generate new dialogue lines for quest givers in World of War-

craft, given a human-written quest objective and a quest title as a prompt. We have evaluated our approach by comparing hand-written texts from the same game with the outputs of our generator on the properties of language quality, coherence and creativity in a crowdsourcing experiment. Our code and dataset are available for re-use by other researchers, and a working demo of the trained model can be found online.

7.2 Related work

Our research is related to work that investigates procedural text generation for video games, which was discussed in Chapter 2, and neural language generation with GPT-2, which was discussed in Chapter 3.

7.2.1 Quest generation

Kybartas and Bidarra [153] discuss various examples of quest generation research in their survey paper about story generation. One of the early approaches is SQUEGE [199]. SQUEGE randomly selects quest templates, called patterns, from a database and then populates the template with appropriate information about characters, locations and items. Quests are represented as branching graphs, where player actions and other quest events are nodes, and the edges signify a temporal order. The authors of ReGEN [155] built on SQUEGE's approach and proposed a method for graph rewriting, which is capable of creating complex branching stories. However, the surface text of the generated quests is still created with a rule-based approach, which limits the variety of the language of the final artifact. Doran and Parberry [87] explored the use of context-free grammars in RPG quests. As rule-based models are generally not generalizable to other domains, these approaches risk becoming repetitive.

Ammanabrolu et al. [6] compared two generation approaches for quest generation in a text-based cooking game: one based on Markov chains, and one based on neural generation. The authors used a fine-tuned GPT-2 model to generate surface text, i.e. cooking instructions, for their generated quests. Crowd-source workers were asked to play two generated quests and evaluate the quests on the properties of perceived creativity and coherence. The authors found that neural generation offers more value and greater coherence, whereas the Markov model produced quests that are more surprising and novel. They also noted that the neural generation approach requires less domain knowledge and is potentially more generalizable to other domains.

7.2.2 NPC dialogue generation

In World of Warcraft, the flavor text for quests consists of quest-giver NPC dialogue. Other researchers have investigated techniques for NPC dialogue gener-

ation, although we have not yet seen dialogue generation for games based on Transformers such as GPT-2. Walker et al. [280] generated dialogue for the prototype role-playing game *SpyFeet*, based on statistical machine learning models trained on film character dialogue. Ryan et al. [229] presented a method for annotating human-authored dialogues, so that different parts of these dialogues can be recombined to form new dialogues. The authors tested their approach on dialogue of the social AI engine system *Comme il faut (CiF)*, used in the game *Prom Week*. *Tracery* [70], a tool for generative grammars, has been used in various games to generate in-game text, including dialogue. Ryan et al. [223] created *Expressionist*, an authoring tool for generating in-game text at runtime. The tool is based on context-free grammars with added markup labeling. Users can use user-chosen tags to annotate non-terminal symbols with arbitrary meta-data, which gives new expressive power to a context-free grammar. Lessard et al. [164] used *Expressionist* to generate dialogue for their resource management game *Hammurabi*.

7.2.3 Other neural text generation for video games

A final notable example of neural text generation for video games is the game *AI Dungeon* [282]. *AI Dungeon* acts as a cross-over between a virtual tabletop *Dungeon Master* and a classic text adventure game. Text adventure games generally consist of a pre-written narrative, and accept input from the player in the form of natural language text, such as ‘open door’, ‘pick up sword’ or ‘fight grue’. *AI Dungeon* is similar to these classic text adventures in that it accepts natural language input from the player. However, instead of following a pre-written story, a neural language model extends the story on the fly, using the user input as prompt. Earlier versions of *AI Dungeon* used GPT-2¹ fine-tuned on text from *Choose Your Own Adventure*-style games.

AI Dungeon’s underlying neural model has some similarities to our approach: both use a fine-tuned version of the pre-trained GPT-2 model, trained on structured data from narrative games. However, *AI Dungeon*’s approach differs from ours in that it was finetuned on a more generic dataset, spanning different language styles, narrative genres and topics. Our GPT-2 model was trained on homogeneous data, namely quests from *World of Warcraft*. Furthermore, the goals of both language models are fundamentally different. *AI Dungeon* is meant to function similar to a *Dungeon Master*. In table-top games, *Dungeon Masters* should be able to respond to any ‘input’ that the players throw at them, within the rules of the game. As a result, *AI Dungeon*’s language model should be able to deal with a huge variety of inputs. By design, the model can be used for text generation without any human supervision; text is generated, and immediately presented to the player. Our language model, on the other hand, is meant as an authoring

¹<https://pcc.cs.byu.edu/2019/11/21/ai-dungeon-2-creating-infinitely-generated-text-adventures-with-deep-learning-language-models/>

aid for game writers. The idea is that generative language models can be used to do the heavy lifting in creating new content, something that may benefit game development companies as the playable content for video games, especially for open-world RPGs and MMORPGs, increases. Outputs are not meant for direct inclusion in a game; instead, they should be checked and possibly modified by a human writer.

7.3 Method

We want to build a text generator that, given a quest title and objective, can generate dialogue lines for a quest-giver NPC. Our approach is fine-tuning a pre-trained GPT-2 language model on a dataset with quest information, and using this fine-tuned model to generate NPC dialogue. In this section, we start by discussing our training data, which is followed by a description of how we fine-tuned the model and used it to generate new dialogue.

7.3.1 Data

When we started this research project, we had not chosen a particular game to work with yet. To fine-tune GPT-2's pre-trained model, we needed a dataset with quest data. Specifically, we wanted to finetune the model with training data that has been annotated with tags that describe the structure of the data, following the approach of Zellers et al. [290]. Consequently, we needed a dataset that included NPC dialogue, and quest titles or quest objectives to contextualize the NPC dialogue. We estimated that using data from Massive Multiple Online RPGs (MMORPGs) would be a good idea, since these generally have more playable content than offline games and single-player games. These games tend to have large player communities, which could mean that information about quests would be obtainable online, for example from fan-websites. Initially, we identified three possible sources for a dataset that fulfills this requirement, with data from popular MMORPG games:

1. Wowhead quest database² which contained 24,981 quests from World of Warcraft [39];
2. Destiny tracker quest database³, containing 680 quests from *Destiny* [50];
3. EVEinfo.com mission database⁴ with 310 *EVE Online* [58] missions

It is possible to use GPT-2 without any fine-tuning, or fine-tune GPT-2 with relatively small datasets. However, since we wanted to teach GPT-2 a particular

²<https://www.wowhead.com/quests>

³<https://db.destinytracker.com/d1/quests>

⁴<https://eveinfo.com/missions/>

Title
The Wayward Crone
Objective
Confront Helena Gentle in her home outside of Fallhaven
Description
The ledger indicates that an old woman named Helena Gentle recently took up residence in a house down the road from the town. The villagers' writings point to her being involved somehow with a variety of maladies that struck the village recently. It's possible that she may know what's behind this spell, if it hasn't afflicted her as well.

Figure 7.3: Quest 'The Wayward Crone' from the World of Warcraft quest database

structure (i.e. a quest consists of a title, objective and dialogue), the dataset should be large enough for the model to capture the structure of the training data. A dataset with a few hundred datapoints is probably not large enough to change the pre-trained model in a significant way, and thus to generate outputs that follow the structure of a quest. We estimated that, individually, the Destiny and EVE Online datasets were probably too small to use as training data. A possible solution for this is combining the three datasets into one large dataset. However, although their structure is the same, the three datasets are too different to combine successfully. EVE Online and Destiny are science fiction games, whereas World of Warcraft is a fantasy game. Destiny quests have very short quest descriptions in the form of a quote, as opposed to World of Warcraft and EVE Online quests, which have longer dialogues as quest description. EVE Online's quest objectives lack in variety, as quests fall in one of only four categories.

During preliminary testing, we found that fine-tuning on one homogeneous dataset leads to higher quality output. Since the Destiny and EVE Online datasets are relatively small (only a few hundred quests compared to WoW's 24,000 quests), we decided to use only the World of Warcraft quest database. Figure 7.3 shows an example quest from this dataset.

7.3.2 Training

We added tags to the dataset that describe the structure of the datapoints. Figure 7.4 shows the tags we used to annotate the quests in the World of Warcraft database, together with a concrete example datapoint from our training set. GPT-2 uses these tags to learn the structure of the datapoints in the training set. During generation, we can use these tags to steer GPT-2 towards a particular output structure: if we provide a partial datapoint (title and objective) as a prompt, GPT-2 will expand it with the part we want to generate (quest description, i.e.

Structure

```
<|startoftext|> [quest title]
<|obj|> [quest objective]
<|text|> [quest description]
<|endoftext|>
```

Example datapoint

```
<|startoftext|> In Dire Need
<|obj|> Hear out the Council of Six in the Purple Parlor.
<|text|> The ill tidings you bear only increase the concerns the Council
has been having. I know you have your hands full with the Tomb of
Sargeras. Make no mistake, the battle for the tomb remains our top
priority—but we must not think the Legion foolish enough to rely on brute
force alone. They are far more cunning than that. Please, hear us out.
<|endoftext|>
```

Figure 7.4: Structure of datapoints in our training set. Quests are annotated with special tags that denote the quest title, objective and description (NPC dialogue).

```
<|startoftext|>In Dire Need<|obj|>Hear out the Council of
Six in the Purple Parlor.<|text|>
```

Figure 7.5: Prompt for the quest ‘In Dire Need’ from World of Warcraft. The prompt is used to generate NPC dialogue for the quest-giver NPC.

7

dialogue). We used the second-largest GPT-2 language model with 774 million parameters, which we deployed in a Google Colab environment, a Jupyter notebook environment in the cloud that comes with computation time on GPU. The basis of our code is Max Woolf’s `gpt-2-simple`⁵ project. Fine-tuning GPT-2 on our annotated training data took approximately 4 hours using an NVIDIA Tesla V100 GPU.

7.3.3 Generation

We use a quest title and objective, together with their start tags, and the starting tag for NPC dialogue, i.e. `<|text|>`, as prompt for the generator. For an example prompt, see Figure 7.5. The generator creates new text by sampling the fine-tuned language model for follow-up tokens. The model continues generating text until it has generated a pre-determined maximum amount of tokens. The output is cut off at the first `<|endoftext|>` token.

⁵M. Woolf. GitHub - minimaxir/gpt-2-simple: <https://github.com/minimaxir/gpt-2-simple>

Title	In Dire Need
Obj	Hear out the Council of Six in the Purple Parlor.
Original dialogue	
The ill tidings you bear only increase the concerns the Council has been having. I know you have your hands full with the Tomb of Sargerass. Make no mistake, the battle for the tomb remains our top priority—but we must not think the Legion foolish enough to rely on brute force alone. They are far more cunning than that. Please, hear us out.	
Temp	Generated dialogue
0.1	The Council of Six has been trying to get a word in, and I'm not sure they'll be able to get one out loud in the Purple Parlor. I'm sure they'll be waiting for you in the Purple Parlor, if you're interested in hearing what they have to say.
0.2	The Council of Six is in dire need of your help. They have been trying to convince me to join their cause, but I refuse to be manipulated. I have no love for the Lich King, but I will not be manipulated into helping them. I have heard that they are holding a meeting in the Purple Parlor. I would like to hear out the Council of Six before I make a decision.
0.3	The Council of Six has been trying to get a word in, but we've been so busy with the attack on the Broken Shore that we haven't had time to hear from them. I'm sure they're worried about their people. I'd like you to go to the Purple Parlor and hear what they have to say. I'm sure they'll be worried about their people, too.
0.4	We are at war with the Legion. The time to strike is now. We need to hear out the Council of Six. They are the ones in charge of running this place and we must listen to them. I will be right behind you.
0.5	The Council of Six is a council of the six. The leader of the Council of Six is called the "Crown Prince." The Crown Prince has a council with a council. The Council of the Six is a council of the six. The Council of the Six is a council of the six. The Crown Prince is a king. The Council of the Six is a council of the six. The Council of the Six is a council of the six.
0.6	And with that, we have a perfect opportunity to drive a wedge between the Horde and the Alliance. I have a feeling that the Council will be willing to overlook the fact that the orcs have been tricked. Bring the Council to the Purple Parlor as soon as you can.
0.7	Hear out the Council of Six in the Purple Parlor!
0.8	Enter The Purple Parlor and you will find me, the Council of Six, and Thomas Paine. I believe you already know who Paine is. Go to the council. Go, . Find them and hear out their concerns.
0.9	The Council has a rallying point all across The Dreamgrove. There is an empty throne oak in the purple parlor, and a creature named Amalia should be there. Unfortunately her days of being known as a Gallywix-level concession vendor are done.

Figure 7.6: Quest descriptions generated by our finetuned GPT-2 model with temperatures ranging from 0.1 to 0.9. All outputs were generated with the prompt from Figure 7.5.

Temp	Generated output
0.7	Your aid is needed here, not in the Garden, but in the Purple Parlor, where the Council of Six holds its meetings. Among those attending are... well, let's see. It wouldn't do for me to name them, would it? The ones that might know are... well... let's see... The Council of Six is worried about the new naaru. I can't leave them in this place. I need to go back to the Hall of the Guardian. Hear out the Council of Six in the Purple Parlor.
0.7	Master Ashwall wants you to head to the Purple Parlor in Zoram'gar Outpost and inform them of what you have learned.
0.7	Hear out the Council of Six in the Purple Parlor!
0.7	The Alliance needs your help! Many of our brothers and sisters have been placed in peril. Please hear out the Council of Six, we need to know how to handle this situation.
0.7	This treaty was reached in less than 24 hours despite the turmoil that was sparked by the betrayal of Ormok. If you haven't heard yet, the Warchief dispatched Blastdancer Kalaran to deal with you personally. I won't be fooled again. Do... do not do... this.
0.9	The Council has a rallying point all across The Dreamgrove. There is an empty throne oak in the purple parlor, and a creature named Amalia should be there. Unfortunately her days of being known as a Gallywix-level concession vendor are done.
0.9	Hear me out! We're in dire need of help. Our explosive ordnance is getting dangerously low on both of our runs. I know sylvari aren't usually known for their industriousness, but this barrel needs tolled and our prophet needs his prayers answered. Who knows, maybe you can help us out.
0.9	I will not keep you waiting any longer. I can offer you an undersea vantage that will let you see Bevel's will done. Please, take advantage of this juxtaposition of nature and technology. Meet with the Council of Six in the Purple Parlor. When you are prepared, we will speak again.
0.9	Greetings, and welcome to the Emerald Sanctum. It is with great pleasure that the Council of Six can present to you its members. Peerless Master of the Arena Shantelos, Peerless Trainer of Arena Mastery, Champion of the Lonely theurge, and Peerless Master of Evodyne Industries are bound by the will of the Council of Six. Do welcome your retinue to the circle of gladiators.
0.9	The felbat is a vicious predator, and the last thing we need in the Broken Isles is another of our kind. Report to the highest ranks of the Kirin Tor and tell them of our plight. The Broken Isles is a dangerous place where only the fittest survive. I'll designate a short period of time for them to consider our concerns. I should take my leave. Stay well.

Figure 7.7: Quest descriptions generated by our finetuned GPT-2 model with a temperature of 0.7 and 0.9. All outputs were generated with the same prompt from Figure 7.5. This table illustrates the wide range in quality in artifacts generated with the same generation parameters.

We can influence generation by changing the temperature parameter. Temperature is a number between 0 and 1 that determines the randomness of completions. Lower temperature leads to less randomness, and might lead to repetitive outputs. A higher temperature leads to more randomness, and consequentially unexpected output, which influences output properties such as coherence, language variety, and interestingness.

We tried to find the best value for the temperature parameter empirically, by manually inspecting outputs of the model that were generated with various temperatures. Figure 7.6 shows example outputs for temperatures between 0.1 and 0.9. Since temperature determines the predictability of outputs, choosing a too low temperature leads to repetitive language in the output text. The example in Figure 7.6 that was generated with a temperature of 0.5, demonstrates this, as it consists mostly of words and phrases that already occurred in the prompt title. With a temperature of 0.7, the generator created a quest description that was an exact copy of the quest objective. The outputs with an even higher temperature of 0.9 were more unexpected and contained the most variety, thus we chose this temperature for generating our evaluation outputs.

Qualitative inspection of the generated quests shows that the fine-tuned GPT-2 model outputs artifacts that are highly coherent with the prompt. In Figures 7.6 and 7.7, multiple generated quest descriptions contain references to ‘the Council of Six’, a name that appears in the quest objective of the prompt.

Interestingly, our model has learned the structure so well that it often generated entire quests by itself. They follow the exact same structure as our training data, i.e. title, objective, and description, with each part delimited by tokens. Because the training set was taken from World of Warcraft, the generated quests contain words, phrases and names that are reminiscent of the lore of the game world.

7.4 Evaluation

7.4.1 Experiment design

We conducted an online survey to evaluate the outputs of the generator. Answers to the survey were collected via GoogleForms. The survey was shared with bachelor students of the University of Twente. To recruit additional participants, a link to the survey was also shared online. Participants received an information brochure and indicated their consent before partaking in the study. They received no remuneration for participation in the study. The evaluation experiment was approved by the Ethics Committee of the Faculty of Electrical Engineering, Mathematics and Computer Science at the University of Twente.

Participants were presented with 20 quests, which were ordered randomly. Every participant rated the same 20 quests. 10 quests were randomly chosen from the World of Warcraft dataset, and 10 quests were generated with our fine-

tuned GPT-2 model, using a randomly chosen title and objective from the training set as prompt. To keep the questionnaire brief, we decided beforehand to only evaluate outputs that had been generated with a single temperature value, namely 0.9. The generated quests were generated in one go, and not cherry-picked from a larger set of outputs. Figures 7.8 – 7.12 list the quests that were used in the evaluation survey. For each quest the participants were asked to indicate how much they agreed with 5 statements on a 7-point Likert scale. Participants rated all 20 quests on the following 5 statements, for the properties language quality, coherence and creativity:

1. The description makes use of correct English.
2. The goal is clear from the description.
3. The description is written in a novel way.
4. The description is surprising.
5. The description is valuable.

For language quality, participants were asked to indicate their agreement with Statement 1. To measure the perceived coherence between the quest objective and the NPC dialogue, we used Statement 2. We used Boden's breakdown of creativity, similar to the approach of Ammanabrolu et al. [6]. According to Boden [41], creativity can be measured in terms of novelty, surprise and value (Statements 3–5). Based on preliminary testing of the survey questions we decided to substitute the word 'valuable' with the word 'creative', as participants found the question about value too confusing.

The survey contained two additional questions about the participant's level of English on the CEFR scale, and their level of familiarity with role-playing video games on a scale of 1–5.

Trouble at the Docks.

Crane Operator Bigglefuzz in Ratchet wants you to retrieve the bottle of 99-Year-Old Port from Mad Magglish who is hiding in the Wailing Caverns.

Original

Oops! Had a little mishap, as you can see! The morning cargo shipment from Booty Bay was huge. Huge I say! Never seen so many crates. Well, I was trying to expedite things so I started unloading a bit more than I could handle. That's when everything came tumbling down. Before I knew what was going on, I saw that menace, Mad Magglish, go running off with a bottle of 99-year-old port. Chased him all the way to Wailing Caverns. No way I was going in there. Maybe you're brave enough to find him?

Generated

My wife took me to the Docks last week to appear on one of those 'news' shows some goblin or goblin-looking creature is running. We were going to get a discount as it was a special occasion, but the first news report I saw, Mad Magglish was already in the caves and stealing all the time. He's the one who gave that old pirate outfit to his son, who is basically a glorified beer sailor now. It's never safe to take old pirates' clothes, they usually have valuable chestnuts inside.

Proof of Strength: Ogron Horn.

Take the proof to Limbflayer at Beastwatch in Gorgrond.

Original

This horn is proof of your strength at slaying ogron and your gladiator friend has born witness! Take the horn back to Limbflayer and claim your accolades!

Generated

Take this to Limbflayer. I can't guarantee that he'll believe it, but I'm confident that he can convince it's none of my business. I've seen the certain beasts growl, so perhaps he'll take it as a personal insult that I'm actually here. But I bet he won't. You've proven your mettle. You deserve the horn.

Figure 7.8: Generated quest descriptions that were used in the evaluation survey, together with the original dialogue as it appears in World of Warcraft. The quest title and objective were selected randomly from the WoW dataset and used as a prompt for the generator. All outputs were generated with a temperature of 0.9.

Rituals of Power.

Bring the book Rituals of Power to Tabetha in Dustwallow Marsh.

Original	Generated
Hm... now that I think about it, I don't have my book! You see, I had some gambling debts and to pay them off I sold the last copy of Rituals of Power to a member of the clergy... A member of the Scarlet Brotherhood. He must have taken it to their library in the Scarlet Monastery in Tirisfal Glades! If you want that book, then you'll have to go to the monastery to find it. And sorry about all the trouble. I'm not myself these days...	Something in this book seems to be purging the memory of the demi-god, Rukhmar. I'm not sure what purpose it could be serving, but it's disabled and cannot be used again until the ritual is prepared. You'll need some materials if you plan on performing the ritual. Bring me a copy of the Rituals of Power and I'll create the ritual for you. It should only take you a few minutes. Travel to Orgrimmar and speak with my assistant, Tabetha.

Spray it Again.

Bring the Box of Bugkill Fumigators to Marcy Curtainfire inside the Ban'Thallow Barrow Den.

Original	Generated
I'm gonna start setting this up. Next one goes to Marcy, further down the den. Head down this corridor here and look for her in the next main chamber.	There you have it. Of course the crystal started glowing before the attack. I will have to perform some further research in the laboratory, but first I'm going to need to go set the rest of the explosives. You seem to know where that stuff goes. It's unfortunately hanging out in a safe in the back of this barrow den. If you want me to retrieve it, all you have to do is give me your key. Of course, I won't charge you for the hassle.

Figure 7.9: Generated quest descriptions that were used in the evaluation survey, together with the original dialogue as it appears in World of Warcraft. The quest title and objective were selected randomly from the WoW dataset and used as a prompt for the generator. All outputs were generated with a temperature of 0.9.

The Bandit Warlord.

Slay Warlord Ihsenn.

Original

Our law demands that those behind this attack face proper retribution. The band of bandits' ringleader is a man known as Warlord Ihsenn. You will find him in the far west of the desert. Make sure his transgressions do not go unpunished.

Generated

There is an enemy that exceeds even Neferset that the guards are too weak to handle, he is called Warlord Ihsenn. If you kill him, perhaps I can retake Felfire Hill.

A Well Rounded Defense.

The Facet of Keanna stored inside Keanna's Will wants you to gather 20 Arcane Residues from the Arcane Anomalies in Karazhan.

Original

Offensive magic was my area of expertise... that much I can remember. I'll do my best to add my arcane channeling powers to yours. I'll need more raw power than I possess right now, however. The arcane anomalies that roam the upper levels of the guardian's library would make excellent targets. Gather their arcane residue, we'll need lots of it.

Generated

Our people have been fighting an endless war against necromantic attacks, and we cannot afford to stagger their resistance. I want you to find and gather arcane residue from arcane anomalies throughout the city. I want you to take this branding iron and release the broken into the city to get a better look at the matter. You will soon find that they are everywhere, and you will find that they generate a powerful explosion.

Figure 7.10: Generated quest descriptions that were used in the evaluation survey, together with the original dialogue as it appears in World of Warcraft. The quest title and objective were selected randomly from the WoW dataset and used as a prompt for the generator. All outputs were generated with a temperature of 0.9.

Fashion History and a Philosophy of Style.

Learn from Eliezer Hammerbeard how to create “flair and drama” in your wardrobe.

Original	Generated
Greetings, <name>. I am a fashion historian and I have astonishing advancements in tailoring, leatherworking, and blacksmithing that I must share with you. Why? A true legend is only as good as their ability to form an epic silhouette on the battlefield and–let’s be honest–this is not where you shine. Speak to me again if you wish to learn.	My job as a tailor is to take your stuff and make it look good. The way I see it, there are two ways to get people to like you: 1. Show them how good you are and 2. Show them how good you are through stylishly made clothes. I will cover both in this class. Please come back to me after this class and I will demonstrate how to take ye best pieces and make ye clothes that people want to buy.

In Dire Need.

Hear out the Council of Six in the Purple Parlor

Original	Generated
The ill tidings you bear only increase the concerns the Council has been having. I know you have your hands full with the Tomb of Sargeras. Make no mistake, the battle for the tomb remains our top priority–but we must not think the Legion foolish enough to rely on brute force alone. They are far more cunning than that. Please, hear us out.	The Council has a rallying point all across The Dreamgrove. There is an empty throne oak in the purple parlor, and a creature named Amalia should be there. Unfortunately her days of being known as a Gallywix-level concession vendor are done.

Figure 7.11: Generated quest descriptions that were used in the evaluation survey, together with the original dialogue as it appears in World of Warcraft. The quest title and objective were selected randomly from the WoW dataset and used as a prompt for the generator. All outputs were generated with a temperature of 0.9.

Balance of Power.

Speak to Romuul's assistants to collect an Optical Disc, Conducting Crystal, and Romuul's Wrench.

Original

The Prophet needs to locate the arkonite crystals, but in order to do that, I'll need to fix the projection discs on that platform. Speak to my assistants around the observatory and bring me an Optical Disc, a Conducting Crystal, and my wrench.

Generated

Sound difficult? We ought to be able to get these books back to Kumul without too much trouble. Don't worry though, I'll make it worth your while. First, we need to hide one of the books. We can do that by giving it a nice, "Hey, you know what? I got this one figured out, go figure it out for me!" put down. Then we need to get the books off our chests before they get confiscated. There will be a bunch of assistants nearby to familiarize them with your face.

The Road to Purgation.

Accompany Kingslayer Orkus to Purgation Isle.

Original

The time of vengeance has come, <name>. Kasha can once more fly, thanks to you. She will take us to Purgation Isle so that we may confront the Stormpike dwarf general, Vanndar Stormpike, and his captain, Balinda Stonehearth. BOTH will pay for what they have done. Are you ready? Orkus cannot do this alone.

Generated

It's that time again. You've earned it. You can finish off these hereers before they make any strikes. Let's go party.

Figure 7.12: Generated quest descriptions that were used in the evaluation survey, together with the original dialogue as it appears in World of Warcraft. The quest title and objective were selected randomly from the WoW dataset and used as a prompt for the generator. All outputs were generated with a temperature of 0.9.

7.4.2 Results

32 participants filled in the online survey. For each participant the scores were grouped into 2 categories: generated and hand-written. We calculated the average for each property, and conducted a sign test on the median as none of the differences showed a normal distribution. Figure 7.13 shows the average score per category for each of the five statements. Three properties, namely language quality, coherence and novelty, were shown to be statistically significantly worse in descriptions generated by the model, with $p < 0.01$. The two other properties,

surprise and creativity, did not show significant differences even with $p < 0.05$.

Even though the generator on average performs worse than human writers, the results are encouraging because the differences in ratings are not very large. The scores of generated quest descriptions for surprise and creativity are close to those scores for the hand-written descriptions. This may be because of the high temperature with which the quest description were generated. Similarly, the differences in language quality and coherence might be due to the high unexpectedness of the generated texts, due to the high temperature setting used during generation. We analyzed whether participant's quest ratings were correlated with the answers to the two questions about English proficiency or familiarity with role-playing video games, but that was not the case.

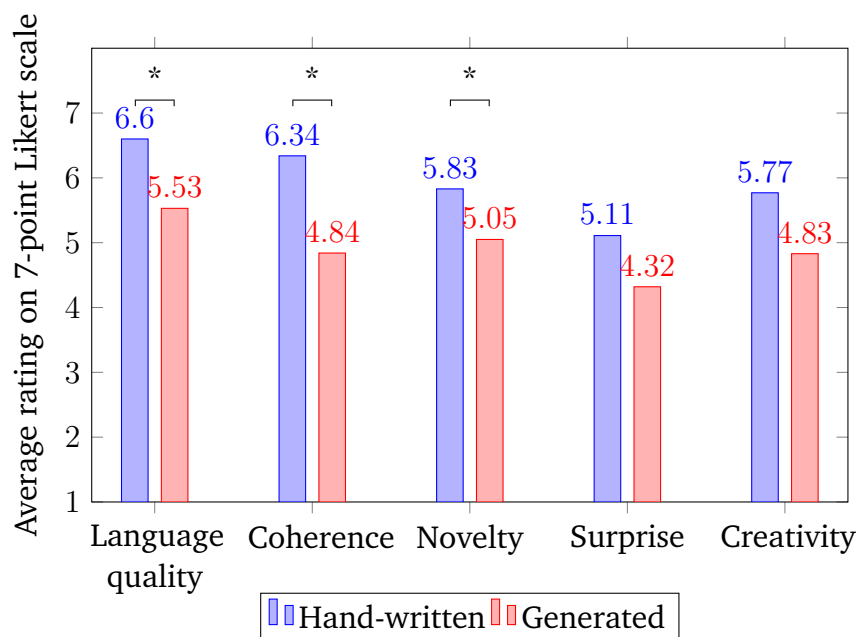


Figure 7.13: Average rating across evaluation properties rated on a 7-point Likert scale, for hand-written and generated quest descriptions. We collected ratings from 32 participants. The * denotes statistical significance with $p < 0.01$.

Figure 7.14 shows the distribution of the ratings across the different properties. For all properties, the generator is capable of generating outputs that score higher than the median rating for hand-written texts. Sometimes the quality of generated quest descriptions even exceeds that of handwritten examples, such as in the case of the properties coherence and surprise. The inconsistency in quality of GPT-2's outputs suggests that we should take a cherry-picking approach to generation. Instead of generating one description, we could use the system to generate multiple outputs for the same prompt, and select the best one. If we can come up with a automatic metric for judging the quality of generated artifacts, we could also use an automated generate-and-test approach. If the generator should be used without any input from the user, generating texts with a lower

temperature (e.g. 0.7) might improve the language quality and coherence of the output, possibly at the detriment of creativity and surprise. As we can see in Figure 7.6, texts generated with a lower temperature seem to be more consistent in both spelling and informative content.

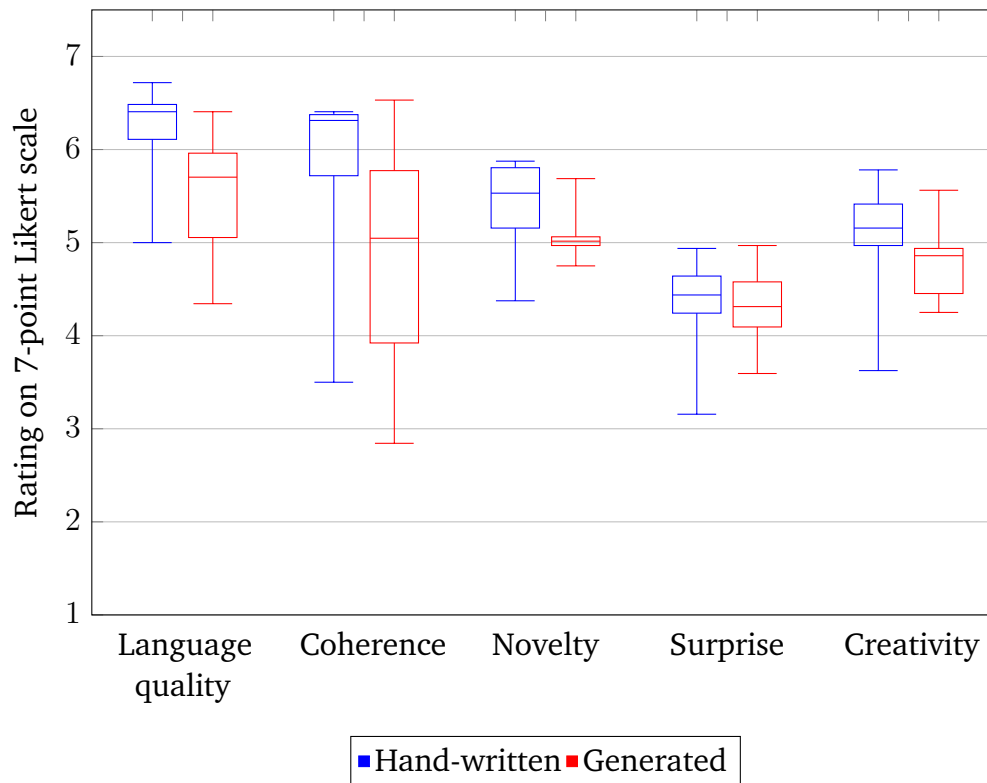


Figure 7.14: Distribution of ratings per evaluation property for hand-written and generated quest descriptions. 32 participants rated each property on a 7-point Likert scale.

7.5 Discussion

Although the fine-tuned model creates outputs that score worse on all three properties, the described approach is a viable option for text generation for games. Teaching GPT-2 a specific output structure by delimiting the training set and prompts with unique tokens proved to be a success. Although the quality of the results may be inconsistent across multiple outputs generated with the same prompt, as shown in Figure 7.7, the outputs indicate that GPT-2 can learn the structure and linguistic style of World of Warcraft quests. Moreover, the ratings distributions show that the generator is capable of creating outputs that are on par with human-written dialogues. A difference between human writers and our fine-tuned model is that the generator can easily create large numbers of quest descriptions from the same prompt. Once a GPT-2 language model has been

fine-tuned, creating large volumes of embryonic quest descriptions is fast and low-cost. Letting a human user cherry-pick outputs with the highest quality, or modify the most creative outputs, seems a feasible alternative to writing new flavor text for RPG quests by hand. Finding the optimal generation temperature could lead to additional improvements in output quality.

GPT-2 original training data WebText has an interesting side-effect. Since GPT-2 was pre-trained on web text, it is likely that generated texts will contain references to internet culture, such as memes, jokes and pop-culture references. Some manually-written World of Warcraft quests also contain references to pop-culture.⁶ Players that read the generated dialogue might conclude that the references were introduced on purpose by the generator.

A weakness of the discussed approach is that the fine-tuned GPT-2 model does not yet generalize to other role-playing games. Since we fine-tuned GPT-2 on World of Warcraft data, the resulting outputs contain references to the lore of World of Warcraft, and the outputs are not easily transferable to other game worlds. However, we imagine the model can be adapted to generalize in a few ways. Firstly, we could try to create a more general model by preprocessing the training data: if we substitute all named entities by a [LOCATION], [NAME] or [FACTION] tag prior to fine-tuning, the outputs will become a lot more generic. This process, called delexicalization, can be used to normalize datasets prior to training a language model for neural generation, e.g. Castro Ferreira et al. [56]. The placeholders in the data can then be filled with names and location of another game world, either by hand or automatically. For the automatic substitution, we could use a language model that performs well on the cloze test.

We might also try to translate outputs from one game to another using techniques from neural machine translation in a post-processing step. For example, we could try to translate game specific terms using distributional semantics, i.e. by building a vector space of word embeddings using words from two different games. For example, if it turns out that ‘Sith’ (Star Wars games) and ‘The Burning Legion’ (World of Warcraft) are often used in the same context, these words might appear close together in the resulting vector space. We could use this information to substitute the word ‘The Burning Legion’ for the word ‘Sith’ in generated outputs.

We could also fine-tune GPT-2 on a heterogeneous dataset from different video games, annotated with additional tags that denote the game world or game genre of that datapoint. These tags could then be used to steer the model towards outputs for a specific genre or game. A prerequisite for this approach is a varied dataset that is large enough to capture significant differences in style and content between games. Although no datasets of this type were available when we performed this research, we will discuss work in this direction in the next chapter.

⁶<https://www.wowhead.com/news/pop-culture-references-in-wow-npcs-and-quests-from-games-and-music-228084>

Other future work in NLP for video games could explore generation in the opposite direction from what we did in this chapter, i.e. take dialogue lines as prompt to generate a quest title and objective. This can be seen as a form of text summarization, as the quest objective is often grounded in the dialogue.

Evaluating the impact of different temperature settings in a more rigorous evaluation could also be useful, although it is doubtful whether these findings would easily generalize to models trained on other (game) data.

Following the approach of fine-tuning GPT-2 on a training set annotated with tags that describe the structure of input texts, we should explore whether we can add additional annotation tags to the training set, for example for capturing expressed sentiment and NPC–player relations. These tags could then be used in prompts, so that we can exercise more control over generation by guiding the generator towards outputs with desired properties. Using the largest GPT-2 model (1.5 billion parameters) might improve the language quality of the generated examples. However, if we start using larger pre-trained language models, we must also investigate whether the size of the training set should be increased proportionately, to prevent the larger model from undertraining on the annotated game data. It would also be interesting to find out how large a dataset of game texts should be, before it can be used to teach GPT-2 the structure and linguistic style of game texts.

7.6 Conclusion

In this chapter we built on the work from the previous chapter, in which we fine-tuned a GPT-2 language model for creating flavor text in games. In this chapter we continued this exploration, this time working with video game data from a commercial role-playing game as source material. We created a dataset with text from World of Warcraft quests, and used an annotated version of this dataset to fine-tune a GPT-2 model. We then used this model to generate new NPC dialogue for quest-giver NPCs in World of Warcraft. To evaluate our approach, we compared generated quests to human-written quests on the properties language quality, coherence and creativity. The evaluation results showed that the generator performs worse than human writers. However, the results are encouraging because the differences in ratings are not very large. The scores of generated quest descriptions for surprise and creativity are close to those scores for the hand-written descriptions. Our results also showed that, in some cases, generated quests were rated higher than human-written quests on the properties ‘content’ and ‘surprise’. Using GPT-2 to generate a large volume of draft quests and letting a human editor curate the results might be a viable alternative to writing new quests fully by hand.

The system described in this chapter is open source. The training data and generator used for the experiments can be found as a Google Colab environment at <https://jakub.thebias.nl/research/QuestGen/colab/>.

Section 7.3.1 described our efforts to find a dataset with video game text that was suitable for fine-tuning GPT-2. It was difficult to find such a corpus, since it had to be large enough to effectively fine-tune the smallest model of GPT-2, contain enough varied content to yield interesting results during generation. To use our approach with special tags, we also needed a dataset with enough contextual information to annotate the datapoints; a text dump of video game string literals would not have sufficed for this reason. In the next chapter, we will address the scarcity of video game corpora. We will discuss what the requirements are for high-quality datasets, and investigate where we can find additional video game corpora.

Collecting video game corpora

In the previous chapter, we saw that using GPT-2 can be a viable and low effort approach for generating flavor text. If we want to train Transformer-based language models for generating flavor text, we need datasets with examples of flavor text. However, datasets with video game text are scarce. In this chapter, we investigate what the requirements are for datasets for training GPT-2, and how we can create new suitable datasets with in-game text from commercial video games.

The main contributions of this chapter are:

- A list of quality criteria for video game text corpora
- An analysis of where such corpora can be obtained
- Three datasets with game texts from popular video games, together with examples of how these corpora can be used in research

This chapter is based on the following publications:

- Thérèse Bergsma, Judith van Stegeren, and Mariët Theune. “Creating a Sentiment Lexicon with Game-Specific Words for Analyzing NPC Dialogue in The Elder Scrolls V: Skyrim”. English. In: *Workshop on Games and Natural Language Processing*. Marseille, France: European Language Resources Association, May 2020, pp. 1–9. ISBN: 979-10-95546-40-5. URL: <https://www.aclweb.org/anthology/2020.gamnlp-1.1>
- Judith van Stegeren and Mariët Theune. “Fantastic Strings and Where to Find Them: The Quest for High-Quality Video Game Text Corpora”. In: *Proceedings of the 2020 Workshop on Intelligent Narrative Technologies*. Oct. 2020

8.1 Introduction

Most text generators for video games still use techniques from more than fifty years ago, such as pattern-matching and string replacement. Newer methods often need large amounts of data for training, but text corpora sourced from video games are scarce. High-quality video game text corpora can be used as resources for many types of research, including but not limited to text generation for games. Moreover, if multiple researchers use the same corpus, this facilitates comparison of their results and can lead to research advancements, for example via the organisation of shared tasks.

In this chapter, we will show that there are many possibilities for creating new video game corpora – if researchers know where to look, and spend some time re-assembling in-game context of collected texts. In Section 8.2, we review existing work on video game text corpora, and their applications in games and NLG research. We then discuss the quality requirements for video game text corpora in Section 8.3. In Section 8.4 we will describe two sources of game text data: game files and online fan-communities. In Section 8.5, we will describe three datasets with game texts. The datasets were sourced from three commercial role-playing games: *Torchlight II* [222], *Star Wars: Knights of the Old Republic* [36] and *The Elder Scrolls* [31]. Finally, we discuss possible applications, limitations and ideas for future work in Section 8.6.

8.2 Related work

8.2.1 Datasets in games and AI research

Inspired by the NLP field, where sharing corpora for shared tasks is the norm, games and AI researchers have started to share datasets to bring the research field forward as well. Although they are termed corpora, a term normally reserved for collections of text, most of these datasets do not contain game text, but other types of game assets. Summerville et al. [259] collected a corpus of video game levels in three annotation formats, which can be used for procedural content generation and level design analysis. This corpus can be used for corpora-based procedural content generation, design analysis and style transfer, e.g. reimagining a level from one game in a (game level) style of another game, or generating variations of an abstract level in multiple game styles.

Anderson and Smith [7] created the Undergraduate Games Corpus, a dataset of games to support research on machine perception of interactive media. The games were sourced from student projects in an undergraduate game development program. The dataset includes full games (including source code and game assets) and metadata from several game platforms: Twine, Bitsy, Construct, and Godot. A notable difference from other game datasets is that all games in the

Undergraduate Games Corpus were appropriately licensed for use and redistribution in research.

Various datasets with gameplay data have been released, for analysing player strategies and training AI-systems that can play games autonomously. For example, Lin et al. [170] created a dataset of *StarCraft* replays that can be used for learning gameplay models. They also reviewed existing *StarCraft* datasets, and proposed a list of quality criteria for gameplay datasets for machine learning. Despite the differences in content, this research can be used as source of inspiration for creators of text corpora: what makes these datasets usable, how are they used by researchers after release, and what is ‘quality’ for datasets in games and AI?

Guzdial et al. [123] used a crowdsourcing platform (Mechanical Turk) to collect stories for their adapted version of *Scheherezade* [165], which can generate complete IF games from a dataset of stories.

In machine learning research, a commonly held view is that ‘more data is better’ [125], as opposed to having better algorithms or better data, and that this determines whether certain problems can be solved efficiently with machine learning. However, in their survey on procedural content generation with machine learning, Summerville et al. [258] argue that this does not hold for video games, as game data do not typically share common data structure or semantics. Consequently, data sources are only usable for PCGML within the same game or game series, which leads to a data scarcity. The authors suggest multiple ways to mitigate this problem, such as artificially enlarging small game content datasets through corrupting (masking) parts of the data, deriving additional datapoints from a single datapoint in the training data, e.g. multiple player paths through the same video game level, and gathering data from a secondary source, such as video.

8.2.2 Text generation for video games

Most research on generating textual game content uses either template-based techniques [53, 177], rewriting techniques based on grammars [223, 119, 126, 117] or graphs [155]. The use of newer NLP techniques, which build on machine learning architectures, must largely still be explored in the context of video games. A notable exception is the work of Walker et al. [280], who used statistical machine learning to create language models of the linguistic style of movie characters. The language models were used to generate dialogue with personality for *SpyFeet*, a prototype roleplaying game. The authors used film dialogues as the basis for the linguistic models.

A recent game that does use newer NLG techniques, i.e. large-scale neural language models, is text adventure game *AI Dungeon* [282]. Earlier versions of the game use OpenAI’s GPT-2 language model [209] to generate personalized text adventures. The latest version uses GPT-3, GPT-2’s even larger successor.

The underlying text generator was trained on a corpus from the website ChooseYourAdventure.com, a community around choose-your-own-adventure style story games.

A limitation of current research is that generator systems are often not based on or compared with material written by professional video game writers. For the purpose of academic research, most researchers create their own games, templates or grammars, or ask research participants to create game texts in crowd-source experiments [200]. An exception is when game developers publish in academic venues about their own text generators, such as Grinblat and Bucklew [119].

8.2.3 Text analysis for video games

We cannot separate *text generation* from *text analysis*, as text analysis can inform generative systems before and after generation. Before generation, a generator can use text analysis techniques to model the structure of its output by codifying patterns observed in examples. After generation, text analysis can be used to evaluate properties of the generated artifacts. Similarly, Summerville et al. [259] list ‘discovering design patterns’ as one of the purposes of the Video Game Level Corpus.

Heritage [128] applied corpus linguistics to video game data, to analyze how gender is represented in video game text. To ensure a representative selection of video game texts, the author selected random language samples from 10 AAA video games published between 2012 and 2016. “AAA” is a term used by the video games industry and the gaming community to denote video game titles that require a large production and marketing budget [181]. The text from these games was collected using a combination of extracting text dump files, using existing fan transcripts, and transcribing parts of the game by hand.

Landwehr, Diesner, and Carley [156] scraped a corpus of World of Warcraft quests from quest repository Allakhazam,¹ and used this to analyse the cultural and narrative elements embedded in quest text. Kybartas and Verbrugge [155] presented an approach for quest generation by using graph rewriting techniques. Their quest generator, called ReGEN, can generate new quests based on changes in game state, making player choices more meaningful. To validate their approach in a quantitative way, the authors also proposed a metric for the quality of a (game) narrative. They used this metric to measure the performance of their quest generator compared to the quests of *The Witcher* and *Elder Scrolls V: Skyrim*. The quest data for the two games was collected from game wikis. Schlünder and Klabunde [233] analysed greetings in NPC dialogue transcriptions of *Skyrim*, and proposed an algorithm for more context-sensitive greeting generation.

¹<https://wow.allakhazam.com/>

Allakhazam’s World of Warcraft quest database was discontinued in 2013.

8.2.4 Related text corpora

Video game text constitutes many different types of text. Depending on properties such as genre and gameplay, a game might consist of dialogue, narratives, quests, and flavor text. By flavor text, we mean game text that has a cosmetic purpose as opposed to a functional one. Text processing for the video games domain can profit from NLP research that studies the types of text that we also encounter in games, such as dialogue and stories. Text corpora for these types of text are much more common, e.g. the CMU movie summary corpus [16] and the ROCStories corpus [191] for stories, and the switchboard corpus [113] for dialogues. However, the usefulness of these corpora for the video games domain is limited, as results on these corpora might not be transferable to video games. Story corpora might contain stories that consist of a few sentences, which is not comparable to the interactive and complex narratives found in video games. Similarly, real-world dialogue corpora might not be similar enough to video game dialogue. Real-world dialogue transcripts are typically linear. This might differ considerably from NPC dialogue in games, which might consist of branching dialogue to incorporate player choices or changes to the in-game world state, such as the player's relation with different in-game factions. Game writers often work within complex constraints that are related to video game development practice. These constraints might not be reflected in corpora from other domains. Game writers might be constrained in terms of utterance length, for example because of voice acting requirements, limited player concentration, or interface design (e.g. dialog box sizes).

8.3 Quality of video game corpora

If games research is to benefit from the recent developments in AI and NLP, high-quality datasets of video game text are necessary. These corpora are useful for analysis, training and evaluation. Recent neural architectures, such as GPT-2 and BERT, can be fine-tuned on small, domain-specific datasets to increase their performance for specific domains or tasks. Video game text corpora can be used for fine-tuning these systems specifically to video game texts, which is likely to increase the effectivity of NLP techniques for the games domain. Additionally, text corpora that contain some kind of ground truth data or labels can be used for evaluating new techniques and systems.

Below, we propose multiple quality criteria for video game text corpora, similarly to the list provided by Lin et al. [170]. The goal of this is two-fold. Firstly, we have collected these corpora as foundation for future work in video game text research. We want to be able to apply corpora-based generation using GPT-2, which was demonstrated in the previous chapter. The technique for guiding GPT-2 places specific requirements on the dataset. Secondly, we noticed that quality criteria for research corpora are not always explicitly discussed in pro-

cedural content generation and NLG research. By explicitly naming our own design criteria for the datasets discussed in this chapter, we hope to encourage other researchers to do the same. Others can either use our list of requirements as a starting point, or design their own requirements. Making corpus requirements explicit has as advantage that it makes the underlying research goals and methodological choices also more explicit.

Richness Datasets should contain both game text and information about their in-game context.

Representativeness Strings in the dataset should be written by professional video game writers. Strings should preferably be sourced from popular or well-known (commercial) games that have a substantial user base.

Diversity Datasets should reflect the diversity of the video games domain.

Portability Datasets should be shared in a portable plain-text format that does not require special tools to read or modify.

We will elaborate on the four criteria below.

Richness

Researchers might be able to find ‘text dumps’ of popular games online, which consist of strings from the game without any context. However, because game text is typically dependent on the underlying game state, logic, and gameplay rules, game texts are inherently context-sensitive. If we try to analyse a game text in isolation, we cannot interpret it correctly. Consequently, game text corpora should provide *rich* information about the context of each text. For example, for dialogue lines, we need information about conversation participants. Which NPC is saying what, to whom, and why? What is their relation to the player character? Is a particular dialogue line part of a larger narrative (such as the main storyline) or a story of minor importance (a side quest, an NPC backstory, flavor text)? Are there specific conditions in which the text is shown, or explicitly hidden from the player? Is there a specific order in which text is presented, or is the player free to choose?

Another challenge is that corpora need labels or some other kind of ground truth before we can use them for supervised machine learning and evaluation. Although in most cases game texts do not have labels in the strict sense of the word, we can use properties from the in-game context as ground truth. We will discuss below how this applies to the datasets presented in this chapter.

The richness requirement is related to one of the desired properties of the Video Game Level Corpus. Summerville et al. [259] mention under ‘Future work’ that it is desirable to have an exhaustive dataset with a lossless representation of the game content. However, it is very hard to define what ‘lossless representation’ means for assets of a particular game. It could mean including solely

unmodified original game assets, at most in a more user-friendly format, or using a representation that closely resembles the game assets as the player sees them in-game. However, game assets are often not stand-alone data structures, and hard to use for researchers due to their highly specialized file structures. It is also complex to perfectly reconstruct a game asset's in-game context, because of the (often opaque) way a game engine assembles a collection of game assets during runtime. Besides these considerations, the definition of losslessness might differ per game as video game assets are highly heterogeneous

Representativeness

Some research uses corpora of video game text that are not *representative* of the video games domain. These corpora might contain text sourced from research games, text written by academics, or text crowdsourced from research participants. Ideally, video game corpora consist of text written by (professional) game writers, sourced from real-world video games. Here, we mean real-world games as opposed to prototype games or research games, which are also prevalent in research but are generally shared with and played by a limited audience. This requirement can be found in research that focuses on text analysis more than generation.

Heritage [128] explicitly named representativeness of the language sample as an important requirement. In the paper, the author spent considerable time explaining the design of the corpus used for analysis. To ensure a representative sample, the author filtered a top 100 list of most popular games for a particular timespan, and took a random sample of the games that met all conditions.

Interestingly, other authors explicitly choose corpora that are not representative for the commercial video games domain. Anderson and Smith [7] included games from undergraduate students in a game development program, which is a more diverse population than the current population of professional game developers of AAA games.

Diversity

Diverse corpora are needed to reflect the diversity in games. There are many different types of in-game texts: NPC dialogue, item descriptions, in-game lore, puzzles and riddles, narration, flavor text, names, quests, tutorials and text from graphical user interfaces. In practice, most researchers tend to work with data from the same games, or the same corpora, e.g. in shared tasks. For example, Morrowind and Skyrim, two games from The Elder Scrolls, are popular subjects in video games research, e.g. for dialogue analysis [233], linguistics of fantasy languages [94], quest generation and analysis [155] and social agents [66, 122]. It is unclear whether this is because of the series' popularity, or the availability of modding software. Similarly, Super Mario [268, 78, 260, 258], Minecraft [230, 10] and Starcraft [269, 274, 162] are popular games in procedural content

generation research.

When a game is popular as an object of research, this can start a virtuous cycle: resources for working with the game data are shared within a research community, and other researchers can easily build upon this work, using datasets or software as a starting point for new research. After a while, a particular dataset might grow into a shared benchmark. Although using the same datasets increases the ease with which we can compare different approaches, it also has drawbacks. If research is limited to only one type of game text, it does not do justice to the diversity of video games. Similarly, we need corpora that span the diversity in game genres, narrative genres and game developer backgrounds. Diversity is in the interest of the research field, as text processing methods might not transfer across game genres, narrative genres, storytelling methods, settings, writing styles and other aspects of game writing.

Anderson and Smith [7] use several dimensions of diversity. Apart from a variety in subject matter, they also included games from a population of student authors that is more diverse than the population of professional game authors. The games also vary in their level of design polish. Anderson and Smith [7] rightly note the paradoxicality of diverse corpora. On one hand, we need large and diverse datasets. On the other hand, the contents of the datasets need to be technically consistent and uniform, so that they can be used for training new AI techniques without the need of data cleaning, generalization or abstraction.

Portability

Finally, to ensure *portability*, corpora should be shared in a plain-text data format that is supported on a variety of platforms, such as CSV or JSON. In this we follow Summerville et al. [259], who published their dataset in machine-readable plaintext files, including a JSON legend for their annotation formats. The VGLC also includes the original game level data, so that other researchers can create their own annotations for the levels. However, this might not always be feasible for game asset corpora, both for copyright and size reasons.

8.4 Obtaining new video game corpora

In this section we discuss methods for obtaining data that can be used as a source for new video game corpora: extracting text from game files, and scraping text from fan-websites.

8.4.1 Extracting text from game files

The highest quality data can be obtained directly from game files, as these contain the actual text that players will see during the game. We discuss three different approaches for this: extracting data from files of open-source games,

using modding software provided by the publisher or developer, and using tools provided by online modding communities. Since we want to collect datasets that fulfill the representativeness property discussed above, we focus on real-world games.

Extracting text from *open-source games* can be an accessible approach to obtaining game texts from real-world games. It is in the interest of the open source community to make the inner working of the game, such as the game engine and the structure of game assets, as understandable and usable as possible. Consequently, files are often stored in open and human-readable formats, the structure of game files and the working of the game engine is often documented, and game files require no proprietary tools for inspection or modification. This is an advantage if we want to extract data from them for analysis.

Open-source games exist in a variety of types. They can be original games that were made available as open source from the start, such as *Endless Sky*, or open-source clones of closed-source games, such as *OpenRA*², an open-source clone of *Command & Conquer: Red Alert*. Some open-source clones are shipped with assets from the original game; others require the original game files. Besides open-source games, there are also efforts to create open-source *game engines*, such as *xoreos*,³ a project to open-source Bioware's Aurora game engine. An open-source game engine, and the accompanying tools, can help us extract game assets from commercial games.

However, most games are not open source. Games files of commercial games might be compressed, to save space and provide fast access for the game engine, or even encrypted, to prevent tampering and theft. In that case, we can use modding software to access the files. As modding tools are created with modification in mind, and not extraction, it varies whether it is possible to export (textual) game assets in bulk.

It is becoming more common for game publishers to release official modding software after the release of the game. Examples of games that come with their own modding toolkit are *Torchlight II* (GUTS), *Morrowind* (TES Construction Kit) and *Skyrim* (Creation Kit). The game's publisher or game development studio has an interest in the success of the official tools, as an active modding community can improve the life expectancy of a newly released game [161].

Official modding tools are often based on the developer's in-house tools. Consequently, they tend to be more robust than their community-provided counterparts discussed below. Their biggest advantage is that they often integrate well with the game engine and game files. Sometimes the publisher also provides extras that increase the usability of the tools, such as documentation and tutorials.

If the publisher has not released any tooling for modifying a game, or official tooling is found to be too restrictive, the player community often starts making their own tooling. Community-provided tools are shared online via modding

²OpenRA, <https://www.openra.net/>

³<https://xoreos.org/>

community websites (such as NexusMods), gaming forums, and gaming platforms (such as Steam Workshop). Tools vary in sophistication from simple scripts to professionally developed software with a GUI and documentation.

Relying on community-built tools has a few caveats. Not every game has an active modding community, and there is no guarantee that a community-provided tool will actually function correctly. Tools may be untested, undocumented, or incompatible with newer computer systems. Finally, community-provided modding tools might require a high level of technical proficiency of the user.

8.4.2 Extracting game text from fan websites

Fan culture can give rise to extensive fan-made websites and wikis, where players collect information about the game, discuss strategies and share fanart. Often these fan-made websites are a great resource for texts (and other media) from the game. Kybartas and Verbrugge [155] used the fan wikis of *The Witcher* and *Skyrim* to obtain information about game quests. In a previous work [26], we used in-game lore books and NPC dialogue sourced from The Elder Scrolls fan websites for sentiment analysis research.

The main advantage of collecting data from fan websites is that the text is already available in plain text, as opposed to text in game files, which might be compressed, encrypted or stored in a proprietary format. A possible drawback is that data from fan websites generally needs considerable data cleaning before it is of comparable quality to data extracted from game files. Since fan wikis are often crowd-sourced, we cannot be sure of the accuracy and completeness of the text we find there. Information might be spread over various pages, or structured in a heterogeneous format. Similar to other crowd-sourced online resources such as Wikipedia, we might find text with errors ranging from spelling mistakes to false information. Another drawback of extracting game text from fan websites is that the texts might be presented without information about in-game context, which is contrary to our richness requirement.

8.5 Datasets

We used the sources mentioned in the previous section to create three datasets with game text. The texts were sourced from popular commercial games: Torchlight II [222], Star Wars: Knights of the Old Republic [36] and games from The Elder Scrolls video game series [31]. We used a different method for each corpus: for Torchlight, we extracted the data using the publisher's official modding software GUTS; for Knights of the Old Republic we used community-built tools; and the corpus with The Elder Scrolls text was scraped from fan-websites. The three datasets contain a broad range of game texts: linear NPC dialogue, branching NPC dialogue, quest objectives, GUI text, and flavor text. We briefly discuss the

collection method and contents for each dataset. Our methods for extracting text data from game assets do not generalise to other games, which is why we have not included a detailed technical description of our data collection methods in this chapter. However, we have provided detailed descriptions of our extraction methods with the released datasets⁴ on Github.

8.5.1 Dataset: Torchlight II quests

An example of a game that comes with modding software provided by the publisher is Torchlight II. Torchlight II is an action role-playing game that takes place in a fantasy world. The game consists of a main story that revolves around the destructive and corrupted Alchemist, and a collection of randomly generated dungeons that the player can explore as side-quests. The publisher, Runic Games, has published their in-house development kit “GUTS”, together with a set of tutorials to teach players how they can change parts of the game and write their own extensions.

Torchlight’s quests follow a similar structure as quests in World of Warcraft and other MMORPGs. Quests in Torchlight often start with a dialogue with a special quest-giver NPC. These NPCs can be identified by a yellow exclamation mark. There is dialogue for starting and finishing a quest. When the player returns to the quest-giver before finishing the quest, the quest-giver sometimes speaks interim dialogue, e.g. “Are you finished running the errand yet?” or “Hurry! What are you waiting for?” Quest dialogue might also contain hints about the quest objective. Figure 8.1 shows the first quest the player encounters in-game, which kicks off the main story line. The objective of this quest is to travel to the Estherian Enclave and warn the Estherians, one of the peoples that inhabit Torchlight’s game world, against the Alchemist, the main antagonist of the game.

⁴<https://github.com/hmi-utwente/video-game-text-corpora>



Figure 8.1: Screenshot from Torchlight II [222] that shows the opening quest dialogue for the quest titled ‘Warn the Estherians’.

Dataset creation

We used GUTS to extract all quests and related dialogue from the game files. When Torchlight II is installed on a pc, its game assets are stored as XML-like UTF-16-encoded plaintext files, which are compressed and stored in PAK archives. We used GUTS to unpack Torchlight’s game files from its main PAK archive. We then created a Python script to parse the XML files, extract the game text, and turn this into a ready-to-use dataset with quest texts and associated NPC dialogue. For accessibility reasons, we have created two datasets. The first dataset is a straight-forward two-dimensional table that lists each line of in-game text and relevant meta-data, such as information about the speaker. Figure 8.2 shows the structure of this table, and describes the contents of each column. Figure 8.3 shows an example for all columns. The second dataset is a nested JSON file, that closely resembles the structure of the original game files. Apart from dialogue lines, this dataset also contains the meta-data of each quest, which determines when and how a player encounters this quest in-game. Both file-formats are highly portable, as they consist of plaintext data that is compatible with various tools and libraries.

In order to create the dataset, we combined data from two types of game assets: quest files and unit files. *Quest files* describe events, story components and dialogue. They are used to control the flow of the game narrative, and make

Column name	Description
Text	The string literal of a piece of text.
Speaker	If the text type is dialogue, this column lists the human-readable name of the speaker (unit) of this piece of text. Some lines do not have a speaker but just appear in the game, as if spoken by an invisible narrator or thought by the player character. GUI texts never have a speaker. The dataset contains 83 unique speaker names.
Dialogue type	One of the text types from Figure 8.4.
Quest displayname	The human-readable name of the quest, as it appears in-game. Some quests are not “real” quests for the player to solve but instead govern in-game animations, checkpoints and passive dialogue. These quests are not shown to the player and have an empty quest name.
Quest name	Internal identifier of this quest as used by the game engine. Every quest has an internal identifier like this, and each unique identifier corresponds to one unique game file.
Questfile	The game file (.DAT files) from which the quest data for this row was extracted.
Speaker unit	Internal identifier of the speaker of this piece of text as used by the game engine. Every speaker has an internal identifier like this, and each unique identifier corresponds to one unique game file.
Unitfile	The game file (.DAT files) from which the speaker data for this row was extracted.
Raw text	Un-edited, raw string for this line of text as it appears in the game files. These strings contain formatting information, such as <code> cFFA1D0FCTemple Steppes u</code> , which tells the game engine to print this piece of text in bold and in colour.

Figure 8.2: Dataset structure of the two-dimensional Torchlight II dataset in CSV-format. Not every datapoint (row) has a value in every field.

up the main storyline, together with a set of side quests that revolve around procedurally-generated dungeons. *Unit files* describe interactive in-game objects, such as NPCs, items and doors. We used the unit files to translate abstract references to NPCs and monsters (alphanumeric identifiers) in quests to human-readable names.

Column name	Example
Text	Go on, then. Glory awaits!
Speaker	Fazeer Shah
Dialogue type	return
Quest displayname	The Djinni's Next Task
Quest name	a2-djinni_2st_task
Questfile	QUESTS/TL2MAINQUEST/ACT2/ A2-DJINNI_2ST_TASK.DAT
Speaker unit	A2-DJINNI
Unitfile	MEDIA/UNITS/MONSTERS/QUESTUNITS ESTHERIANS/A2-DJINNI.DAT
Raw text	Go on, then. Glory awaits!

Figure 8.3: Example datapoint from the two-dimensional Torchlight II dataset in CSV-format.

Dataset contents

The dataset of Torchlight II quests consists of 184 quests, of which 131 quests contain text. Quests without text are used for controlling in-game objects, such as doors and checkpoints.

Torchlight's quest files can contain many different types of texts, such as NPC dialogue, flavor text, back story and GUI text. Most of the texts are dialogue lines. Whether a particular dialogue line is shown in-game depends on the player's progress for that respective quest. Quests might also contain flavor text. For an overview of the different text types, see Figure 8.4. Figure 8.5 shows three lines of dialogue from one of the side-quests in the game. The amount of dialogue contained in each quest varies. Simple side-quests contain only a few lines of dialogue for one NPC, i.e. the quest-giver NPC that acts as the start and completion point of a quest. Larger quests may contain dialogue lines for multiple NPCs.

The Torchlight II dataset consists of about 1000 pieces of text, of which approximately 70% is NPC dialogue. 27 datapoints contain a long-form story synopsis that summarizes part of the main quest. The remaining datapoints are GUI text, which describe quest objectives in one or two lines. Figure 8.6 shows the exact number of texts for each category.

Text type	Text field	Description
Dialogue	intro	Dialogue text of the NPC that introduces the quest to the player.
Dialogue	return	Dialogue that the NPC speaks when the player returns to the quest-giver before completing the quest.
Dialogue	complete	Dialogue for when the player returns to the quest-giver NPC after successfully completing the quest objective. After this text, the player receives a reward for completion of the quest, or is shown a new intro text to start a follow-up quest.
Dialogue	passive	Stand-alone dialogue lines that act as flavor text.
GUI	details	The goal or objectives of the quest, as shown upon quest acceptance.
GUI	huddetails	A list of quest objectives. This list is shown in the game UI when the quest is active.
GUI	more details	Extra backstory for quests from the main quest-line.

Figure 8.4: Text types in Torchlight II quest data, and their purpose in the game.

Text field	Example text
intro	Hello! I thought I heard a human moving around out there. Listen, my name's Medrus. I got ambushed by some Sturmbeorn, and managed to get clear ... but I got pretty badly injured in the process. I can treat it, but I need some Merryweather Leaves. They grow around here, but I'm too weak to look for them. Think you can find some for me, bring 'em back here? You'll be rewarded, I promise.
return	Any luck finding the Merryweather Leaves? I'm not sure how much longer I can hold on ...
complete	You found some! Oh, thank the gods. A few moments' work, and ... yes, there it is: a healing poultice. Now it will just take a little rest, and I'll be good as new. As it turns out, you brought back more leaves than I needed. So, here: a Healing Poultice for you, as a reward. Should you be badly injured, it'll set you right in no time!"
details	Collect Merryweather Leaves from a Merryweather Bush, and return to Medrus, in the Temple Steppes.
huddetails	- Gather Merryweather Leaves for Medrus in the Temple Steppes
huddetails_complete	- Return to Medrus in the Temple Steppes

Figure 8.5: Text from the Torchlight II dataset for the sidequest “The Merryweather Poultice”. The first three texts are NPC dialogue, and the last three texts are GUI text. This quest does not contain the field ‘passive’, as the only character that has dialogue related to this quest is the quest-giver NPC. The quest also misses the field ‘moredetails’, which only occurs in quests that are part of the main questline.

Text type	Text field	Amount
Dialogue	complete	83
Dialogue	passive	449
Dialogue	return	83
Dialogue	intro	75
GUI text	details, details_complete	136
GUI text	huddetails, huddetails_complete	155
Story synopsis	moredetails	27
		1008

Figure 8.6: Text types and text fields in the Torchlight II dataset, and the number of texts of that type.

8.5.2 Dataset: Knights of the Old Republic dialogue

Star Wars: Knights of the Old Republic (KOTOR) is a turn-based action RPG by BioWare [36]. The game, which is set in the Star Wars universe, is known for its high-quality writing, complex narrative and branching dialogues. During conversations with NPCs, players can choose from a set of pre-written dialogue options. Depending on their choices, different things happen in the game. Player choices affect the player character's abilities, NPC relations and story endings.

KOTOR's dialogues are highly subjective and affective. The game story deals with the battle of the good Jedi against the evil Sith, and conversations in the game reflect this theme: dialogues do not only revolve around collecting information, but also around feelings, relationships, and complex moral choices. Because of this, the dataset contains many different dialogue acts: characters joke, fight, grieve, lie, bargain, persuade and fall in love with each other. For an in-depth discussion of KOTOR's narrative, we refer the reader to [284, p. 59–69].

Figure 8.7 shows an example of Knights of the Old Republic's branching dialogue. This dialogue is encountered when the player is trying to infiltrate a Sith military base on the planet Taris. Upon entering the base, they are stopped and questioned by the receptionist. The player can choose from multiple dialogue options to try to persuade or threaten the receptionist, see Figure 8.8. The dialogue is branching, but there are only two possible outcomes – the receptionist either raises the alarm, or flees – and the outcomes don't influence the larger narrative. However, if the player manages to persuade the receptionist, this has a strategic advantage. If the alarm is raised, the player immediately has to fight a group of Sith soldiers. If they manage to invade the base stealthily, the player can take the soldiers in the Sith base by surprise.

	Char.	Text	Next line
1	T	Hey - you can't come in here! This is a restricted area! You better tell me what you're doing or I'm going to hit the alarm!	2A, 2B, 2C, 2D
2A	P	[Persuade] I'm here for a meeting. Don't worry - I know where I have to go.	3
2B	P	[Persuade] I work here - look, I've got an access card.	3
3	T	[Failure] Nice try. What - you think that just because I'm pretty I'm also stupid?	4
4	T	You better tell me why you're here or I'll hit this alarm and you'll have about three seconds before this office is swarming with Sith soldiers!	2A, 2B, 2C, 2D
2C	P	[Persuade] Look, here's 50 credits - just don't hit that alarm.	6A, 6B
6A	T	[Success] 50 credits? You've got a deal! The Sith have made my life a living hell ever since they took over this base.	7
7	T	You know, it's about time someone stood up to these Sith! Just do me a favor and wait until I'm out of here before you start blasting the place up.	Dialogue ends: receptionist flees.
6B	T	[Failure] You must think I'm pretty stupid! If I take your bribe the Sith will have me executed!	8
8	T	I don't know what you're up to, but I have the feeling it might get me in a lot of trouble. I hate to do this, but you'll have to explain yourself to the guards.	9
9	P	Don't hit the alarm. I don't want to have to hurt you.	10
2D	P	Touch that alarm and you're dead!	10
10	T	Don't shoot! I just work here - I never wanted any part of this! I would have quit when the Sith took over, but they wouldn't let me!	11A, 11B
11A	P	Okay - get out of here and I'll let you live!	7
11B	P	You're in the wrong place at the wrong time. It's bad luck, but I still have to kill you!	12
12	T	You shouldn't threaten a girl who's got an alarm right at her desk!	Dialogue ends: alarm is raised

Figure 8.7: Branching dialogue from Knights of the Old Republic between the player character (P) and the Twi'lek receptionist (T) of a Sith military base. The dialogue has two possible outcomes: the receptionist either raises the alarm (line 12) or flees (line 6). Lines 2 and 11 are choice moments for the player. Line 6 features Persuade, a skill that the player can use. Depending on the outcome of that skill (a diceroll), the next line is either 6A [Success] or 6B [Failure].

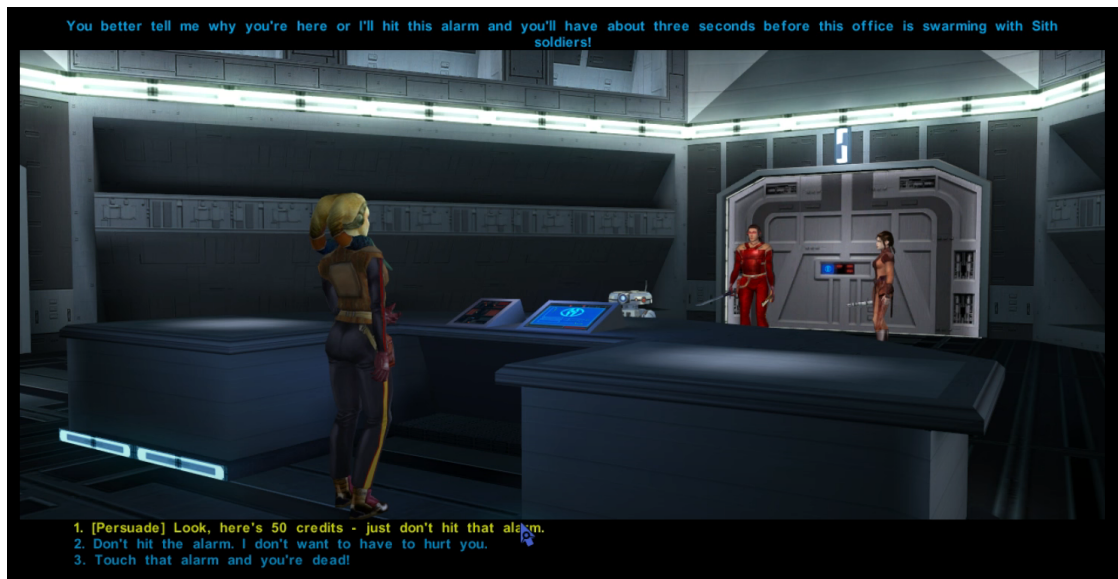


Figure 8.8: The player character (in red) infiltrates a Sith military base and is stopped by the Twi'lek receptionist. The player can choose how they will react to her challenge. Screenshot from Star Wars: Knights of the Old Republic [36].

Dataset creation

Text from KOTOR is not easily accessible outside the game, since the game assets are stored in compressed archive files in a proprietary format. We extracted all game assets with text using `xoreos-tools`, a collection of open-source modding tools⁵ provided by the Xoreos project. Xoreos⁶ is an ongoing open-source project to reimplement publisher BioWare's Aurora game engine. We developed a collection of scripts in Python and bash to parse and re-assemble the game files from the extracted files. The scripts reconstruct all dialogue trees from the game to create a dialogue corpus in CSV-format. For a technical description of the extraction and re-assembly of the dialogues, we refer the reader to our dataset on Github.⁷

Dataset contents

To construct the Knights of the Old Republic dataset, we used a similar method as for the gamefiles of Torchlight II. KOTOR's dialogue texts are stored in special dialogue files. Each text represents one conversation turn in a dialogue. KOTOR contains branching dialogue with a graph-like structure. Intuitively, branching dialogue has a tree structure, but KOTOR's dialogues contain cycles. These allow the player to go through the same dialogue multiple times, for example to ask a NPC to repeat information.

⁵<https://github.com/xoreos/xoreos-tools>

⁶xoreos: an open-source reimplementation of the BioWare Aurora Engine. <https://xoreos.org/>

⁷<https://github.com/hmi-utwente/video-game-text-corpora/>

Knights of the Old Republic's dialogue files describe branching dialogues as a linked list. Each line of text refers to the lines that come afterward, together with a set of logical constraints for each transition. If the constraints are fulfilled in the game's current world state, the next line of dialogue becomes active.

However, even if we have access to dialogue files, they cannot be read as is. Dialogue files only contain *identifiers*, which refer to characters, maps, string literals, or game files, instead of the actual data. In order to obtain the dialogue lines as a player sees them in-game, we had to translate the identifiers back to the actual data. We combined the data from dialogue files (.dlg files) with the data from Knights of the Old Republic's string lookup-file (.tlk files). We also wanted to know who is speaking which line of dialogue. To get human-readable character names, we also combined the dialogue lines with information from character files (.utc files). Some dialogue lines also contain animation codes for one or more dialogue participants. Animations govern facial expressions (looking sad, looking angry, laughing) and character actions (taunt, walk, torture).

The reassembled dataset contains over 25,000 lines of dialogue of 556 uniquely-named dialogue participants (listeners and speakers). The dataset contains 3305 dialogue tree root nodes, i.e. dialogue lines where the player or an NPC starts a conversation. Besides conversations between multiple humanoid characters, the dataset also includes interactions between the player and droids (robots), security systems, doors, and other interactive game objects, as the game represents these interactions as dialogue lines. For example, if the player interacts with a droid, the droid might “reply” with “This droid is damaged and inactive” or “Whirr. Click. Beep. Boop.” Since it is text data from the game's .dlg files, we decided to keep these object interactions in the dataset.

Each datapoint in the dataset describes one turn in a conversation between two or more characters. Besides the dialogue text, the dataset contains the name of the speaker, the name of the listener (optional), the dialogue tree (references to other dialogue lines), which character animations should be played during the dialogue line, and game developer comments. For an overview of the information included in each datapoint, see Figure 8.9. Figure 8.10 shows basic statistics for this dataset.

Key	Description	Example value
Id	28209	Identifier of this dialogue act in the dataset
Speaker	Judge Shelkar	The character or object that communicates the line
Listener	Player	The character that listens to the line
Text	For your crimes against Manaan and the Selkath you are banned forever from this world, on pain of death!	String literal
Animation	'Judge Shelkar': 'Talk_Forceful'	3D animation that should be played during the delivery of the line
Comment	if the player is exiled	Game development notes
Previous	[28208, 28252, 28314, 28332]	Identifiers of previous dialogue lines
Next	[28210, 28213, 28215, 28218]	Identifiers of possible follow-up dialogue lines
Source DLG	man26_pcexile	The .dlg file in which this conversation turn can be found.

Figure 8.9: Datapoint (conversation turn) from the KOTOR dataset. Dialogues consist of multiple turns. Dialogues are stored as double linked list and can be reconstructed by walking the linked list, i.e. following the 'previous' and 'next' references.

Conversation turns	25572
Uniquely named characters	556
Dialogue root nodes	3305
Dialogue end nodes	4157
Dialogue lines with animation data	2207
Total .dlg source files	984
Total .utc source files	830

Figure 8.10: Statistics about the Knights of the Old Republic dataset.

8.5.3 Dataset: The Elder Scrolls documents

The Elder Scrolls (TES) is a series of role-playing video games by Bethesda Softworks, consisting of single-player role-playing games *Arena* (1994), *Daggerfall* (1996), *Morrowind* (2002), *Oblivion* (2006), *Skyrim* (2011) and multiplayer on-line game *The Elder Scrolls Online* (2014). Games in the series are open-world games, which means the player can explore the game world at their own pace and choose which objectives they want to focus on. The games take place in a fantasy world called Tamriel, which has a rich history that is communicated in various ways throughout the game: through NPC dialogues, quests objectives, cut scenes, and in-game documents, such as books and notes.

This last category of books and other written texts plays an important role in the game. These documents are collectible objects that the player can find as they travel through the world. Books can be opened and read by the player, as shown in Figure 8.11. The books contain flavor text, i.e. text that is not a critical part of the game's main narrative, but gives the player background information about the world they are exploring. Their length varies from a few words to a few hundred words, and some books are part of a series of multiple volumes.

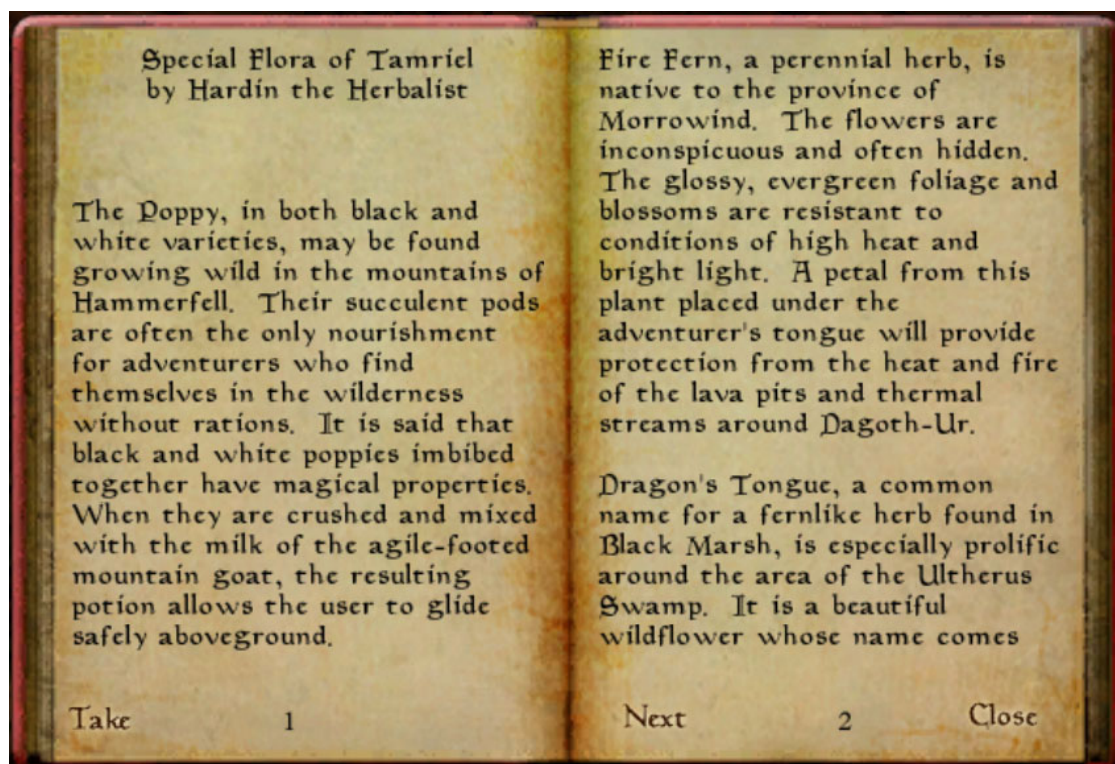


Figure 8.11: Screenshot from *Morrowind* (2002). This is what a player sees when reading the in-game book 'Special flora of Tamriel', a handbook on the medicinal uses of various plants and fungi that occur in Tamriel.

The importance of books can be felt throughout the game world. Cities in the game have bookshops, libraries and record halls. Figure 8.12 shows the

library of Vivec, one of the major cities in the game *Morrowind*. Some books are deemed dangerous by local rulers, for example because they criticize the state religion, and consequently forbidden. In the questline for the Morag Tong, *Morrowind*'s guild of assassins, the player can obtain 'writs of execution': licenses that authorize them to perform sanctioned executions, similar to the letters of marque used by privateers in the 17th century. In *Skyrim*, multiple quests involve obtaining rare books and manuscripts for the Mage's Guild librarian.



Figure 8.12: The library of Vivec, from *Morrowind* (2002)

Dataset creation and contents

*The Imperial Library*⁸ is a fan-website for The Elder Scrolls, which collects in-game documents from the series. We scraped the text of over 4800 in-game books, letters and notes from the website. The final dataset consists of 4890 documents (at least 4470 unique titles) from six games. Together, they form a corpus of over 160,000 sentences and 2,000,000 tokens. The Imperial Library website also lists metadata for the in-game documents, such as title, fictional author information, and a short summary of each document. We included this metadata in the dataset as well. For an overview of the structure of the data and an example, see Figure 8.13.

⁸<https://www.imperial-library.info/books/all/by-category>

Key	Description	Example value
Game	The game from which this document is sourced.	Morrowind
Url	The url of the original webpage	https://www.imperial-library.info/content/dying-mans-last-words
Author	The (fictional) author of the document	Indie
Title	The title of the document	A Dying Man's Last Words
Summary	Summary of the document, written by fan-site contributors	The last words of a world-renowned archaeologist.
Text	Text of the document	It's been many days since the collapse. I have had many good and exciting adventures. I fear this is the last. I am still unsure what happened. Was it a trap that caused the collapse? I didn't hear the click of any device. Perhaps it was simply a freak accident, and I was simply in the wrong place at the right time. (...)

Figure 8.13: Datapoint from the The Elder Scrolls dataset for *A Dying Man's Last Words*, a book in diary form. The book contains references to the Indiana Jones movies. It can be found in a burial tomb, next to a skeleton grasping for a ruby.

8.6 Discussion

8.6.1 Applications

Video game text corpora can be used for a variety of purposes. Summerville et al. [259] name three purposes for their Video Game Level Corpus: corpora-based procedural generation, design analysis and style transfer. These applications also apply to the corpora presented in the previous section. Carrillo Masso [55] and Heritage [128] used video game text corpora to analyze the representation of gender in commercial video games. Although our three corpora are highly specialized because they only focus on one game (series) each, they can be used to explore the same type of research questions. All three datasets could be used to investigate the morphological and onomastic properties of toponyms, as researched by Fekete and Porkoláb [94], as they contain names of in-game locations. The names would still need to be extracted, but we estimate that recent named entity recognition methods could be used to automatically detect

place-names.

Because the datasets differ in included text types and size, they lend themselves to different types of research. We will discuss a few ideas for applications for the specific datasets below.

The Torchlight II quest dataset contains text type annotations, which can be used to filter specific types of text by in-game purpose. For example, since we can distinguish between quests from the main quest line and side quests, we can use this dataset to study the differences between these quest types. If we are researching flavor text, we can look at quest objects that contain ‘passive dialogue’. Another example is quest objectives. Quest objectives are very short summaries of the task the player should perform to complete the quest. This task is also described in the NPC quest giver opening dialogue, albeit with more words and more flavor than the objective. This means that we can use the list of quest objectives as ground truth for summaries of quest introduction dialogue. This data combination could be used to evaluate summarization techniques from the NLP field, to see how well these techniques perform in a video games context.

In contrast to the Torchlight II dataset, the Knights of the Old Republic dataset consists of only one type of text: dialogue. We can use it to analyse and generate both linear and branching dialogue. The main strength of this dataset is its size, in terms of total lines of dialogue, the different speakers, and the breadth of the covered topics and sentiments. Because the lines were directly extracted from a game that is known for its high-quality writing, the dataset can be considered representative of commercial video game writing. As a result, this dataset can be used for style analysis, and training dialogue generation systems where the envisioned application domain is video games. The dataset has been annotated with speaker and listener information, and some conversations involve more than two characters, so the dataset can be used for multi-party dialogue generation. Additionally, the dialogues can be used for analysing character relationships and sentiment. It can also be used to study the writing of a particular genre or setting, in this case science fiction and the Star Wars universe. Because of the high number of domain-specific fantasy words, the dataset can also be used to evaluate NLP techniques for domain-specific languages or low-resource languages.

The Star Wars dataset contains 2,207 dialogue lines that are annotated with animation data, that indicate which character animations should be played during the delivery of the dialogue line. Although less than 10% of the dataset is annotated this way, the animation annotations are particularly rich because they convey actions and mental states, such as intent or emotions, of game characters. In other words, we could interpret some of these annotations as affective labels. Figure 8.14 contains an example of a dialogue where the lines have animation annotations. The dialogue lines with affective labels can be used for sentiment analysis and affective text generation.

Id	Speaker	Text	Animation
28181	Mandalorian	You've been holding out on us again. Since you haven't given us enough money, I guess we're going to have to take it out of you piece by piece!	Mandalorian: Taunt Farmer: Horror Duros Warrior: Talk Laughing
28182	Farmer	No! Please! Take my wife and children instead! Anything!	Farmer: Talk Pleading Mandalorian: Ready weapon
<i>The Mandalorian shoots the farmer</i>			
28183	Mandalorian	Ha-ha! Mmm... Wife and children. Sounds like a good idea...	Mandalorian: Victory Duros Warrior: Talk Laughing Duros Warrior : Talk Laughing Duros Warrior: Talk Laughing

Figure 8.14: Reconstructed dialogue from the Knights of the Old Republic dataset. This dialogue is taken from a cut scene in which bandits, a Mandalorian and multiple Duros aliens, harrass a farmer on the planet Dantooine. All three lines contain animation data. Every turn has only one possible successor, so this dialogue is linear.

Because this dataset is so large, we can use it as a basis for smaller, task-specific datasets. For example, we could filter the dataset for questions and answers by searching for dialogue trees with question marks. We could also filter dialogues with personal histories (search for sentences with high sentiment and subjectivity scores), jokes (lines with ‘laughing’ animations), or requests for help or mercy (lines with a ‘talk pleading’ animation).

The Elder Scrolls dataset with in-game books consists of flavor text that describes the game world that the player’s character inhabits. It can be used to study the structure and contents of game lore and game settings. This can be used for analysis, like in the research of Landwehr, Diesner, and Carley [156], or for generating new game lore [119, 126].

The in-game books are interesting to analyse because of the way they explicitly inform the player about the game world, which differs from dialogue. The dialogues from Torchlight II and KOTOR mostly implicitly describe their setting. In KOTOR, players should derive the meaning of words like *rancor* (a monster), *droid* (robot) and *vibroword* (a melee weapon) from their context, as these terms are not explained in the game. This differs from the books in The Elder Scrolls,

which explicitly describe the game’s high-fantasy setting through fictional reference works such as dictionaries, maps, manuals, cookbooks and histories.

Because of its size, the dataset can also be useful in cases where a relatively large corpus is needed for machine learning. For example, in an earlier work [26], we used a preliminary version of this dataset for sentiment analysis on game texts. We created a language model from the lore text to learn the implicit relations between English words and non-English words from the games’ setting, which was then used to adapt a sentiment analysis lexicon for English to the domain of *The Elder Scrolls*.

8.6.2 Limitations

The *Star Wars* dataset was the most complex to create, as we needed to combine several game assets to access the dialogue lines. Understanding how these game asset files are interwoven required some reverse engineering, which is error-prone. Extracting the dialogue data of the game was relatively straightforward using the utilities of the *xoreos* project, but reconstructing the dialogue lines as the player sees them in-game proved challenging. Consequentially, the dataset might not be complete. *KOTOR*’s master file with strings contains all 49,265 strings from the game, of which 48,000 strings are non-empty. However, when we reconstructed the dialogue, we ended up with 25,575 lines. This means that more than 20,000 strings from the game are not part of our dataset. In-depth inspection showed that these other strings include (among others) error messages, menu and button texts, item flavour text, quest objectives, attack names and descriptions, character properties and abilities, help text, and text from tutorials and demos. To obtain a complete dataset, we could easily make a string dump of the master file, but this does not provide us with the context of those strings in the game.

Some of the books in *The Elder Scrolls* contain images, such as treasure maps, and the picture book *’ABCs for Barbarians’*. We did not scrape the related images, and did not include these in the dataset.

8.6.3 Future work

For the creation of the *Star Wars* dataset, we only used the character files and dialogue files. By reconstructing game context from the other game files, such as the maps, mission files, item files and audio files, other datasets could be created, both textual datasets annotated with additional in-game context and datasets for other modalities. To give a concrete example, many dialogue lines are governed by game scripts (.nss files) with conditional logic. It would be useful to know the exact logical preconditions for all dialogue lines. These conditions are of the form “The player character has the Scout class”, “The player has at least X dark side points”, “The player chose to free the NPC earlier in the game” or “The player successfully finished mission Y”. We could find references to these scripts

in the .dlg files, but we did not manage to automatically extract the contents of all these scripts and link them to the dialogue lines. It would also be useful to expand the dialogue lines with additional information about the speaker, such as their gender, (alien) race, faction, and moral alignment. This information could be obtained from the character files.

The Knights of the Old Republic game files also contain many examples of flavour text, especially for quests and items. Extracting the quest text for Star Wars quests would bring the dataset closer to the Torchlight dataset, which would allow for cross-training a language model on both datasets, and experiments related to transfer learning. A language model could be trained on data from one game, and then prompts could be generated with in-game context from the other dataset. Flavor text for in-game items could be used for corpus-based procedural generation of item descriptions. The outputs of such a generator could be compared to combinatorial procedural generation with grammars, a popular technique for procedural item generation in AAA games, e.g. Diablo [40].

In theory, the method we used to extract the dialogue from KOTOR could be applied to all video games that use Bioware's Aurora game engine, using the open source toolkit provided by xoreos. The xoreos website has a list of all supported games, which includes popular narrative-heavy games such as The Witcher [59] and Dragon Age [34]. We estimate that small changes to our extraction method will be necessary because of the minor changes to game engines between game development projects of the same studio. Still, this could be a viable alternative to crowdsourcing this information from fan-websites, which is the standard method for text from these games, e.g. [155, 128].

8.7 Conclusion

In this chapter, we discussed our requirements for video game text corpora: richness, representativeness, diversity and portability, and contrasted these to the requirements we find in other research. We described the places where source data for building new corpora can be found, namely in game files and on fan websites. Video game text is not always easily accessible, even if we have direct access to the original game files. Three categories of games for which this problem is solved or mitigated are open source games, games with official modding software, and games with an active modding community. When looking to create new corpora, we advise researchers to look in these places first. Finally, we presented three ready-to-use datasets with text from a number of popular role-playing games. These datasets can be used for various applications, such as NPC personality modeling, sentiment analysis, dialogue generation, lore generation, and quest generation. The three datasets are available online, together with the code used for extracting the data: <https://github.com/hmi-utwente/video-game-text-corpora>. Anyone that has access to the game files (i.e. owns the game) of the respective games can use the code to re-create the datasets.

Part III

Conclusion

This thesis investigated text generation for video games, especially flavor text for role-playing games. In this chapter, we will start with briefly summarizing the thesis in Section 9.1. In Section 9.2 we will revisit the research questions from the introduction and reflect on our findings. We will finish with discussing some ideas for future work in Section 9.3.

9.1 Summary

As we stated in the introduction, this thesis contains work on procedural generation of textual game assets. Chapter 2 and 3 discussed the relevant literature on procedural content generation and natural language generation, respectively. Video game text is context-sensitive, and thus we need to incorporate video game context in generated assets. For this, we introduced the term game-level coherence, i.e. textual coherence at the video game level, in addition to local and global coherence. To date, there has been little research in this direction, so it is challenging to find work on which we can build. However, video game text tends to be fictional, which is why video game text generation is close to fiction generation.

In Chapter 4, we analysed story generation projects from NaNoGenMo 2018 to find text generation methods for creating coherent novel-length texts. We identified four approaches: hardcoding a narrative, hardcoding narrative elements, emergent narrative, and methods that evoke the feeling of a coherent text. A common critique from the procedural generation community on natural language generation tools is that they are too complex to understand or use for non-experts. To address this problem, we specifically looked for methods that are computationally light-weight and easy to understand. Especially the ‘evoking coherence’ approach, which frames generated text in such a way that the reader will perceive it as plausible, seems particularly useful given this goal. If we can

integrate the tricks mentioned in this chapter in procedural content generation, we might be able to influence the reader's perception of procedurally generated game assets, even if they are created by unsophisticated text generation techniques.

We tested this in Chapter 5 with the creation of Churnalist, a flavor text generator that creates headlines from a short piece of input text. The 'evoking coherence' trick that we used for Churnalist is word reuse. The hypothesis was that if Churnalist reuses words from the input in its output, readers will perceive the resulting headlines as coherent with the input text. Churnalist generates 'new' headlines using text substitution, a technique that is straight-forward to implement, fully transparent, and easy to understand for non-experts. Our evaluation results confirmed that indeed readers perceived those headlines that reused phrases from the input as more coherent than headlines without phrases from the input. Churnalist's language quality was rated significantly worse than those of human written headlines, which might be a limiting factor in whether the outputs were rated plausibly part of a video game in the functional evaluation.

We tried to mitigate the language quality problem in Chapter 6, where we modified Churnalist to include a large-scale neural language model instead of rule-based substitution. This language model was based on recent neural model GPT-2, which can create coherent texts using its latent knowledge of the English language. Our model was fine-tuned on a dataset of headlines to learn the language that is specific to newspaper headlines. We experimented with guiding the language model towards generating headlines with words that occur in Churnalist's input text, by conditioning it on an implicit generation task during training. An error analysis showed that the language quality of Churnalist's outputs improved dramatically when generated with the GPT-2 model. However, the neural language model had difficulty dealing with non-English words in the prompts. Non-English words are common in game narratives with a fictional setting, e.g. fantasy or science fiction games. We estimated that fine-tuning or cross-training GPT-2 on video game text might mitigate this problem in the model, as fine-tuning on game text can teach GPT-2 the latent semantics of these non-English words, making the model more robust.

In Chapter 7 we tested this hypothesis by finetuning GPT-2 on video game text. We searched online for potential ways to create a new corpus, which should be large enough to train GPT-2 and contain contextual meta-data. The meta-data could then be used to teach GPT-2 an implicit NLG task, and guide the generator to an output with desired properties, similar to the headline generation task from Chapter 6. We eventually decided on a dataset with World of Warcraft quest data scraped from a fan-website. This new dataset was used to create a neural model that can be used to generate World of Warcraft quests, using a title and objective as prompt. We evaluated a collection of quests created by the model on language quality, coherence and creativity, and found that generated quests were rated slightly worse than manually authored quests. However, once

the language model was trained, it could create large volumes of text without additional effort, and the evaluation showed that some generated quests were rated higher than quests created by human writers. This suggests that a generate-and-curate strategy could be a worthwhile approach for supporting game writers in their creative work.

The combination of a neural model with latent knowledge of English, and conditioning the neural language model on a specific generation task seems a worthwhile direction for procedural content generation for video games, provided we can find the data to condition the neural model on. In order to teach the model a generation task, we need to have meta-data that describes what kind of output we want. Generating video game text that fits within a specific game context requires training data that includes this information. However, video game corpora are scarce to begin with, even without additional contextualization. In Chapter 8 we extended the data collection method from Chapter 7: we investigated where similar datasets could be found, and what good properties for those datasets would be. We showed how we can obtain video game texts using different approaches: from original game files or from fan-websites. If a game is open source, its texts can often be extracted directly from the game files. If the game is not open source, we can try to obtain game texts using either official, publisher-sanctioned modification software, or tools created by the video games modding community. We also presented three new datasets obtained via these approaches. The datasets contained text from three commercial role-playing games: all in-game text from *Torchlight II*, dialogue from *Star Wars: Knights of the Old Republic* and in-game books from *The Elder Scrolls*. The datasets can be used for a variety of applications in NLP and NLG, included style analysis, linguistic analysis, sentiment analysis, research on gender and (fantasy) race representation, branching dialogue generation and quest generation.

9.2 Revisiting the research questions

Section 1.5 stated the following research goal for this dissertation:

Research goal. *To investigate and develop methods for generating coherent flavor text for video games, while keeping in mind the risks and costs.*

This research goal was the common thread throughout the work of this thesis. The goal was implemented via a set of research questions. We will now answer these research questions using the findings from this thesis.

As stated in Section 1.3, the existing work on text generation for games is limited. Since flavor text is fiction, we started by investigating the existing methods for generating coherent works of fiction. We then assessed whether we can transfer these methods from the domain of fiction to generating coherent flavor text.

RQ 1. *Which methods have been used in the wild for generating coherent works of fiction?*

This question was answered in Chapter 4, where we analyzed the contributions of text generation challenge NaNoGenMo 2018 as a case study. As we saw in the introduction, some procedural generation techniques are hard to use in practice, for example because of lack of transparency, required expert knowledge or operational cost. We specifically looked at techniques used in NaNoGenMo contributions to make sure the techniques were easy to use in practice. The challenge runs only for 30 days, which precludes generators that are overly complex or labor-intensive to build, and most participants are not natural language processing experts.

We found that NaNoGenMo text generators that could create coherent long-form narratives used one of four approaches: hardcoding the full narrative in a domain-specific language, hardcoding parts of the narrative, such as a beginning or ending, creating emergent narrative from rule-systems, and evoking a sense of coherence by playing with the expectations of the reader.

Of course, the findings of this chapter only represent a snapshot of one creative community at a specific point in time. There are probably more approaches for than those four that we found. None of the approaches were new from a natural language processing research point of view. Additionally, popular methods for text generation will probably have changed in subsequent years, for example because of the availability of large-scale pre-trained neural language models such as GPT-2. Nonetheless, I believe it is worthwhile for researchers to monitor the use of generative techniques outside of published research. Studying other creative communities, such as those involving Twitter bots, procedural games, poetry generation, and interactive fiction, might lead to new insights for text generation research. Our contribution was analyzing how NLG techniques are used by creative practitioners for creating fiction. NaNoGenMo contributions can show us which techniques are easy to deploy and intuitive to understand for non-experts. Additionally, some of the programming tools and framing tricks used by the participants might not yet be known to academic researchers.

RQ 2. *How can these methods be adapted for generating coherent flavor text for video games?*

This research question was addressed in Chapter 4 and Chapter 5. In Chapter 4 we translated the four approaches we found to the video games domain. We found that all four approaches used by NaNoGenMo participants are already used in existing commercial games. The last method, evoking coherence, is the least used in video games, and is the most interesting to explore further because of its low cost and potentially high impact. In Chapter 5 we explored this approach in practice with our headline generator Churnalist. Churnalist tries to evoke coherence in its outputs by reusing words and phrases from its input. Indeed, our evaluation showed that word reuse works for increasing perceived coherence

of generated artifacts: evaluation participants preferred all headlines that re-use words from the input, not just the headlines generated by Churnalist, when rating headlines on their meaningfulness and the plausibility that they were part of a video game.

RQ 3. *How can we use Transformer-based language models to generate flavor text for video games?*

Transformer-based models have shown that they can create coherent text even without any fine-tuning. For example, various journalists have used GPT-2 to generate news articles, although most seem to use a manual generate-and-curate process, where multiple paragraphs are generated, and the best are hand-picked by an editor for further editing. This method can not be copied for generating flavor text. Flavor text for video games should fit with the larger game narrative and the in-game context in which it is shown to the player. Not all flavor text is prose text, so it should also have the expected form. When given a certain prompt, GPT-2 typically generates multiple paragraphs of texts in the same format as the prompt, e.g. HTML tags in the prompt will lead to HTML tags in the output, whereas a prose prompt tends to result in more English prose. To ensure both the coherence and proper format of flavor text generated by GPT-2, we fine-tuned GPT-2 on a implicit natural language generation task, using special tags and task-specific corpora. The same approach had already been used in earlier GPT-2 work on machine translation and patent generation. GPT-2 is an opaque, non-transparent language model, so the internals of the language model are hard to inspect. Researchers are currently using statistical analysis to analyze neural models for latent biases and other problems. We expect that these efforts will lead to additional ways to fine-tune and guide GPT-2 towards generating a specific piece of text. However, the fine-tuning and generation method that we used, i.e. using special tags to guide the language model, seems the best way to generate specific artifacts with GPT-2, such as coherent flavor text.

We tested this approach for the video games domain in Chapter 6 and Chapter 7. Both chapters feature a neural language model fine-tuned on task-specific data. In Chapter 6 we presented Churnalist version 2, which generates headlines from GPT-2 fine-tuned on GigaWord headlines. In Chapter 7, we generated flavor text dialogues using GPT-2 fine-tuned on hand-written World of Warcraft quests.

In both cases, we expanded the example texts in the training set with special tags and annotations. By fine-tuning GPT-2 on these annotated examples, the language model learned an implicit generative task. It learned that a valid output should contain the special tags, and that there is an implicit relation between the annotations and the text. The tags could then be used to steer the generative model towards a desired output. In the case of Churnalist version 2, the language model learned that the phrase after `<|object|>` should also occur in the output headline. We used this to generate headlines that contain user-specified seed words. We conducted an error analysis of 300 generated flavor text headlines that were created with Churnalist version 2, which showed

that the language quality problems prevalent in headlines from Churnalist version 1 had largely disappeared. Most headlines were grammatical and could be used as flavor text without manual modifications by a human editor. However, a new type of problem appeared: the neural language model sometimes ‘hallucinates’ English words that are not in the prompt. This was especially the case for prompts that include rare English words, or non-English fantasy words.

In the case of the language model fine-tuned on World of Warcraft quests, the language model learned that a quest title and objective are followed by a short piece of dialogue in the style that is specific to World of Warcraft. We evaluated 10 generated flavor text dialogues with an online survey on the properties of language quality, coherence, novelty, surprise and creativity. The generated quests were rated as significantly worse on language quality, coherence and novelty. The ratings for surprise and creativity were close to those of human-written NPC dialogues.

Can we conclude that large neural language models are the solution for generating flavor text? If the prompt and the desired flavor text contain only regular English language text, then probably yes. GPT-2 was trained on such a large corpus of web text that it has a good grasp of the English language, and sometimes even shows something resembling common sense logic.

Our experiments in generating flavor text headlines showed that if we train the model on a corpus of day-to-day English text, the fine-tuned language model has difficulties processing prompts that include non-English words. This can be a problem when we want to generate video game texts, as these often contain fantasy words.

We tried to mitigate this problem in our experiments with generating World of Warcraft flavor text dialogues. This time, we fine-tuned GPT-2 on a corpus of video game specific text sourced from the target game. The results were very different from the first experiment: generated NPC dialogues contained many references to fictional names and locations, races, and the specific style of dialogue found in World of Warcraft. The language model could now interpret non-English words, and refer to characters, objects and locations that were contained in the prompt. However, the generated flavor text dialogues can still be greatly improved. The dialogues show little inner logic. Generated dialogues might contain references to existing World of Warcraft people, objects or locations even when these are not relevant or logical in the context of the quest. This is due to the low amount of game context that was used as training set and embedded in the prompt. The generator was only provided with the quest title and objective as context.

RQ 4. *What are the advantages and disadvantages of rule-based and Transformer-based approaches for generating coherent flavor text?*

We have tried to answer this question using three text generation systems: rule-based Churnalist in Chapter 5, and Churnalist version 2, which uses GPT-2,

in Chapter 6, and the NPC dialogue generator for World of Warcraft from Chapter 7.

In Chapter 5 we described Churnalist, a modular rule-based system for generating flavor text headlines. Churnalist was not meant to be innovative in terms of text generation methods. Instead, it was created for investigating our hypothesis that word substitution will lead to flavor text that is perceived to be coherent. Churnalist was a minimum viable prototype, in the sense that the method of text generation, i.e. word substitution, was kept as simple as possible. The generator's algorithm was mostly based on randomisation instead of linguistic knowledge, and the system was built using as many readily available sources (such as open source libraries and open data) as possible. We also focused on creating English flavor text as opposed to Dutch flavor text, as there are more resources available for English than for Dutch. However, even with these choices, developing Churnalist was not straightforward. During development, we ran into various practical problems: open source NLP libraries do not have perfect accuracy, some functionalities of NLP libraries might only be available for specific languages; and open data might contain noise or errors. These problems trickle down the pipeline and degenerate the quality of outputs. On top of that, building a complete system architecture from various moving parts turned out to be error-prone and labor-intensive.

Of course, Churnalist is only one rule-based system for one application. Rule-based systems have proven their worth in a variety of useful contexts, such as the medical domain. However, creating a rule-based system from scratch might be too laborious for various *creative* applications, where truthfulness and correctness are less important than when generating non-fiction text. Additionally, rule-based systems are often tailored to very specific applications, and might not easily adapt to other scenarios. This also counts to some extent for neural language models trained on a specific task. However, when we take into account the sheer size and variety of their training data, we can conclude that, broadly speaking, large-scale neural language models are more general than most rule-based systems. For these reasons, we estimate that neural language models are generally more useful for creative text generation applications.

In Chapter 6 we presented Churnalist version 2, which generated headlines from GPT-2 fine-tuned on GigaWord headlines. Developing this system only required fine-tuning GPT-2, which was much faster than developing Churnalist version 1, even when we take into account the computational cost of fine-tuning. Training the model only required a large enough dataset with text examples, in this case the annotated Gigaword headlines.

We analyzed headlines from Churnalist version 2 on language quality, and compared them to the headlines created by Churnalist version 1. We found that the neural language model leads to outputs with higher language quality compared to the rule-based system. However, using the neural model also had disadvantages: the generator is not transparent, in contrast to its rule-based pre-

decessor, and the generator does not react well to non-English words in the input. Fine-tuning a neural language model for the video games domain also requires a large corpus of example texts, which can be problematic as not many corpora are currently available.

In Chapter 7, we generated flavor text dialogues using GPT-2 fine-tuned on hand-written World of Warcraft quests. Choosing World of Warcraft meant that we had ample training data to work with. The game's size, longevity and popularity all result in an abundance of online resources, such as guides and wikis, for players – which we can scrape to obtain a corpus of example texts.

In both neural flavor text generation systems, GPT-2's latent knowledge of language resulted in the creation of acceptable quality surface text. Especially if there is a human in the loop, the advantages of neural language models, such as fast development and good quality surface text, outweigh the drawbacks of neural language models, such as opacity. Using neural language models requires some technical expertise, but so does building a real rule-based system of sufficient linguistic quality to be of use in practice. The biggest practical drawback of using neural language models for generating fiction is their training data requirement. However, new developments in neural language models are being made very rapidly. Google is investigating increasingly smaller models, and recent work has focused on statistical analyses of models to assess robustness and hidden biases. Newer models might require less training data for learning a specific generation task, or might be able to deal with out-of-vocabulary words in a more robust way. Currently, we cannot yet say with specificity how large a training set should be at the very minimum before it is useful for finetuning a neural model.

RQ 5. *If we want to generate coherent flavor text using data-driven NLG methods, what are the requirements for training data, and where can we find data that fulfill these requirements?*

This question was addressed in Chapter 8. We proposed a list of requirements for video games based on our experiences with fine-tuning neural language models: richness, representativeness, diversity and portability. Richness, representativeness and portability are requirements for single datasets. In our experiments with generating flavor text for quests, we noticed better results when we used a homogeneous dataset when fine-tuning GPT-2 on a generation task. This suggests that diversity within one dataset is not necessarily a good thing. However, our rationale for diversity as requirement still applies to all datasets in a research field as a whole: the datasets that we use for video games research should reflect the diversity of the video games domain. In this chapter we also described how new video game text corpora can be collected. They can be scraped from fan-websites, or obtained from original game files, either by working with open source games, official modding tools or community-built modding tools. To demonstrate the feasibility of our approach, we collected three datasets with

texts from commercial video games. It is also possible to create new or expanded corpora by taking existing corpora and creating additional synthetic datapoints, for example by masking part of the data, or extracting information from video game paratext, such as reviews or Twitch video streams.

9.3 Directions for future work

This thesis explored flavor text generation for video games. The breadth of video games and text generation research meant that the scope of the research was limited to specific video game genres and text generation methods. We focused on two specific methods for text generation, namely modular rule-based systems in Chapter 5, and GPT-2 in Chapter 6 and 7. This thesis also explored how resources created by online communities, such as fan-wikis, modding software and creative open source projects, can contribute to academic games research. Because video games are such a broad medium, we limited ourselves to narrative-heavy commercial role-playing games. These limitations mean that further work is needed to assess whether the findings of this thesis also apply to other types of games. In this section we will discuss ideas for future work.

Neural language models

Even though we used large-scale neural language models for text generation, we focused on one specific architecture, i.e. GPT-2, and one specific implementation for that architecture, i.e. `gpt-2-simple`. Research into complex language models has gained considerable momentum, and new and better language models are being developed every month. Future research needs to explore expressive power, modeling capabilities and training methods for other neural language models, such as BERT, while taking the requirements specific to procedural generation for video games into account.

In the context of GPT-2, it would be interesting to see how we can incorporate additional contextual data during generation, such as a player model or aspects of the game state. This could improve the coherence of generated assets with the rest of the game world. Our experiment with generating World of Warcraft dialogues showed that it is not enough if a language model just copies the writing style of a particular game. The dialogues generated by our fine-tuned language model contained names of characters and locations, but the language model has no knowledge of these entities, which makes it hard to refer to them in a realistic context. For example, the language model might refer to far-away in-game locations as if they are situated close by, or it might refer to dead characters as if they were still alive. It could also generate positive sentences about negative events or evil characters. This is not a critical problem if the language model is generating content under supervision of a human (writer), who has knowledge of the game world and a clear definition of valid content. This behaviour is also

unproblematic if the goal of the language model is to generate large volumes of content that can be used as a basis for further refinement by human writers. All our language models were used for offline generation in the context of human-computer co-creation, where the human user influences and curates the outputs of the generator. In future, we should also try to develop generative models (1) that are suitable for online generation and (2) that do not require human supervision, such as used in online game AI Dungeon [282]. This would significantly broaden the possible applications of large language models.

More NLP methods should be tested specifically on video game data. There is a data bias in NLP: researchers tend to use the same corpora again and again. This has advantages such as comparability, ease, etc. but also leads to wildly varying results on corpora from other domains than the benchmarks, e.g. sentiment analysis on things that are not product reviews. This one-sidedness could decrease the robustness of NLP techniques across domains. This data bias should be mitigated by testing standard NLP tasks on video game data, and confirming that tasks that are considered to be ‘solved’ in NLP research perform similarly on video game data as on benchmark data. Further developments in NLP for video game texts could then be used for large-scale analysis of game data, which in turn could benefit NLG for video games. For example, we could automatically annotate new corpora with game text, and use the annotations for fine-tuning language models for text generation.

Video game corpora

In the research literature we can find training sets for neural language models that range from a few hundred datapoints to millions of datapoints. It would be useful to know how large new video game corpora should be, if they are to be used for fine-tuning large-scale language models such as GPT-2. As the choice for a specific optimization algorithm, neural network architecture and goal task probably also influence the optimal training set size, a systematic exploration of minimal training set sizes for various NLP tasks and models would be useful reference material for researchers that want to create new corpora.

Of the four video game corpora that were created for this thesis, only two were already used in a research project. The quest dialogue generator from Chapter 7 was trained on the World of Warcraft dataset, and in an earlier work [26], we used the dataset with documents from The Elder Scrolls to create a game-specific lexicon for sentiment analysis. Our corpora could be used in many other research projects. The dataset with Knights of the Old Republic dialogue could be used for the creation of a branching dialogue generator, using a similar fine-tuning method as was used for the World of Warcraft dialogue generator. The Torchlight II dataset is particularly versatile because of the various categories of in-game text it contains. Torchlight II is a typical example of action role-playing games in the hack-and-slash subgenre, which means that similar games, such as Diablo [40], Sacred [9], Titan Quest [140] and Path of Exile [120], might

contain the same type of text. We could employ these similarities for investigating the generalizability of fine-tuned language models to different game worlds. To start, the Torchlight II data could be combined with the World of Warcraft dataset to see if we can generate Torchlight II quests using World of Warcraft data, and vice versa. The corpus could also be used for quest generation, similar to the research in Chapter 7. Outside of a text generation context, the corpora could also be used for natural language processing research, e.g. for validating solutions to common NLP tasks on special-domain corpora, or for text analytics as part of game studies, e.g. research on gender stereotypes in video games or linguistic properties of fantasy languages. Future research could apply our method for extracting Star Wars: Knights of the Old Republic dialogue to other games created with Bioware’s Aurora game engine, such as Mass Effect [35], Dragon Age: Origins [34] and The Witcher [59]. The assets from these games will probably all have a similar structure, because their games were built on the same game engine. As the games assets share the same structure, but have very different contents, this would create corpora that are particularly interesting for investigating cross-training neural models on different texts, and assessing the transferability and generalizability of the fine-tuned language models.

Narrative coherence for video games

As we discussed in the introduction, narrative coherence played an important role in this thesis. However, we found that regular definitions of coherence were not directly applicable to games, which is why we introduced the concept of ‘game-level coherence’. A more formal definition of what coherence means in the context of games would be useful, as it could be used to develop methods for automatically measuring narrative coherence. Whether we want to measure the narrative coherence of regular, linear texts or interconnected game texts, this is also an area worth investigating. As far as we know, there currently exist no methods for automatically assessing syntactic and semantic coherence in longer texts. Scheuter [232] did some interesting preliminary work in this direction in 2021. Such an automatic assessment would have been useful for our survey of NaNoGenMo contributions, as we could have automatically selected the most interesting and most coherent contributions from NaNoGenMo participants. Automatically assessing the narrative coherence of video game assets could be useful for procedural content generation research. However, creating automatic assessments that are generalizable, instead of game-specific or genre-specific, will remain challenging as game development is such a dynamic, fast-changing practice.

Bibliography

- [1] A.I. Design. *Rogue*. Game [various platforms]. Epyx, San Francisco, California, US. 1980.
- [2] Tarn Adams and Zach Adams. *Dwarf Fortress*. Game [PC]. Bay 12 Games, Silverdale, WA, USA. Played May 2018. Bay 12 Games, 2002.
- [3] David H. Ahl. *BASIC computer games: Microcomputer edition*. 1978.
- [4] Khalid Alnajjar, Leo Leppänen, Hannu Toivonen, et al. “No time like the present: methods for generating colourful and factual multilingual news headlines”. In: *Proceedings of the 10th International Conference on Computational Creativity*. Association for Computational Creativity. 2019.
- [5] Alba Amato. “Procedural content generation in the game industry”. In: *Game Dynamics*. Ed. by Oliver Korn and Newton Lee. Springer, 2017, pp. 15–25.
- [6] Prithviraj Ammanabrolu, William Broniec, Alex Mueller, Jeremy Paul, and Mark O. Riedl. “Toward Automated Quest Generation in Text-Adventure Games”. In: *Proceedings of the 4th Workshop on Computational Creativity in Language Generation*. Tokyo, Japan: Association for Computational Linguistics, Oct. 2019, pp. 1–12. URL: <https://www.aclweb.org/anthology/2019.ccnlg-1.1>.
- [7] Barrett R. Anderson and Adam M. Smith. “The Undergraduate Games Corpus: A Dataset for Machine Perception of Interactive Media”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.1 (May 2021), pp. 3–11. URL: <https://ojs.aaai.org/index.php/AAAI/article/view/16071>.
- [8] Wissam Antoun, Fady Baly, and Hazem Hajj. “AraGPT2: pre-trained transformer for Arabic language generation”. In: *arXiv preprint arXiv:2012.15520* (2020).
- [9] Ascaron Entertainment. *Sacred*. Game [PC]. Encore, San Jose, California, US. San Jose, California, US, Mar. 2004.

- [10] Maren Awiszus, Frederik Schubert, and Bodo Rosenhahn. “World-GAN: a Generative Model for Minecraft Worlds”. In: *CoRR* abs/2106.10155 (2021). arXiv: 2106.10155. URL: <https://arxiv.org/abs/2106.10155>.
- [11] Ruth Aylett. “Narrative in virtual environments-towards emergent narrative”. In: *Proceedings of the AAAI fall symposium on narrative intelligence*. 1999, pp. 83–86.
- [12] Ruth Aylett, Sandy Louchart, Joao Dias, Ana Paiva, and Marco Vala. “FearNot!—an experiment in emergent narrative”. In: *International Workshop on Intelligent Virtual Agents*. Springer. 2005, pp. 305–316.
- [13] Kenny Backus. “Managing output: boredom versus chaos”. In: *Procedural Generation in Game Design*. Ed. by Tanya X Short and Tarn Adams. AK Peters/CRC Press, 2017. Chap. 2, pp. 13–21.
- [14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. “Neural Machine Translation by Jointly Learning to Align and Translate”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015. URL: <http://arxiv.org/abs/1409.0473>.
- [15] Andrea Ballatore, Michela Bertolotto, and David C. Wilson. “An Evaluative Baseline for Geo-Semantic Relatedness and Similarity”. In: *Geoinformatica* 18.4 (Oct. 2014), pp. 747–767. ISSN: 1384-6175. DOI: 10.1007/s10707-013-0197-8. URL: <https://doi.org/10.1007/s10707-013-0197-8>.
- [16] David Bamman, Brendan O’Connor, and Noah A. Smith. “Learning Latent Personas of Film Characters”. In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Sofia, Bulgaria: Association for Computational Linguistics, Aug. 2013, pp. 352–361. URL: <https://www.aclweb.org/anthology/P13-1035>.
- [17] Michele Banko, Vibhu O Mittal, and Michael J Witbrock. “Headline generation based on statistical translation”. In: *Proceedings of the 38th Annual Meeting of ACL*. Association for Computational Linguistics. 2000, pp. 318–325.
- [18] Eric Barone. *Stardew Valley*. Game [PC]. ConcernedApe, Los Angeles, California, United States and Chucklefish, London, United Kingdom. Played November 2016. Los Angeles, California, United States, Feb. 2016.
- [19] Nuno Barreto, Amílcar Cardoso, and Licínio Roque. “Computational creativity in procedural content generation: A state of the art survey”. In: *Proceedings of the 2014 conference of science and art of video games*. 2014.
- [20] Gabriella A. B. Barros, Antonios Liapis, and Julian Togelius. “Data adventures”. In: *Proceedings of the FDG workshop on Procedural Content Generation in Games, Pacific Grove*. Association for Computing Machinery. 2015.

- [21] Gabriella A. B. Barros, Antonios Liapis, and Julian Togelius. “Murder mystery generation from open data”. In: *Proceedings of the International Conference on Computational Creativity*. 2016.
- [22] Gabriella A.B. Barros, Michael Cerny Green, Antonios Liapis, and Julian Togelius. “Who killed albert einstein? From open data to murder mystery games”. In: *IEEE Transactions on Games* (2018).
- [23] Chris Bateman. *Game writing: Narrative skills for videogames*. Charles River Media, 2007.
- [24] Anja Belz. “Fully Automatic Journalism: We Need to Talk About Nonfake News Generation.” In: *TTO*. 2019.
- [25] Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. “On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?” In: *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*. FAccT ’21. Virtual Event, Canada: Association for Computing Machinery, 2021, pp. 610–623. ISBN: 9781450383097. DOI: 10.1145/3442188.3445922. URL: <https://doi.org/10.1145/3442188.3445922>.
- [26] Thérèse Bergsma, Judith van Stegeren, and Mariët Theune. “Creating a Sentiment Lexicon with Game-Specific Words for Analyzing NPC Dialogue in The Elder Scrolls V: Skyrim”. English. In: *Workshop on Games and Natural Language Processing*. Marseille, France: European Language Resources Association, May 2020, pp. 1–9. ISBN: 979-10-95546-40-5. URL: <https://www.aclweb.org/anthology/2020.gamnlp-1.1>.
- [27] Bethesda Game Studios. *Radiant story generation*. Software, part of The Elder Scrolls IV: Skyrim. Bethesda Softworks, Rockville, Maryland, US. Bethesda Softworks.
- [28] Bethesda Game Studios. *The Elder Scrolls III: Morrowind*. Game [PC]. Bethesda Softworks, Rockville, Maryland, US. Bethesda Softworks, 2002.
- [29] Bethesda Game Studios. *The Elder Scrolls IV: Oblivion*. Game [PC]. Bethesda Softworks, Rockville, Maryland, US and 2K Games, Novato, California, US. Bethesda Softworks and 2K Games, 2006.
- [30] Bethesda Game Studios. *The Elder Scrolls V: Skyrim*. Game [PC]. Bethesda Softworks, Rockville, Maryland, US. Bethesda Softworks, 2011.
- [31] Bethesda Softworks. *The Elder Scrolls I-V and The Elder Scrolls Online*. Game series [PC]. Bethesda Softworks, Rockville, Maryland, US. Bethesda Softworks, 1994–2014.
- [32] Bethesda Softworks. *The Elder Scrolls II: Daggerfall*. Game [PC]. Bethesda Softworks, Rockville, Maryland, US. Bethesda Softworks, 1996.
- [33] Bethesda Softworks. *The Elder Scrolls: Arena*. Game [MS-DOS]. Bethesda Softworks, Rockville, Maryland, US. Bethesda Softworks, 1994.

- [34] BioWare. *Dragon Age: Origins*. Game [PC]. Electronic Arts, California, US. 2009.
- [35] BioWare. *Mass Effect*. Game [PC]. 2007.
- [36] BioWare. *Star Wars: Knights of the Old Republic*. Game [PC]. LucasArts, San Francisco, US. LucasArts, 2003.
- [37] Michael Blatz and Oliver Korn. “A Very Short History of Dynamic and Procedural Content Generation”. In: *Game Dynamics*. Springer, 2017, pp. 1–13.
- [38] Blizzard Entertainment. *Diablo III*. Game [PC]. Blizzard Entertainment, Irvine, California, United States. Played June 2012. Irvine, California, United States, May 2012.
- [39] Blizzard Entertainment. *World of Warcraft*. Game [PC]. Blizzard Entertainment, Irvine, California, US. Blizzard Entertainment, 2004.
- [40] Blizzard North. *Diablo*. Game [PC]. Blizzard Entertainment, Irvine, California, U.S. Irvine, California, U.S., Jan. 1997.
- [41] Margaret A. Boden. “Creativity in a nutshell”. In: *Think* 5.15 (2007), pp. 83–96. ISSN: 1477-1756. DOI: 10.1017/s147717560000230x.
- [42] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. “Enriching Word Vectors with Subword Information”. In: *Transactions of the Association of Computational Linguistics* 5 (2017), pp. 135–146.
- [43] Richard Boon. “Writing for games”. In: *Game writing: Narrative skills for videogames*. Ed. by Chris Bateman. Charles River Media, 2007.
- [44] Hudson Borges and Marco Tulio Valente. “What’s in a GitHub star? understanding repository starring practices in a social coding platform”. In: *Journal of Systems and Software* 146 (2018), pp. 112–129.
- [45] David Braben and Ian Bell. *Elite*. Game [various platforms]. Acornsoft, Cambridge, England. Cambridge, England, Sept. 1984.
- [46] Daniel Braun, Ehud Reiter, and Advait Siddharthan. “SaferDrive: An NLG-based behaviour change support system for drivers”. In: *Natural Language Engineering* 24.4 (2018), pp. 551–588.
- [47] Daniel Braun, Anupama Sajwan, and Florian Matthes. “User-adaptable Natural Language Generation for Regression Testing within the Finance Domain”. In: *Proceedings of the 22nd International Conference on Enterprise Information Systems - Volume 1: ICEIS, INSTICC*. SciTePress, Jan. 1, 2020, pp. 613–618. ISBN: 978-989-758-423-7. DOI: 10.5220/0009563306130618. URL: https://www.daniel-braun.com/wp-content/uploads/2020/05/ICEIS_2020_198.pdf. published.

- [48] Scott Brisson, Ehsan Noei, and Kelly Lyons. “We Are Family: Analyzing Communication in GitHub Software Repositories and Their Forks”. In: *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 2020, pp. 59–69. DOI: 10.1109/SANER48275.2020.9054834.
- [49] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. “Language Models are Few-Shot Learners”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901. URL: <https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfc4967418bfb8ac142f64a-Paper.pdf>.
- [50] Bungie. *Destiny*. Game [PlayStation, Xbox]. Bungie, Bellevue, Washington, US. Activision and Bungie, 2014.
- [51] Charles B Callaway and James C Lester. “Narrative prose generation”. In: *Artificial Intelligence* 139.2 (2002), pp. 213–252.
- [52] Giuseppe Carenini and Johanna D Moore. “Generating and evaluating evaluative arguments”. In: *Artificial Intelligence* 170.11 (2006), pp. 925–952.
- [53] Maria Fernanda Caropreso, Diana Inkpen, Fazel Keshtkar, and Shahzad Khan. “Template Authoring Environment for the Automatic Generation of Narrative Content”. In: *Journal of Interactive Learning Research* 23.3 (2012), pp. 227–249.
- [54] Maria Fernanda Caropreso, Diana Inkpen, Shahzad Khan, and Fazel Keshtkar. “Automatic generation of narrative content for digital games”. In: *2009 International Conference on Natural Language Processing and Knowledge Engineering*. IEEE. 2009, pp. 1–8.
- [55] Isamar Carrillo Masso. “Developing a methodology for corpus-based computer game studies”. In: *Journal of Gaming & Virtual Worlds* 1.2 (2009), pp. 143–169.
- [56] Thiago Castro Ferreira, Diego Moussallem, Emiel Krahmer, and Sander Wubben. “Enriching the WebNLG corpus”. In: *Proceedings of the 11th International Conference on Natural Language Generation*. Tilburg University, The Netherlands: Association for Computational Linguistics, Nov. 2018, pp. 171–176. DOI: 10.18653/v1/W18-6521. URL: <https://aclanthology.org/W18-6521>.

- [57] Marc Cavazza and Fred Charles. “Dialogue Generation in Character-based Interactive Storytelling.” In: *AIIDE*. 2005, pp. 21–26.
- [58] CCP Games. *EVE Online*. Game [PC]. CCP Games, Reykjavík, Iceland. CCP Games, 2003.
- [59] CD Projekt Red. *The Witcher*. Game [PC]. Atari, Inc., New York City, New York, US and CD Projekt Red, Łódź, Poland. CD Projekt Red, 2007.
- [60] CD Projekt Red. *The Witcher 3: Wild Hunt*. Game [PC]. CD Projekt, Warsaw, Poland. Warsaw, Poland, May 2015.
- [61] Cellar Door Games. *Rogue Legacy*. Game [PC]. Cellar Door Games, Toronto, Ontario, CA. 2013.
- [62] Wenhui Chen, Yu Su, Xifeng Yan, and William Yang Wang. “KGPT: Knowledge-Grounded Pre-Training for Data-to-Text Generation”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, Nov. 2020, pp. 8635–8648. DOI: 10.18653/v1/2020.emnlp-main.697. URL: <https://aclanthology.org/2020.emnlp-main.697>.
- [63] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. “On the Properties of Neural Machine Translation: Encoder–Decoder Approaches”. In: *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 103–111. DOI: 10.3115/v1/W14-4012. URL: <https://www.aclweb.org/anthology/W14-4012>.
- [64] Noam Chomsky. “Three models for the description of language”. In: *IRE Transactions on information theory* 2.3 (1956), pp. 113–124.
- [65] *Choose Your Own Adventure*. Gamebook series. 1979–1998 (original series). Bantam Books, 1979.
- [66] Andry Chowanda, Martin Flintham, Peter Blanchfield, and Michel Valstar. “Playing with Social and Emotional Game Companions”. In: *Intelligent Virtual Agents*. Ed. by David Traum, William Swartout, Peter Khooshabeh, Stefan Kopp, Stefan Scherer, and Anton Leuski. Cham: Springer International Publishing, 2016, pp. 85–95. ISBN: 978-3-319-47665-0.
- [67] Carlos A Colmenares, Marina Litvak, Amin Mantrach, and Fabrizio Silvestri. “Heads: Headline generation as sequence prediction using an abstract feature-rich space”. In: *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2015, pp. 133–142.
- [68] Colossal Order. *Cities: Skylines*. Game [PC]. Paradox Interactive, Stockholm, Sweden. Paradox Interactive, 2017.

- [69] Simon Colton and Geraint A. Wiggins. “Computational Creativity: The Final Frontier?” In: *Proceedings of the 20th European Conference on Artificial Intelligence*. ECAI’12. Montpellier, France: IOS Press, 2012, pp. 21–26. ISBN: 9781614990970.
- [70] Kate Compton, Quinn Kybartas, and Michael Mateas. “Tracery: an author-focused generative text tool”. In: *International Conference on Interactive Digital Storytelling*. Springer. 2015, pp. 154–161.
- [71] Kate Compton and Michael Mateas. “Casual Creators”. In: *Proceedings of the Sixth International Conference on Computational Creativity* June. 2015, p. 228.
- [72] Michael Cook. “A vision for continuous automated game design”. In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. Vol. 13. 1. 2017.
- [73] Michael Cook. “Procedural Generation and Information Games”. In: *2020 IEEE Conference on Games (CoG)*. IEEE. 2020, pp. 253–260.
- [74] Michael Cook and Simon Colton. “A rogue dream: Automatically generating meaningful content for games”. In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. Vol. 10. 1. 2014.
- [75] Michael Cook and Simon Colton. “Neighbouring Communities: Interaction, Lessons and Opportunities”. In: *International Conference on Computational Creativity*. Salamanca, Spain, 2018.
- [76] Michael Cook, Jeremy Gow, Gillian Smith, and Simon Colton. “Danesh: Interactive Tools For Understanding Procedural Content Generators”. In: *IEEE Transactions on Games* (2021), pp. 1–1. DOI: 10.1109/TG.2021.3078323.
- [77] Paul A Crutcher. “Magic: The Gathering, A Literary Text”. In: *Americana: The Journal of American Popular Culture (1900-present)* 16.1 (2017). https://www.americanpopularculture.com/journal/articles/spring_2017/crutcher.htm, retrieved April 9, 2021.
- [78] Steve Dahlskog and Julian Togelius. “Patterns and Procedural Content Generation: Revisiting Mario in World 1 Level 1”. In: *Proceedings of the First Workshop on Design Patterns in Games*. DPG ’12. Raleigh, North Carolina, USA: Association for Computing Machinery, 2012. ISBN: 9781450318549. DOI: 10.1145/2427116.2427117. URL: <https://doi.org/10.1145/2427116.2427117>.
- [79] Richard Dansky. “Introduction to game narrative”. In: *Game writing: Narrative skills for videogames*. Ed. by Chris Bateman. Charles River Media, 2007.

- [80] Barbara De Kegel and Mads Haahr. “Procedural puzzle generation: a survey”. In: *IEEE Transactions on Games* 12.1 (2019), pp. 21–40.
- [81] Kees van Deemter, Emiel Krahmer, and Mariët Theune. “Squibs and Discussions: Real versus Template-Based Natural Language Generation: A False Opposition?” In: *Computational Linguistics* 31.1 (2005), pp. 15–24. DOI: 10.1162/0891201053630291. URL: <https://www.aclweb.org/anthology/J05-1002>.
- [82] Natlie Dehn. “Story Generation After TALE-SPIN.” In: *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*. Vol. 81. A. Drinan (ed.). 24-28 August 1981, University of British Columbia Vancouver, B.C., Canada. IJCAI, 1981, pp. 16–18.
- [83] Wendy Despain. *Professional techniques for video game writing*. Second. CRC Press, 2020.
- [84] Wendy Despain. *Writing for video game genres: From FPS to RPG*. CRC Press, 2009.
- [85] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: <https://www.aclweb.org/anthology/N19-1423>.
- [86] Bruno Dias. *Voyageur*. Game [PC]. Game at itch.io. <https://brunodias.itch.io/voyageur>. 2017.
- [87] Jonathon Doran and Ian Parberry. “A Prototype Quest Generator Based on a Structural Analysis of Quests from Four MMORPGs”. In: *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*. PCGames ’11. Bordeaux, France: Association for Computing Machinery, 2011. ISBN: 9781450308724. DOI: 10.1145/2000919.2000920. URL: <https://doi.org/10.1145/2000919.2000920>.
- [88] Bonnie Dorr, David Zajic, and Richard Schwartz. “Hedge Trimmer: A Parse-and-Trim Approach to Headline Generation”. In: *Proceedings of the HLT-NAACL 03 Text Summarization Workshop*. 2003, pp. 1–8. URL: <https://www.aclweb.org/anthology/W03-0501>.
- [89] Mikhail G Dozmorov. “GitHub statistics as a measure of the impact of open-source bioinformatics software”. In: *Frontiers in bioengineering and biotechnology* 6 (2018), p. 198.

- [90] Ondřej Dušek, Jekaterina Novikova, and Verena Rieser. “Findings of the E2E NLG Challenge”. In: *Proceedings of the 11th International Conference on Natural Language Generation*. Tilburg University, The Netherlands: Association for Computational Linguistics, Nov. 2018, pp. 322–328. DOI: 10.18653/v1/W18-6539. URL: <https://www.aclweb.org/anthology/W18-6539>.
- [91] Doug Dymment. *Hamurabi*. Game [various platforms]. 1968.
- [92] Eidos Montréal. *Deus Ex: Human Revolution*. Game [PC]. Square Enix, Shinjuku, Tokyo, Japan. Square Enix, 2011.
- [93] Amin Fadaeddini, Babak Majidi, and Mohammad Eshghi. “A Case Study of Generative Adversarial Networks for Procedural Synthesis of Original Textures in Video Games”. In: *2018 2nd National and 1st International Digital Games Research Conference: Trends, Technologies, and Applications (DGRC)*. 2018, pp. 118–122. DOI: 10.1109/DGRC.2018.8712070.
- [94] Tamás Fekete and Ádám Porkoláb. “From Arkngthand to Wretched Squalor: Fictional place-names in The Elder Scrolls universe”. In: *ICAME Journal* 43.1 (2019), pp. 23–58.
- [95] Freehold Games. *Caves of Qud*. Game [PC/Mac/Linux]. Freehold Games, USA. Freehold Games, 2018.
- [96] Marie Gustafsson Friberger and Julian Togelius. “Generating game content from open data”. In: *Proceedings of the International Conference on the Foundations of Digital Games*. ACM. 2012, pp. 290–291.
- [97] Marie Gustafsson Friberger and Julian Togelius. “Generating interesting monopoly boards from open data”. In: *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*. IEEE. 2012, pp. 288–295.
- [98] Leon Fröhling and Arkaitz Zubiaga. “Feature-based detection of automated language models: tackling GPT-2, GPT-3 and Grover”. In: *PeerJ Computer Science* 7 (2021), e443.
- [99] Rubén H García-Ortega, Pablo García-Sánchez, Juan J Merelo, Aránzazu San-Ginés, and Ángel Fernández-Cabezas. “The story of their lives: Massive procedural generation of heroes’ journeys using evolved agent-based models and logical reasoning”. In: *European Conference on the Applications of Evolutionary Computation*. Springer. 2016, pp. 604–619.
- [100] Richard Garfield. *Magic: The Gathering*. Card game. 1993.
- [101] Albert Gatt and Emiel Krahmer. “Survey of the State of the Art in Natural Language Generation: Core tasks, applications and evaluation”. In: *Journal of Artificial Intelligence Research* 61 (2018), pp. 65–170.

- [102] Albert Gatt, Francois Portet, Ehud Reiter, Jim Hunter, Saad Mahamood, Wendy Moncur, and Somayajulu Sripada. “From data to text in the neonatal intensive care unit: Using NLG technology for decision support and information management”. In: *Ai Communications* 22.3 (2009), pp. 153–186.
- [103] Albert Gatt and Ehud Reiter. “SimpleNLG: A realisation engine for practical applications”. In: *Proceedings of the 12th European Workshop on Natural Language Generation (ENLG 2009)*. 2009, pp. 90–93.
- [104] Lorenzo Gatti, Gözde Özbal, Marco Guerini, Oliviero Stock, and Carlo Strapparava. “Heady-Lines: A Creative Generator Of Newspaper Headlines”. In: *Companion Publication of the 21st International Conference on Intelligent User Interfaces, IUI 2016*. 2016, pp. 79–83. DOI: 10.1145/2876456.2879469. URL: <http://doi.acm.org/10.1145/2876456.2879469>.
- [105] Lorenzo Gatti and Judith van Stegeren. “Improving Dutch sentiment analysis in Pattern”. In: *Computational Linguistics in the Netherlands Journal* 10 (Dec. 2020).
- [106] Samuel Gehman, Suchin Gururangan, Maarten Sap, Yejin Choi, and Noah A. Smith. “RealToxicityPrompts: Evaluating Neural Toxic Degeneration in Language Models”. In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, Nov. 2020, pp. 3356–3369. DOI: 10.18653/v1/2020.findings-emnlp.301. URL: <https://aclanthology.org/2020.findings-emnlp.301>.
- [107] Dominique Geissler, Elisa Nguyen, Daphne Theodorakopoulos, and Lorenzo Gatti. “Pokérator-Unveil your inner Pokémon”. In: *11th International Conference on Computational Creativity*. 2020.
- [108] Pablo Gervás. “Propp’s Morphology of the Folk Tale as a Grammar for Generation”. In: *Proceedings of the 2013 Workshop on Computational Models of Narrative*. Vol. 32. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2013.
- [109] Pablo Gervás. “Story Generator Algorithms”. In: *The Living Handbook of Narratology*. Hamburg: Hamburg University 19 (2012).
- [110] H Geuvers, W Geraedts, B Geron, and J van Stegeren. “A type system for Continuation Calculus”. In: *Proceedings Fifth International Workshop on Classical Logic and Computation, CL&C 2014, Vienna, Austria, July 13, 2014*. Ed. by Paulo Oliva. Vol. 164. EPTCS. 2014, pp. 1–18.
- [111] D. Gkatzia, V. Rieser, and O. Lemon. “How to talk to strangers: Generating medical reports for first-time users”. In: *2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. 2016, pp. 579–586. DOI: 10.1109/FUZZ-IEEE.2016.7737739.

- [112] Dimitra Gkatzia, Oliver Lemon, and Verena Rieser. “Natural Language Generation enhances human decision-making with uncertain information”. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 264–268. DOI: 10.18653/v1/P16-2043. URL: <https://www.aclweb.org/anthology/P16-2043>.
- [113] John J Godfrey, Edward C Holliman, and Jane McDaniel. “SWITCHBOARD: Telephone speech corpus for research and development”. In: *[Proceedings] ICASSP-92: 1992 IEEE International Conference on Acoustics, Speech, and Signal Processing*. Vol. 1. IEEE. 1992, pp. 517–520.
- [114] Eli Goldberg, Norbert Driedger, and Richard I Kittredge. “Using natural-language processing to produce weather forecasts”. In: *IEEE Expert* 9.2 (1994), pp. 45–53.
- [115] Hugo Gonalo Oliveira. “O Poeta Artificial 2.0: Increasing meaningfulness in a poetry generation twitter bot”. In: *Proceedings of the Workshop on Computational Creativity in Natural Language Generation (CC-NLG 2017)*. 2017, pp. 11–20.
- [116] Hugo Gonalo Oliveira. “PoeTryMe: a versatile platform for poetry generation”. In: *Proceedings of the ECAI 2012 Workshop on Computational Creativity, Concept Invention, and General Intelligence (C3GI at ECAI 2012)*. Montpellier, France, 2012.
- [117] Michael Cerny Green, Gabriella A.B. Barros, Antonios Liapis, and Julian Togelius. “DATA agent”. In: *Proceedings of the 13th International Conference on the Foundations of Digital Games*. ACM. 2018, p. 19.
- [118] Shai Gretz, Yonatan Bilu, Edo Cohen-Karlik, and Noam Slonim. “The workweek is the best time to start a family – A Study of GPT-2 Based Claim Generation”. In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, Nov. 2020, pp. 528–544. DOI: 10.18653/v1/2020.findings-emnlp.47. URL: <https://aclanthology.org/2020.findings-emnlp.47>.
- [119] Jason Grinblat and Brian Bucklew. “Subverting historical cause & effect: generation of mythic biographies in Caves of Qud”. In: *Proceedings of the 12th International Conference on the Foundations of Digital Games*. ACM. Hyannis, Massachusetts: ACM, 2017, pp. 1–7. ISBN: 978-1-4503-5319-9. DOI: 10.1145/3102071.3110574. URL: <http://doi.acm.org/10.1145/3102071.3110574>.
- [120] Grinding Gear Games. *Path of Exile*. Game [PC]. Grinding Gear Games, Auckland, New Zealand. Auckland, New Zealand, Oct. 2013.

- [121] The Guardian and Liam Porr. *A robot wrote this entire article. Are you scared yet, human?* The Guardian, Opinion, Artificial Intelligence (AI). Published September 8, 2020. Retrieved August 11, 2021 from <https://www.theguardian.com/commentisfree/2020/sep/08/robot-wrote-this-article-gpt-3>. Sept. 2020.
- [122] Manuel Guimaraes, Pedro Santos, and Arnav Jhala. “CiF-CK: An architecture for social NPCs in commercial games”. In: *2017 IEEE Conference on Computational Intelligence and Games (CIG)*. 2017, pp. 126–133. DOI: 10.1109/CIG.2017.8080425.
- [123] Matthew Guzdial, Brent Harrison, Boyang Li, and Mark O. Riedl. “Crowdsourcing Open Interactive Narrative.” In: *FDG*. 2015.
- [124] Gary Gygax and Dave Arneson. *Dungeons & Dragons*. Tabletop role-playing game. 1974.
- [125] Alon Halevy, Peter Norvig, and Fernando Pereira. “The unreasonable effectiveness of data”. In: *IEEE Intelligent Systems* 24.2 (2009), pp. 8–12.
- [126] Jason Andrew Hall, Benjamin Williams, and Christopher J Headleand. “Artificial Folklore for Simulated Religions”. In: *2017 International Conference on Cyberworlds (CW)*. IEEE. Chester, UK, 2017, pp. 229–232.
- [127] Mark Hendrikx, Sebastiaan Meijer, Joeri Van Der Velden, and Alexandru Iosup. “Procedural content generation for games: A survey”. In: *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 9.1 (2013), p. 1.
- [128] Frazer Heritage. “Applying corpus linguistics to videogame data: Exploring the representation of gender in videogames at a lexical level”. In: *Game studies* 20.3 (2020).
- [129] Douglas Hofstadter. “Preface 4: The Ineradicable Eliza Effect and Its Dangers”. In: *Fluid concepts and creative analogies: computer models of the fundamental mechanisms of thought*. Basic Books, New York, 1995.
- [130] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. “The Curious Case of Neural Text Degeneration”. In: *International Conference on Learning Representations*. 2020.
- [131] Ari Holtzman, Jan Buys, Maxwell Forbes, Antoine Bosselut, David Golub, and Yejin Choi. “Learning to Write with Cooperative Discriminators”. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 1638–1649. URL: <https://www.aclweb.org/anthology/P18-1152>.

- [132] Saar Hommes, Chris van der Lee, Felix Clouth, Jeroen Vermunt, Xander Verbeek, and Emiel Krahmer. “A Personalized Data-to-Text Support Tool for Cancer Patients”. In: *Proceedings of the 12th International Conference on Natural Language Generation*. Tokyo, Japan: Association for Computational Linguistics, Oct. 2019, pp. 443–452. DOI: 10.18653/v1/W19-8656. URL: <https://www.aclweb.org/anthology/W19-8656>.
- [133] Kristina Höök. “Affective loop experiences: designing for interactional embodiment”. In: *Philosophical Transactions of the Royal Society B: Biological Sciences* 364.1535 (2009), pp. 3585–3595.
- [134] Ian D Horswill. “Dear Leader’s Happy Story Time: A Party Game Based on Automated Story Generation”. In: *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*. 2016.
- [135] David Howell. *Contributions: parts of Magic that are all my fault*. Online. Retrieved September 14, 2021 from <https://web.archive.org/web/20190130162157/http://howell.seattle.wa.us/games/mtg/contributions.html>. Archived January 30, 2019.
- [136] Chenyang Huang, Osmar R Zaiane, Amine Trabelsi, and Nouha Dziri. “Automatic dialogue generation with expressed emotions”. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. 2018, pp. 49–54.
- [137] James Hunter, Yvonne Freer, Albert Gatt, Ehud Reiter, Somayajulu Sripada, and Cindy Sykes. “Automatic generation of natural language nursing shift summaries in neonatal intensive care: BT-Nurse”. In: *Artificial Intelligence in Medicine* 56.3 (2012), pp. 157–172. ISSN: 0933-3657. DOI: <https://doi.org/10.1016/j.artmed.2012.09.002>. URL: <https://www.sciencedirect.com/science/article/pii/S0933365712001170>.
- [138] Infocom. *Zork: The Great Underground Empire - Part I*. Game [PC]. Infocom, Cambridge, Massachusetts. Played 2017. Infocom, Dec. 1980.
- [139] Barry Ip. “Narrative Structures in Computer and Video Games: Part 1: Context, Definitions, and Initial Findings”. In: *Games and Culture* 6.2 (2011), pp. 103–134. DOI: 10.1177/1555412010364982. eprint: <http://doi.org/10.1177/1555412010364982>. URL: <https://doi.org/10.1177/1555412010364982>.
- [140] Iron Lore Entertainment. *Titan Quest*. Game [PC]. THQ, Agoura Hills, California, US. Agoura Hills, California, US, June 2006.
- [141] Rishabh Jain, Aaron Isaksen, Christoffer Holmgård, and Julian Togelius. “Autoencoders for level generation, repair, and recognition”. In: *Proceedings of the ICCG Workshop on Computational Creativity and Games*. 2016, p. 9.

- [142] Mark R. Johnson. “Towards qualitative procedural generation”. In: *Proceedings of Computational Creativity and Games Workshop 2016 (CCGW16)*. Paris, France, 2016.
- [143] Mark R. Johnson. *Ultima Ratio Regum*. Game [PC]. July 2012.
- [144] Annette Karmiloff-Smith. “Language and cognitive processes from a developmental perspective”. In: *Language and cognitive processes* 1.1 (1985), pp. 61–85.
- [145] Isaac Karth. “Preliminary poetics of procedural generation in games”. In: *Proc. Digital Games Research Association* (2018).
- [146] Manuel Kerssemakers, Jeppe Tuxen, Julian Togelius, and Georgios N Yannakakis. “A procedural procedural level generator generator”. In: *2012 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE. 2012, pp. 335–341.
- [147] Ahmed Khalifa, Philip Bontrager, Sam Earle, and Julian Togelius. “PC-GRL: Procedural content generation via reinforcement learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. Vol. 16. 1. 2020, pp. 95–101.
- [148] Chloé Kiddon, Luke Zettlemoyer, and Yejin Choi. “Globally coherent text generation with neural checklist models”. In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. 2016, pp. 329–339.
- [149] Klei Entertainment. *Oxygen Not Included*. Game [PC]. Klei Entertainment, Vancouver, British Columbia, CA. 2019.
- [150] Alexander Koller and Ronald P. A. Petrick. “Experiences with planning for natural language generation”. In: *Computational Intelligence* 27.1 (2011), pp. 23–40. DOI: <https://doi.org/10.1111/j.1467-8640.2010.00370.x>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8640.2010.00370.x>. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8640.2010.00370.x>.
- [151] Max Kreminski. *Epitaph*. Game [Online]. Game at itch.io. <https://mkremins.itch.io/epitaph>. 2016.
- [152] Karen Kukich. “Design of a knowledge-based report generator”. In: *21st Annual Meeting of the Association for Computational Linguistics*. 1983, pp. 145–150.
- [153] Quinn Kybartas and Rafael Bidarra. “A Survey on Story Generation Techniques for Authoring Computational Narratives”. In: *IEEE Transactions on Computational Intelligence and AI in Games* 9.3 (Sept. 2017), pp. 239–253. ISSN: 1943068X. DOI: 10.1109/TCIAIG.2016.2546063.

- [154] Quinn Kybartas, Rafael Bidarra, John-Jules Ch Meyer, et al. “Procedural generation of populations for storytelling”. In: *Proc. PCG 2015-Workshop on Procedural Content Generation for Games, co-located with the Tenth International Conference on the Foundations of Digital Games (IKBC15)*, 2015. 2016.
- [155] Quinn Kybartas and Clark Verbrugge. “Analysis of ReGEN as a graph-rewriting system for quest generation”. In: *IEEE Transactions on Computational Intelligence and AI in Games* 6.2 (2014), pp. 228–242.
- [156] Peter Landwehr, Jana Diesner, and Kathleen M. Carley. “The Words of Warcraft: relational text analysis of quests in an MMORPG”. In: *DiGRA '09 - Proceedings of the 2009 DiGRA International Conference: Breaking New Ground: Innovation in Games, Play, Practice and Theory*. Brunel University, Sept. 2009. URL: <http://www.digra.org/wp-content/uploads/digital-library/09287.34363.pdf>.
- [157] Alon Lavie and Abhaya Agarwal. “METEOR: An Automatic Metric for MT Evaluation with High Levels of Correlation with Human Judgments”. In: *Proceedings of the Second Workshop on Statistical Machine Translation*. Prague, Czech Republic: Association for Computational Linguistics, June 2007, pp. 228–231. URL: <https://www.aclweb.org/anthology/W07-0734>.
- [158] Michael Lebowitz. “Creating a Story-Telling Universe”. In: *Proceedings of the Eighth International Joint Conference on Artificial Intelligence - Volume 1*. IJCAI'83. Karlsruhe, West Germany: Morgan Kaufmann Publishers Inc., 1983, pp. 63–65.
- [159] Michael Lebowitz. “Planning stories”. In: *Proceedings of the 9th annual conference of the cognitive science society*. 1987, pp. 234–242.
- [160] Chris van der Lee, Emiel Krahmer, and Sander Wubben. “PASS: A Dutch data-to-text system for soccer, targeted towards specific audiences”. In: *Proceedings of the 10th International Conference on Natural Language Generation*. 2017, pp. 95–104.
- [161] Daniel Lee, Dayi Lin, Cor-Paul Bezemer, and Ahmed E Hassan. “Building the perfect game—an empirical study of game modifications”. In: *Empirical Software Engineering* (2020), pp. 1–34.
- [162] Scott Lee, Aaron Isaksen, Christoffer Holmgård, and Julian Togelius. “Predicting resource locations in game maps using deep convolutional neural networks”. In: *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*. 2016.

- [163] Leo Leppänen, Myriam Munezero, Stefanie Sirén-Heikel, Mark Granroth-Wilding, and Hannu Toivonen. “Finding and Expressing News from Structured Data”. In: *Proceedings of the 21st International Academic Mindtrek Conference*. AcademicMindtrek ’17. Tampere, Finland: Association for Computing Machinery, 2017, pp. 174–183. ISBN: 9781450354264. DOI: 10.1145/3131085.3131112. URL: <https://doi.org/10.1145/3131085.3131112>.
- [164] Jonathan Lessard, Etienne Brunelle-Leclerc, Timothy Gottschalk, Marc-Antoine Jetté-Léger, Odile Prouveur, and Christopher Tan. “Striving for Author-Friendly Procedural Dialogue Generation”. In: *Proceedings of the 12th International Conference on the Foundations of Digital Games*. FDG ’17. Hyannis, Massachusetts: Association for Computing Machinery, 2017. ISBN: 9781450353199. DOI: 10.1145/3102071.3116219. URL: <https://doi.org/10.1145/3102071.3116219>.
- [165] Boyang Li, Stephen Lee-Urban, George Johnston, and Mark O. Riedl. “Story generation with crowdsourced plot graphs”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 27. 1. 2013, pp. 598–604.
- [166] Boyang Li, Stephen Lee-Urban, and Mark O. Riedl. “Crowdsourcing interactive fiction games.” In: *FDG*. 2013, pp. 431–432.
- [167] Jiwei Li, Will Monroe, Alan Ritter, Dan Jurafsky, Michel Galley, and Jianfeng Gao. “Deep Reinforcement Learning for Dialogue Generation”. In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, Nov. 2016, pp. 1192–1202. DOI: 10.18653/v1/D16-1127. URL: <https://www.aclweb.org/anthology/D16-1127>.
- [168] Antonios Liapis and Georgios N. Yannakakis. “Boosting Computational Creativity with Human Interaction in Mixed-Initiative Co-Creation Tasks”. In: *Proceedings of the ICCG workshop on Computational Creativity and Games*. 2016.
- [169] Chin-Yew Lin. “ROUGE: A Package for Automatic Evaluation of Summaries”. In: *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, July 2004, pp. 74–81. URL: <https://www.aclweb.org/anthology/W04-1013>.
- [170] Zeming Lin, Jonas Gehring, Vasil Khalidov, and Gabriel Synnaeve. “Star-data: A StarCraft AI research dataset”. In: *Thirteenth Artificial Intelligence and Interactive Digital Entertainment Conference*. 2017.
- [171] Carl-Gustav Linden. “Decades of Automation in the Newsroom”. In: *Digital Journalism* 5.2 (2017), pp. 123–140. DOI: 10.1080/21670811.2016.1160791. eprint: <https://doi.org/10.1080/21670811.2016.1160791>. URL: <https://doi.org/10.1080/21670811.2016.1160791>.

- [172] Hugo Liu and Push Singh. “ConceptNet — a practical commonsense reasoning tool-kit”. In: *BT technology journal* 22.4 (2004), pp. 211–226.
- [173] Jialin Liu, Sam Snodgrass, Ahmed Khalifa, Sebastian Risi, Georgios N Yannakakis, and Julian Togelius. “Deep learning for procedural content generation”. In: *Neural Computing and Applications* 33.1 (2021), pp. 19–37.
- [174] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. “RoBERTa: A Robustly Optimized BERT Pretraining Approach”. In: *CoRR* abs/1907.11692 (2019). arXiv: 1907.11692. URL: <http://arxiv.org/abs/1907.11692>.
- [175] Ricardo Lopes and Rafael Bidarra. “Adaptivity challenges in games and simulations: a survey”. In: *IEEE Transactions on Computational Intelligence and AI in Games* 3.2 (2011), pp. 85–99.
- [176] Ludeon Studios. *RimWorld*. Game [PC]. Ludeon Studios, Ottawa, Ontario, CA. 2018.
- [177] Stephanie M. Lukin, James O. Ryan, and Marilyn A. Walker. “Automating direct speech variations in stories and games”. In: *Tenth Artificial Intelligence and Interactive Digital Entertainment Conference*. 2014.
- [178] MachineGames. *Wolfenstein: The New Order*. Game [PC]. Bethesda Softworks, Rockville, Maryland. 2014.
- [179] Yana Malysheva. “Dynamic quest generation in Micro Missions”. In: *2012 IEEE International Games Innovation Conference*. IEEE. 2012, pp. 1–3.
- [180] Chris Martens. “Ceptre: A language for modeling generative interactive systems”. In: *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*. 2015.
- [181] Colin Charles Mathews and Nia Wearn. “How are modern video games marketed?” In: *The Computer Games Journal* 5.1 (2016), pp. 23–37.
- [182] Maxis. *SimCity 2000*. Game [PC]. Maxis Software Inc./Electronic Arts. Maxis, 1996.
- [183] Maxis. *SimCity 3000*. Game [PC]. Maxis Software Inc./Electronic Arts. Maxis, 1999.
- [184] Jeffrey D McGovern and Gavin Scott. “EloquentRobot: A tool for automatic poetry generation”. In: *Proceedings of the Seventh ACM Conference on Bioinformatics, Computational Biology, and Health Informatics*. 2016.
- [185] James R Meehan. “TALE-SPIN, An Interactive Program that Writes Stories”. In: *IJCAI*. R. Schank (ed.). Inside Computer Understanding. Hillsdale: Lawrence Erlbaum, 197–225. 1977.

- [186] Nick Montfort. “Curveship: An Interactive Fiction System for Interactive Narrating”. In: *Proceedings of the Workshop on Computational Approaches to Linguistic Creativity*. Boulder, Colorado: Association for Computational Linguistics, June 2009, pp. 55–62. URL: <https://www.aclweb.org/anthology/W09-2008>.
- [187] Nick Montfort. “Generating narrative variation in interactive fiction”. PhD thesis. University of Pennsylvania, 2007.
- [188] Nick Montfort. *New Novel Machines: Nanowatt and World Clock*. Trope Tank Technical Report TROPE-13–03, July 2014. 2014.
- [189] Jane Morris and Graeme Hirst. “Lexical cohesion computed by thesaural relations as an indicator of the structure of text”. In: *Computational linguistics* 17.1 (1991), pp. 21–48.
- [190] Christopher Moser and Xiaowen Fang. “Narrative structure and player experience in role-playing games”. In: *International Journal of Human-Computer Interaction* 31.2 (2015), pp. 146–156.
- [191] Nasrin Mostafazadeh, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James Allen. “A Corpus and Cloze Evaluation for Deeper Understanding of Commonsense Stories”. In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California: Association for Computational Linguistics, June 2016, pp. 839–849. DOI: 10.18653/v1/N16-1098. URL: <https://www.aclweb.org/anthology/N16-1098>.
- [192] *MTG Wiki: David Howell*. Online. Retrieved September 14, 2021 from https://mtg.fandom.com/wiki/David_Howell.
- [193] Daniel Muriel and Garry Crawford. *Video games as culture: considering the role and importance of video games in contemporary society*. Routledge, 2018.
- [194] Courtney Napoles, Matthew Gormley, and Benjamin Van Durme. “Annotated GigaWord”. In: *Proceedings of the Joint Workshop on Automatic Knowledge Base Construction and Web-scale Knowledge Extraction*. Association for Computational Linguistics. 2012, pp. 95–100.
- [195] Liubov Nesterenko. “Building a System for Stock News Generation in Russian”. In: *Proceedings of the 2nd International Workshop on Natural Language Generation and the Semantic Web (WebNLG 2016)*. Edinburgh, Scotland: Association for Computational Linguistics, Sept. 2016, pp. 37–40. URL: <https://www.aclweb.org/anthology/W16-3507>.
- [196] Nintendo R&D1 and Intelligent Systems. *Super Metroid*. Game [SNES]. Nintendo, Kyoto, Japan. Kyoto, Japan, Apr. 1994.

- [197] Obsidian Entertainment. *Pillars of Eternity*. Game [PC]. Paradox Interactive, Stockholm, Sweden. Played December 2019. Stockholm, Sweden, Mar. 2015.
- [198] Hugo Gonalo Oliveira. “A survey on intelligent poetry generation: Languages, features, techniques, reutilisation and evaluation”. In: *Proceedings of the 10th International Conference on Natural Language Generation*. 2017, pp. 11–20.
- [199] Curtis Onuczko, Duane Szafron, Jonathan Schaeffer, Maria Cutumisu, Jeff Siegel, Kevin Waugh, and Allan Schumacher. “A Demonstration of SQUEGE: A CRPG Sub-Quest Generator”. In: *Proceedings of the Third AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. AIIDE’07. Stanford, California: AAAI Press, 2007, pp. 110–111.
- [200] Jeff Orkin and Deb Roy. “The restaurant game: Learning social behavior and language from thousands of players online”. In: *Journal of Game Development* 3.1 (2007), pp. 39–60.
- [201] Pedro Javier Ortiz Su  rez, Beno  t Sagot, and Laurent Romary. “Asynchronous pipelines for processing huge corpora on medium to low resource infrastructures”. en. In: ed. by Piotr Ba  ski, Adrien Barbaresi, Hanno Biber, Evelyn Breiteneder, Simon Clematide, Marc Kupietz, Harald L  ngen, and Caroline Iliadi. *Proceedings of the Workshop on Challenges in the Management of Large Corpora (CMLC-7) 2019*. Cardiff, 22nd July 2019. Mannheim: Leibniz-Institut f  r Deutsche Sprache, 2019, pp. 9–16. DOI: 10.14618/ids-pub-9021. URL: <http://nbn-resolving.de/urn:nbn:de:bsz:mh39-90215>.
- [202] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. “Bleu: a Method for Automatic Evaluation of Machine Translation”. In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, July 2002, pp. 311–318. DOI: 10.3115/1073083.1073135. URL: <https://www.aclweb.org/anthology/P02-1040>.
- [203] Ronald Petrick and Mary Ellen Foster. “Planning for social interaction in a robot bartender domain”. In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 23. 1. 2013.
- [204] Kelsey Piper. *An AI helped us write this article. An impressive new language AI writes product reviews and news articles. Its creators are worried about misuse*. Vox, Future Perfect. Published February 14, 2019. Retrieved August 11, 2021 from <https://www.vox.com/future-perfect/2019/2/14/18222270/artificial-intelligence-open-ai-natural-language-processing>. Feb. 2019.

- [205] David Plans and Davide Morelli. “Experience-driven procedural music generation for games”. In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.3 (2012), pp. 192–198.
- [206] Yevgeniy Puzikov and Iryna Gurevych. “E2E NLG Challenge: Neural Models vs. Templates”. In: *Proceedings of the 11th International Conference on Natural Language Generation*. Tilburg University, The Netherlands: Association for Computational Linguistics, Nov. 2018, pp. 463–471. DOI: 10.18653/v1/W18-6557. URL: <https://www.aclweb.org/anthology/W18-6557>.
- [207] Katyanna Quach. *This news article about the full public release of OpenAI’s ‘dangerous’ GPT-2 model was part written by GPT-2*. The Register, Software, AI + ML. Published November 6, 2019. Retrieved August 11, 2021 from https://www.theregister.com/2019/11/06/openai_gpt2_released/. Nov. 2019.
- [208] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. “Improving language understanding by generative pre-training”. In: (2018). https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf. Retrieved April 8, 2021.
- [209] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. “Language models are unsupervised multitask learners”. In: (2019). <https://github.com/openai/gpt-2>. Retrieved August 31, 2020.
- [210] Aaron A Reed. “Sharing authoring with algorithms: Procedural generation of satellite sentences in text-based interactive stories”. In: *Proceedings of The third workshop on Procedural Content Generation in Games*. ACM. Raleigh, North Carolina, 2012.
- [211] Ehud Reiter. “An architecture for data-to-text systems”. In: *Proceedings of the Eleventh European Workshop on Natural Language Generation*. Association for Computational Linguistics. 2007, pp. 97–104.
- [212] Ehud Reiter. “NLG vs. templates”. In: *Proceedings of the Fifth European Workshop on Natural Language Generation*. Leiden, the Netherlands, 1995, pp. 95–105.
- [213] Ehud Reiter and Robert Dale. “Building applied natural language generation systems”. In: *Natural Language Engineering* 3.1 (1997), pp. 57–87.
- [214] Ehud Reiter, Roma Robertson, and Liesl M Osman. “Lessons from a failure: Generating tailored smoking cessation letters”. In: *Artificial Intelligence* 144.1-2 (2003), pp. 41–58.
- [215] Mark O Riedl and R. Michael Young. “Narrative planning: Balancing plot and character”. In: *Journal of Artificial Intelligence Research* 39 (2010), pp. 217–268.

- [216] Mark O. Riedl and R. Michael Young. “An intent-driven planner for multi-agent story generation”. In: *Autonomous Agents and Multiagent Systems, International Joint Conference on*. Vol. 2. IEEE Computer Society. 2004, pp. 186–193.
- [217] Alan Ritter, Colin Cherry, and William B. Dolan. “Data-Driven Response Generation in Social Media”. In: *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*. Edinburgh, Scotland, UK.: Association for Computational Linguistics, July 2011, pp. 583–593. URL: <https://www.aclweb.org/anthology/D11-1054>.
- [218] Justus Robertson and R Michael Young. “Automated gameplay generation from declarative world representations”. In: *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*. 2015.
- [219] Rockstar Studios. *Red Dead Redemption 2*. Game [PC]. Rockstar Games, New York City, New York, United States. Played February 2021. New York City, New York, United States, Nov. 2019.
- [220] Anna Rohrbach, Lisa Anne Hendricks, Kaylee Burns, Trevor Darrell, and Kate Saenko. “Object Hallucination in Image Captioning”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, Oct. 2018, pp. 4035–4045. DOI: 10.18653/v1/D18-1437. URL: <https://aclanthology.org/D18-1437>.
- [221] Samuel Rönqvist, Jenna Kanerva, Tapio Salakoski, and Filip Ginter. “Is Multilingual BERT Fluent in Language Generation?” In: *Proceedings of the First NLPL Workshop on Deep Learning for Natural Language Processing*. Turku, Finland: Linköping University Electronic Press, Sept. 2019, pp. 29–36. URL: <https://www.aclweb.org/anthology/W19-6204>.
- [222] Runic Games. *Torchlight II*. Game [PC]. Runic Games, Seattle, Washington, US. Runic Games, 2012.
- [223] James Ryan, Ethan Seither, Michael Mateas, and Noah Wardrip-Fruin. “Expressionist: An authoring tool for in-game text generation”. In: *International Conference on Interactive Digital Storytelling*. Springer. 2016, pp. 221–233.
- [224] James O. Ryan, Tyler Brothers, Michael Mateas, and Noah Wardrip-Fruin. “Juke joint: characters who are moved by music”. In: *Proc. Experimental AI in Games* (2016).
- [225] James O. Ryan, Michael Mateas, and Noah Wardrip-Fruin. “Characters who speak their minds: Dialogue generation in Talk of the Town”. In: *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*. 2016.

- [226] James O. Ryan, Michael Mateas, and Noah Wardrip-Fruin. "Open design challenges for interactive emergent narrative". In: *International Conference on Interactive Digital Storytelling*. Springer. 2015, pp. 14–26.
- [227] James O. Ryan, Adam Summerville, Michael Mateas, and Noah Wardrip-Fruin. "Toward characters who observe, tell, misremember, and lie". In: *Proc. Experimental AI in Games 2* (2015).
- [228] James O. Ryan, Adam J. Summerville, and Ben Samuel. "Bad News: A game of death and communication". In: *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. 2016, pp. 160–163.
- [229] James Owen Ryan, Casey Barackman, Nicholas Kontje, Taylor Owen-Milner, Marilyn A. Walker, Michael Mateas, and Noah Wardrip-Fruin. "Combinatorial dialogue authoring". In: *International Conference on Interactive Digital Storytelling*. Springer. 2014, pp. 13–24.
- [230] Christoph Salge, Michael Cerny Green, Rodrigo Canaan, Filip Skwarski, Rafael Fritsch, Adrian Brightmoore, Shaofang Ye, Changxing Cao, and Julian Togelius. "The AI settlement generation challenge in minecraft". In: *KI-Künstliche Intelligenz 34.1* (2020), pp. 19–31.
- [231] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter". In: *CoRR abs/1910.01108* (2019). arXiv: 1910.01108. URL: <http://arxiv.org/abs/1910.01108>.
- [232] Laura Scheuter. "Does it make sense? Analyzing coherence in longer fictional discourse on a syntactic and semantic level". Master thesis. 2021.
- [233] Björn Schlünder and Ralf Klabunde. "Greetings Generation in Video Role Playing Games". In: *Proceedings of the 14th European Workshop on Natural Language Generation*. 2013, pp. 167–171.
- [234] Anne Schneider, Alasdair Mort, Chris Mellish, Ehud Reiter, Phil Wilson, and Pierre-Luc Vaudry. "MIME- NLG Support for Complex and Unstable Pre-hospital Emergencies". In: *Proceedings of the 14th European Workshop on Natural Language Generation*. Sofia, Bulgaria: Association for Computational Linguistics, Aug. 2013, pp. 198–199. URL: <https://www.aclweb.org/anthology/W13-2128>.
- [235] Jason Schreier. *Blood, sweat, and pixels: The triumphant, turbulent stories behind how video games are made*. Harper, 2017.
- [236] Lauren R. Shapiro and Judith A. Hudson. "Tell me a make-believe story: Coherence and cohesion in young children's picture-elicited narratives". In: *Developmental Psychology* 27.6 (1991), pp. 960–974.

- [237] Shi-Qi Shen, Yan-Kai Lin, Cun-Chao Tu, Yu Zhao, Zhi-Yuan Liu, Mao-Song Sun, et al. “Recent advances on neural headline generation”. In: *Journal of Computer Science and Technology* 32.4 (2017), pp. 768–784.
- [238] Emily Sheng, Kai-Wei Chang, Premkumar Natarajan, and Nanyun Peng. “The Woman Worked as a Babysitter: On Biases in Language Generation”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 3407–3412. DOI: 10.18653/v1/D19-1339. URL: <https://www.aclweb.org/anthology/D19-1339>.
- [239] Igor Shevchenko. *NaNoGenMo: как компьютеры пишут новеллы [NaNoGenMo: how computers write novels]*. Online. Retrieved February 16, 2022 from <https://habr.com/en/post/313862/>. 2016.
- [240] Tanya X Short and Tarn Adams, eds. *Procedural Generation in Game Design*. AK Peters/CRC Press, 2017.
- [241] Tanya X Short and Tarn Adams. *Procedural Storytelling in Game Design*. CRC Press, 2019.
- [242] Vered Shwartz, Rachel Rudinger, and Oyvind Tafjord. ““You are grounded!”: Latent Name Artifacts in Pre-trained Language Models”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, Nov. 2020, pp. 6850–6861. DOI: 10.18653/v1/2020.emnlp-main.556. URL: <https://aclanthology.org/2020.emnlp-main.556>.
- [243] Adam M Smith and Michael Mateas. “Answer set programming for procedural content generation: A design space approach”. In: *IEEE Transactions on Computational Intelligence and AI in Games* 3.3 (2011), pp. 187–200.
- [244] Irene Solaiman, Miles Brundage, Jack Clark, Amanda Askell, Ariel Herbert-Voss, Jeff Wu, Alec Radford, and Jasmine Wang. “Release Strategies and the Social Impacts of Language Models”. In: *CoRR abs/1908.09203* (2019). arXiv: 1908.09203. URL: <http://arxiv.org/abs/1908.09203>.
- [245] Karen Spärck Jones and Julia R. Galliers. *Evaluating natural language processing systems: An analysis and review*. Vol. 1083. Lecture Notes in Computer Science. Springer Science & Business Media, 1996.
- [246] Robyn Speer, Joshua Chin, and Catherine Havasi. “ConceptNet 5.5: An Open Multilingual Graph of General Knowledge”. In: 2017, pp. 4444–4451. URL: <http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14972>.

- [247] Sharon L. Spencer and Jill Fitzgerald. “Validity and structure, coherence, and quality measures in writing”. In: *Journal of Reading Behavior* 25.2 (1993), pp. 209–231.
- [248] Squad. *Kerbal Space Program*. Game [PC]. Private Division, New York City, New York, US. 2015.
- [249] Judith van Stegeren and Jakub Myśliwiec. “Fine-tuning GPT-2 on annotated RPG quests for NPC dialogue generation”. In: *Proceedings of the 16th International Conference on the Foundations of Digital Games (FDG) 2021*. FDG ’21. Online: ACM, Aug. 2021.
- [250] Judith van Stegeren and Mariët Theune. “Churnalist: Fictional Headline Generation for Context-appropriate Flavor Text”. In: *Proceedings of the Tenth International Conference on Computational Creativity*. ICC3 ’19. Charlotte, NC, USA: Association for Computational Creativity (ACC), 2019, pp. 65–72.
- [251] Judith van Stegeren and Mariët Theune. “Fantastic Strings and Where to Find Them: The Quest for High-Quality Video Game Text Corpora”. In: *Proceedings of the 2020 Workshop on Intelligent Narrative Technologies*. Oct. 2020.
- [252] Judith van Stegeren and Mariët Theune. “Narrative Generation in the Wild: Methods from NaNoGenMo”. In: *Proceedings of the Second Workshop on Storytelling*. Florence, Italy: Association for Computational Linguistics, Aug. 2019, pp. 65–74. URL: <https://www.aclweb.org/anthology/W19-3407>.
- [253] Judith van Stegeren and Mariët Theune. “Remixing Headlines for Context-Appropriate Flavor Text”. In: *Proceedings of the IEEE Conference On Games*. London, UK: IEEE, Aug. 2019.
- [254] Judith van Stegeren and Mariët Theune. “Towards Generating Textual Game Assets from Real-world Data”. In: *Proceedings of the 13th International Conference on the Foundations of Digital Games*. FDG ’18. Malmö, Sweden: ACM, 2018, 43:1–43:4. ISBN: 978-1-4503-6571-0. DOI: 10.1145/3235765.3235809. URL: <http://doi.acm.org/10.1145/3235765.3235809>.
- [255] Andrew Stockdale. “ClueGen: An exploration of procedural storytelling in the format of murder mystery games”. In: *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*. 2016.
- [256] Christina R. Strong, Manish Mehta, Kinshuk Mishra, Alistair Jones, and Ashwin Ram. “Emotionally Driven Natural Language Generation for Personality Rich Characters in Interactive Games.” In: *Proceedings of the Third Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*. 2007, pp. 98–100.

- [257] Anne Sullivan, Mirjam Palosaari Eladhari, and Michael Cook. “Tarot-based narrative generation”. In: *Proceedings of the 13th International Conference on the Foundations of Digital Games*. ACM. 2018, p. 54.
- [258] A. Summerville, S. Snodgrass, M. Guzdial, C. Holmgård, A. K. Hoover, A. Isaksen, A. Nealen, and J. Togelius. “Procedural Content Generation via Machine Learning (PCGML)”. In: *IEEE Transactions on Games* 10.3 (2018), pp. 257–270.
- [259] Adam Summerville, Sam Snodgrass, Michael Mateas, and Santiago Ontanón. “The VGLC: The video game level corpus”. In: *Proceedings of the 7th Workshop on Procedural Content Generation*. Aug. 2016.
- [260] Adam J. Summerville and Michael Mateas. “Super Mario as a String: Platformer Level Generation Via LSTMs”. In: *DiGRA/FDG ’16 - Proceedings of the First International Joint Conference of DiGRA and FDG*. 2016.
- [261] Petra Sundström. *Exploring the Affective Loop*. Licentiate thesis. 2005.
- [262] Ivo Swartjes and Mariët Theune. “The virtual storyteller: Story generation by simulation”. In: *BNAIC*. 2008, pp. 257–264.
- [263] Alan Tapscott, Carlos León, and Pablo Gervás. “Generating Stories Using Role-playing Games and Simulated Human-like Conversations”. In: *Proceedings of the 3rd Workshop on Computational Creativity in Natural Language Generation (CC-NLG 2018)*. 2018, pp. 34–42.
- [264] *The Sims*. Game [PC].
- [265] Mariët Theune, Esther Klabbers, Jan-Roelof de Pijper, Emiel Krahmer, and Jan Odijk. “From data to speech: a general approach”. In: *Natural Language Engineering* 7.1 (2001), pp. 47–86.
- [266] Mariët Theune, Nanda Slabbers, and Feikje Hielkema. “The Narrator: NLG for digital storytelling”. In: *Proceedings of the Eleventh European Workshop on Natural Language Generation*. Saarbrücken, Germany, June 2007, pp. 109–112. URL: <https://www.aclweb.org/anthology/W07-2317>.
- [267] Julien “Orteil” Thiennot. *Cookie Clicker*. Game [PC/Browser]. Aug. 2013.
- [268] Julian Togelius, Emil Kastbjerg, David Schedl, and Georgios N. Yannakakis. “What is Procedural Content Generation? Mario on the Borderline”. In: *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*. PCGames ’11. Bordeaux, France: Association for Computing Machinery, 2011. ISBN: 9781450308724. DOI: 10.1145/2000919.2000922. URL: <https://doi.org/10.1145/2000919.2000922>.
- [269] Julian Togelius, Mike Preuss, Nicola Beume, Simon Wessing, Johan Hagelbäck, and Georgios N Yannakakis. “Multiobjective exploration of the starcraft map space”. In: *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*. IEEE. 2010, pp. 265–272.

- [270] Julian Togelius and Jurgen Schmidhuber. “An experiment in automatic game design”. In: *2008 IEEE Symposium On Computational Intelligence and Games*. IEEE. 2008, pp. 111–118.
- [271] Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, and Cameron Browne. “Search-based procedural content generation: A taxonomy and survey”. In: *IEEE Transactions on Computational Intelligence and AI in Games* 3.3 (2011), pp. 172–186.
- [272] Rijnard van Tonder, Asher Trockman, and Claire Le Goues. “A panel data set of cryptocurrency development activity on GitHub”. In: *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE. 2019, pp. 186–190.
- [273] Peter Unold, Claus Leth Gregersen, Allan Ove Kjeldbjerg, and The Freeciv project. *Freeciv*. Game [PC]. <https://github.com/freeciv/>. 1996.
- [274] Alberto Uriarte and Santiago Ontañón. “PSMAGE: Balanced map generation for StarCraft”. In: *2013 IEEE Conference on Computational Intelligence in Games (CIG)*. 2013, pp. 1–8. DOI: 10.1109/CIG.2013.6633644.
- [275] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., 2017. URL: <https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf>.
- [276] Tony Veale. “The shape of tweets to come: Automating language play in social networks”. In: *Multiple Perspectives on Language Play*. Language Play and Creativity 1 (2016), pp. 73–92.
- [277] Ramakrishna Vedantam, C. Lawrence Zitnick, and Devi Parikh. “CIDEr: Consensus-based image description evaluation”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 4566–4575. DOI: 10.1109/CVPR.2015.7299087.
- [278] Wietse de Vries, Andreas van Cranenburgh, Arianna Bisazza, Tommaso Caselli, Gertjan van Noord, and Malvina Nissim. *BERTje: A Dutch BERT Model*. arXiv:1912.09582. Dec. 2019. URL: <http://arxiv.org/abs/1912.09582>.
- [279] Wietse de Vries and Malvina Nissim. “As good as new. How to successfully recycle English GPT-2 to make models for other languages”. In: *CoRR* abs/2012.05628 (2020). arXiv: 2012.05628. URL: <https://arxiv.org/abs/2012.05628>.

- [280] Marilyn A. Walker, Ricky Grant, Jennifer Sawyer, Grace I. Lin, Noah Wardrip-Fruin, and Michael Buell. “Perceived or not perceived: Film character models for expressive NLG”. In: *International Conference on Interactive Digital Storytelling*. Springer. 2011, pp. 109–121.
- [281] Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. “Universal Adversarial Triggers for Attacking and Analyzing NLP”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 2153–2162. DOI: 10.18653/v1/D19-1221. URL: <https://aclanthology.org/D19-1221>.
- [282] Nick Walton. *AI Dungeon*. Game [PC, Android, IOS]. <https://www.aidungeon.io>. Latitude Games, 2019.
- [283] Alex Wang and Kyunghyun Cho. “BERT has a Mouth, and It Must Speak: BERT as a Markov Random Field Language Model”. In: *Proceedings of the Workshop on Methods for Optimizing and Evaluating Neural Language Generation*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 30–36. DOI: 10.18653/v1/W19-2304. URL: <https://www.aclweb.org/anthology/W19-2304>.
- [284] Noah Wardrip-Fruin. *Expressive Processing: Digital Fictions, Computer Games, and Software Studies*. The MIT Press, 2009. ISBN: 0262013436.
- [285] Michael Witwer, Kyle Newman, Jon Peterson, and Sam Witwer. *Dungeons & Dragons Art & Arcana: A Visual History*. Ten Speed Press, 2018.
- [286] Wizards of the Coast. *Magic: The Gathering: How to play*. Online. Retrieved September 16, 2021 from <https://magic.wizards.com/en/magic-gameplay>.
- [287] Yager Development. *Spec Ops: The Line*. Game [PC, Linux]. 2K Games, Novato, California, US. 2012.
- [288] Georgios N. Yannakakis and Julian Togelius. “Experience-driven procedural content generation”. In: *IEEE Transactions on Affective Computing* 2.3 (2011), pp. 147–161.
- [289] Lili Yao, Nanyun Peng, Weischedel Ralph, Kevin Knight, Dongyan Zhao, and Rui Yan. “Plan-And-Write: Towards Better Automatic Storytelling”. In: *Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)*. 2019.
- [290] Rowan Zellers, Ari Holtzman, Hannah Rashkin, Yonatan Bisk, Ali Farhadi, Franziska Roesner, and Yejin Choi. “Defending Against Neural Fake News”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 9054–9065. URL: <http://papers.nips.cc/paper/9106-defending-against-neural-fake-news.pdf>.

- [291] ZeniMax Online Studios. *The Elder Scrolls Online*. Game [PC]. Bethesda Softworks, Rockville, Maryland, US. Bethesda Softworks, 2014.
- [292] Jichen Zhu, Antonios Liapis, Sebastian Risi, Rafael Bidarra, and G Michael Youngblood. “Explainable AI for designers: A human-centered perspective on mixed-initiative co-creation”. In: *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE. 2018, pp. 1–8.
- [293] Yaoming Zhu, Sidi Lu, Lei Zheng, Jiaxian Guo, Weinan Zhang, Jun Wang, and Yong Yu. “Texygen: A Benchmarking Platform for Text Generation Models”. In: *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. SIGIR ’18. Ann Arbor, MI, USA: Association for Computing Machinery, 2018, pp. 1097–1100. ISBN: 9781450356572. DOI: 10.1145/3209978.3210080. URL: <https://doi.org/10.1145/3209978.3210080>.

Summary

Role-playing video games allow us to immerse ourselves in virtual worlds and take on the role of virtual characters. To help the player with immersion, these games often contain “flavor text”: colourful decorative texts, such as item descriptions or dialogues with non-player characters, that help bring the game world and its narrative to life. Flavor text supports the game’s narrative. It provides context for the events happening in the game, and justification for the player’s actions.

As an example of flavor text, let’s look at action game *Diablo III* (Blizzard, 2012). In *Diablo III*, players can find gold and treasure as a reward for slaying monsters and completing quests. One of these treasures is a legendary ring called “Leoric’s signet.” This rare item has its own flavor text, which hints at the corrupted original owner of the jewel, a recurring antagonist in the *Diablo* series:

King Leoric had hoped to pass along the symbol of his family’s noble lineage to his first son before they met an unfortunate fate.

This example illustrates the power of flavor text: this short piece of text tells us a bit of the backstory of an important character, thus giving more depth to *Diablo*’s game world.

Video games are getting more and more extensive, which means that they also need more and more content, including flavor text. Normally, flavor text is written by specialized writers. This dissertation investigates how we can automatically create flavor text using natural language generation, or NLG. Language generation is part of natural language processing (NLP), the research field that combines computer science and linguistics. Generating flavor text is also part of procedural content generation (PCG), the field that examines how we can automatically generate (game) content, such as music, levels, textures, images, or even complete games. Relevant developments from these two fields are discussed in Chapter 2 and 3.

Most research on NLG is concerned with the creation of non-fiction text, such as text for information systems and expert systems, rather than fiction, such as stories, poetry, and video game texts. Good fictional texts have very different properties than good non-fiction texts, which means generating fiction requires

a different approach than generating non-fiction. Therefore, in Chapter 4 of this dissertation, we look at story generation to gain inspiration for flavor text generation. In Chapters 5, 6 and 7 we examine different text generators for flavor text. The flavor text in this thesis was created using (1) a formal rule-based text generator and (2) multiple text generators that build on a deep learning language model called GPT-2. GPT-2 is particularly good at generating coherent, non-factual texts, so it is well suited for generating fiction.

GPT-2 can learn new tasks based on examples. In this dissertation, the task is to generate flavor text with a particular form (headlines) or content (flavor text about space ships for a science fiction game). Teaching GPT-2 this task requires a large amount of video game texts that can serve as examples. Video game text datasets are still very scarce at the moment. That is why I have developed several new datasets with video game texts from existing roleplaying video games. These datasets, or corpora, are discussed in Chapter 8.

Because game text, including flavor text, is often very context sensitive, I also investigated how we can communicate this in-game context to the text generator. The generator can then integrate the in-game context into generated texts. I also looked for ways to play with the reader's expectations. If a text generator creates a text that is random, boring, or contains linguistic errors, can we make the player believe that this was precisely the intent of the generator?

Samenvatting

In role-playing video games, oftewel video games met een rollenspel-component, kunnen we de rol aannemen van virtuele personages en ons onderdompelen in een virtuele wereld. Deze games bevatten vaak “flavor text”: decoratieve teksten, zoals kleurrijke beschrijvingen en dialogen met andere personages, die de gamewereld tot leven brengen en zo de speler helpen met het opgaan in het spel. Daarnaast ondersteunt flavor text het overkoepelende verhaal van een video game: het biedt context voor de gebeurtenissen in het spel en rechtvaardiging voor de acties van de speler.

Een concreet voorbeeld van flavor text vinden we in actiegame *Diablo III* (Blizzard, 2012). In *Diablo III* kunnen spelers goudstukken en schatten vinden als beloning voor het verslaan van monsters en het voltooien van opdrachten. Een van deze schatten is een legendarische ring genaamd “Leoric’s signet” (Leoric’s zegelring). Dit zeldzame juweel heeft zijn eigen flavor text, die verwijst naar de ontaalde oorspronkelijke eigenaar, een terugkerende slechterik in de *Diablo* videogame-serie:

Koning Leoric had gehoopt dit symbool van de adellijke afkomst van zijn familie door te kunnen geven aan zijn oudste zoon, voordat zij voortijdig tot een ongelukkig einde kwamen.

Dit voorbeeld illustreert de kracht van flavor text: dit korte stukje tekst vertelt ons iets over de achtergrond van een belangrijk personage, en geeft zo meer diepgang aan *Diablo*’s gamewereld.

Video games worden steeds uitgebreider, wat betekent dat ze ook steeds meer inhoud nodig hebben, waaronder ook flavor text. Normaal gesproken wordt flavor text geschreven door gespecialiseerde schrijvers. Dit proefschrift onderzoekt hoe we flavor text automatisch kunnen creëren met behulp van natuurlijke taalgeneratie (natural language generation, of NLG). Taalgeneratie is onderdeel van natural language processing (NLP), het onderzoeksgebied dat informatica en taalkunde combineert. Het genereren van flavor text is ook onderdeel van procedural content generation (PCG), het vakgebied dat onderzoekt hoe we automatisch (game) content kunnen genereren, zoals muziek, levels, textures, afbeeldingen of zelfs complete games. Relevante ontwikkelingen uit deze twee vakgebieden worden besproken in Hoofdstuk 2 en 3.

Het meeste onderzoek naar NLG houdt zich bezig met het creëren van non-fictietekst, zoals tekst voor informatiesystemen en expert-systemen, in plaats van fictie, zoals verhalen, poëzie en videogameteksten. Goede fictieve teksten hebben hele andere eigenschappen dan goede non-fictie teksten, en het genereren van fictie vereist dan ook een andere aanpak dan het automatisch creëren van non-fictie. In Hoofdstuk 4 van deze dissertatie bekijken we daarom story generation (tekstgeneratie voor verhalen) om inspiratie op te doen voor flavor text generation. In Hoofdstuk 5, 6 en 7 bekijken we verschillende tekstgeneratoren voor flavor text. De flavor text in dit proefschrift is gemaakt met (1) een op formele regels gebaseerde tekstgenerator en (2) meerdere tekstgeneratoren die voortbouwen op een taalmodel gebaseerd op deep learning, genaamd GPT-2. GPT-2 is bijzonder goed in het genereren van coherente, niet-feitelijke teksten, en dus heel geschikt voor het genereren van fictie.

GPT-2 kan nieuwe taken leren op basis van voorbeelden. In dit proefschrift is die taak flavor text genereren met een bepaalde vorm (bijvoorbeeld flavor text in de vorm van krantenkoppen) of inhoud (bijvoorbeeld flavor text voor een science fiction game die gaat over ruimteschepen). Het aanleren van zo'n nieuwe taak vereist een grote hoeveelheid videogameteksten die als voorbeeld kunnen dienen. Datasets met videogameteksten zijn op dit moment nog erg schaars. Daarom heb ik meerdere nieuwe datasets ontwikkeld met videogameteksten van bestaande roleplaying video games. Deze datasets, of corpora, worden besproken in Hoofdstuk 8.

Omdat gametekst, inclusief flavor text, vaak erg contextgevoelig is, heb ik tevens onderzocht hoe we deze in-game context kunnen doorgeven aan de tekstgenerator. De generator kan dan de in-game context integreren in gegenereerde teksten. Tenslotte heb ik ook gezocht naar manieren om met de verwachtingen van de lezer te spelen. Stel je voor dat een tekstgenerator een tekst maakt die heel willekeurig is, heel saai is, of taalkundige fouten bevat. Kunnen we de speler dan laten geloven dat dit juist precies de bedoeling van de generator was?

Acknowledgements

The most valuable lessons I learned in the past four years stem from working with others. These lessons were not about text generation, but about building friendships, writing, coaching and collaborating.

Many people contributed to this thesis and my development as a researcher. Firstly, I would like to thank Mariët, my daily supervisor, and Dirk, my promotor. Mariët has a talent for bringing out the best in people. She was always available for providing feedback on my work, answering questions and giving encouragement. She was also the driving factor for any progress in my writing skills in the past few years. Mariët frequently asked me the two most important writing questions: “What did you want to say with this particular section?” and “And is that what you actually wrote down?” Dirk made me see that any two topics can be combined into a (sometimes new) scientific area: technology and food, VR and healthcare, coaching and robots. His philosophical questions helped me to keep a broad perspective on my research. I am grateful for the creative freedom that I received from my supervisors. Without it, my PhD progression might have been more straightforward, but also much less interesting.

Many thanks to the members of my thesis committee, Franciska de Jong, Gerben Westerhof, Pieter Spronck, Ehud Reiter, and Mike Cook, who tackled this pile of words with interest and enthusiasm. Thank you for your useful comments and encouraging words in conference hallway tracks and online gatherings.

I am grateful to the students that I supervised over the years. I thank Thérèse Bergsma, Jakub Myśliwiec and Laura Scheuter in particular, as their research projects contributed to the work in this dissertation. Thérèse and Jakub eventually turned into co-authors, which is an achievement to be proud of.

I thank my fellow Data2Game members from the University of Twente, Tilburg University, Veiligheidsregio Twente and HBA Lab for the enjoyable collaboration. I am also grateful to the Dutch Research Council (NWO), which funded the Data2Game research project. Johannes and Paris, thank you for the monthly rides to Hengelo and the enjoyable pre- and post-meeting conversations. I thank Thomas for the technical support.

I am grateful to my Human Media Interaction colleagues for sharing their enthusiasm and experience. I have never encountered a research department that was more creative and multidisciplinary than HMI. Every HMI member has at

some point or other contributed significantly to this thesis, in the form of friendly chats, companionship during coffee breaks, advice about supervising students, teaching and writing, suggesting scientific methodology, and recommending interesting papers. I would like to specifically mention Michel, for our bi-weekly machine learning and career development sessions, Daniël for converting me to decaf coffee, Merijn, whose sage advice helped me give shape to the first research paper I was really happy with, and Deniece, for all the good conversations about Life, the Universe and Everything. Jelte and Lorenzo, thank you for allowing me to adopt the couch in your office as my favorite place for thinking (strategically positioned right next to the coffee machine). You have also been tremendously helpful as my paranymphs during the final stages of my PhD. A special shoutout to Charlotte and Alice, which are the behind-the-scenes forces that keep HMI going amid the academic chaos.

I thank the members of ICIS (now DIS) at Radboud University Nijmegen for their inspiring teaching, and their hospitality in the Mercator building. I am especially grateful to Henk Barendregt, who taught me both vipassana meditation and mathematics. I remember one specific lecture, when he told our class that to become a good mathematician, you need to learn to be comfortable with having a million wrong insights. Until you have a single *right* insight... and you've suddenly found a new theorem! I would also like to thank Herman Geuvers, Bart Jacobs, Alexandra Silva, and Frits Vaandrager, who gave me the first taste of what it means to do research.

The work in this thesis builds on software and data that was generously created and shared by video game modding communities, open source communities and open data communities. I thank the NaNoGenMo community, as their contributions led to my first published paper and the inspiration for my first flavor text generator. The Twitter communities around procedural generation, NLP, and video games research pointed me to various useful resources, video games, conferences, research papers, blogposts and talks. I am also grateful to the contributors of NexusMods and the Xoreos project. Without you, the dataset creation from Chapter 8 would not have been possible. A shout-out to Michiel Werring, who occasionally acted as my informant from the world of AAA game development.

I am profoundly grateful for my #RU friends. I feel incredibly lucky to be part of such a large and diverse group of friends. Without you, I would never have finished this dissertation. There are a few subsets that deserve a special mention. Academic fellows from #promovendi, thank you for preparing me for every aspect of PhD life. You kept me motivated and sane, and through you, I learned to survive in academia. The usual suspects of our bouldering group, book club, rock band, and film club provided welcome distraction, entertainment and good conversations during the challenges of COVID-isolation. Besides companionship in Overwatch, Valheim and Sea of Thieves, the people from #rugaming provided me with many of the examples of flavor text and game narrative that can be

found in this thesis. May the mead flow steadily in your Valheim halls!

Annelies, Hanne, Irene, Linda, Margot, Margot, Marlie, Marrit, Mara, I wish every person a group of inspiring examples like you.

I thank Anton and Francis from Tacos On Tour for their amazing barbecued food, war stories, and hilarious postcards. I would like to thank Tim (and kantoorkat Sam) for much-needed peer-support, great lunches and a lovely writing environment during the final stages of this thesis.

Finally, I want to thank my family, Aad, Jacqueline, Marjo, Frank, Thijs, Linda, and Gerdriaan, for their love and continuous support.

Judith van Stegeren
Nijmegen, February 2022

About the author

Judith van Stegeren was born on 29 September 1990, in Utrecht, The Netherlands. She received both a BSc degree (2013) and MSc degree (2015) in Computer Science from Radboud University Nijmegen. From 2018–2021, she pursued a PhD degree at the University of Twente in Enschede. Prior to joining the Human Media Interaction research group, she worked as a digital security specialist at the Dutch National Cyber Security Centre in the Hague. Her research interests include natural language processing, procedural art, and data engineering. A more complete overview of Judith’s curriculum vitae can be found on her personal webpage, <https://www.judithvanstegeren.com>.

List of publications

Below is a list of peer-reviewed academic publications that Judith has authored or co-authored, in reverse chronological order:

1. Judith van Stegeren and Jakub Myśliwiec. “Fine-tuning GPT-2 on annotated RPG quests for NPC dialogue generation”. In: *Proceedings of the 16th International Conference on the Foundations of Digital Games (FDG) 2021*. FDG ’21. Online: ACM, Aug. 2021
2. Lorenzo Gatti and Judith van Stegeren. “Improving Dutch sentiment analysis in Pattern”. In: *Computational Linguistics in the Netherlands Journal* 10 (Dec. 2020)
3. Judith van Stegeren and Mariët Theune. “Fantastic Strings and Where to Find Them: The Quest for High-Quality Video Game Text Corpora”. In: *Proceedings of the 2020 Workshop on Intelligent Narrative Technologies*. Oct. 2020
4. Thérèse Bergsma, Judith van Stegeren, and Mariët Theune. “Creating a Sentiment Lexicon with Game-Specific Words for Analyzing NPC Dialogue in The Elder Scrolls V: Skyrim”. English. In: *Workshop on Games and Natural Language Processing*. Marseille, France: European Language Resources

- Association, May 2020, pp. 1–9. ISBN: 979-10-95546-40-5. URL: <https://www.aclweb.org/anthology/2020.gamnlp-1.1>
5. Judith van Stegeren and Mariët Theune. “Remixing Headlines for Context-Appropriate Flavor Text”. In: *Proceedings of the IEEE Conference On Games*. London, UK: IEEE, Aug. 2019
 6. Judith van Stegeren and Mariët Theune. “Narrative Generation in the Wild: Methods from NaNoGenMo”. In: *Proceedings of the Second Workshop on Storytelling*. Florence, Italy: Association for Computational Linguistics, Aug. 2019, pp. 65–74. URL: <https://www.aclweb.org/anthology/W19-3407>
 7. Judith van Stegeren and Mariët Theune. “Churnalist: Fictional Headline Generation for Context-appropriate Flavor Text”. In: *Proceedings of the Tenth International Conference on Computational Creativity*. ICCV ’19. Charlotte, NC, USA: Association for Computational Creativity (ACC), 2019, pp. 65–72
 8. Judith van Stegeren and Mariët Theune. “Towards Generating Textual Game Assets from Real-world Data”. In: *Proceedings of the 13th International Conference on the Foundations of Digital Games*. FDG ’18. Malmö, Sweden: ACM, 2018, 43:1–43:4. ISBN: 978-1-4503-6571-0. DOI: 10.1145/3235765.3235809. URL: <http://doi.acm.org/10.1145/3235765.3235809>
 9. H Geuvers et al. “A type system for Continuation Calculus”. In: *Proceedings Fifth International Workshop on Classical Logic and Computation, CL&C 2014, Vienna, Austria, July 13, 2014*. Ed. by Paulo Oliva. Vol. 164. EPTCS. 2014, pp. 1–18

SIKS Dissertation Series

2016

- 2016-01 Syed Saiden Abbas (RUN), Recognition of Shapes by Humans and Machines
- 2016-02 Michiel Christiaan Meulendijk (UU), Optimizing medication reviews through decision support: prescribing a better pill to swallow
- 2016-03 Maya Sappelli (RUN), Knowledge Work in Context: User Centered Knowledge Worker Support
- 2016-04 Laurens Rietveld (VU), Publishing and Consuming Linked Data
- 2016-05 Evgeny Sherkhonov (UVA), Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers
- 2016-06 Michel Wilson (TUD), Robust scheduling in an uncertain environment
- 2016-07 Jeroen de Man (VU), Measuring and modeling negative emotions for virtual training
- 2016-08 Matje van de Camp (TiU), A Link to the Past: Constructing Historical Social Networks from Unstructured Data
- 2016-09 Archana Nottamkandath (VU), Trusting Crowdsourced Information on Cultural Artefacts
- 2016-10 George Karafotias (VUA), Parameter Control for Evolutionary Algorithms
- 2016-11 Anne Schuth (UVA), Search Engines that Learn from Their Users
- 2016-12 Max Knobbout (UU), Logics for Modelling and Verifying Normative Multi-Agent Systems
- 2016-13 Nana Baah Gyan (VU), The Web, Speech Technologies and Rural Development in West Africa - An ICT4D Approach
- 2016-14 Ravi Khadka (UU), Revisiting Legacy Software System Modernization
- 2016-15 Steffen Michels (RUN), Hybrid Probabilistic Logics - Theoretical Aspects, Algorithms and Experiments
- 2016-16 Guangliang Li (UVA), Socially Intelligent Autonomous Agents that Learn from Human Reward
- 2016-17 Berend Weel (VU), Towards Embodied Evolution of Robot Organisms
- 2016-18 Albert Meroño Peñuela (VU), Refining Statistical Data on the Web
- 2016-19 Julia Efremova (Tu/e), Mining Social Structures from Genealogical Data
- 2016-20 Daan Odijk (UVA), Context & Semantics in News & Web Search
- 2016-21 Alejandro Moreno Céleri (UT), From Traditional to Interactive Playspaces: Automatic Analysis of Player Behavior in the Interactive Tag Playground
- 2016-22 Grace Lewis (VU), Software Architecture Strategies for Cyber-Foraging Systems
- 2016-23 Fei Cai (UVA), Query Auto Completion in Information Retrieval
- 2016-24 Brend Wanders (UT), Repurposing and Probabilistic Integration of Data; An Iterative and data model independent approach
- 2016-25 Julia Kiseleva (TU/e), Using Contextual Information to Understand Searching and Browsing Behavior

- 2016-26 Dilhan Thilakaratne (VU), In or Out of Control: Exploring Computational Models to Study the Role of Human Awareness and Control in Behavioural Choices, with Applications in Aviation and Energy Management Domains
- 2016-27 Wen Li (TUD), Understanding Geo-spatial Information on Social Media
- 2016-28 Mingxin Zhang (TUD), Large-scale Agent-based Social Simulation - A study on epidemic prediction and control
- 2016-29 Nicolas Höning (TUD), Peak reduction in decentralised electricity systems - Markets and prices for flexible planning
- 2016-30 Ruud Mattheij (UvT), The Eyes Have It
- 2016-31 Mohammad Khelghati (UT), Deep web content monitoring
- 2016-32 Eelco Vriezekolk (UT), Assessing Telecommunication Service Availability Risks for Crisis Organisations
- 2016-33 Peter Bloem (UVA), Single Sample Statistics, exercises in learning from just one example
- 2016-34 Dennis Schunselaar (TUE), Configurable Process Trees: Elicitation, Analysis, and Enactment
- 2016-35 Zhaochun Ren (UVA), Monitoring Social Media: Summarization, Classification and Recommendation
- 2016-36 Daphne Karreman (UT), Beyond R2D2: The design of nonverbal interaction behavior optimized for robot-specific morphologies
- 2016-37 Giovanni Sileno (UvA), Aligning Law and Action - a conceptual and computational inquiry
- 2016-38 Andrea Minuto (UT), Materials that Matter - Smart Materials meet Art & Interaction Design
- 2016-39 Merijn Bruijnes (UT), Believable Suspect Agents; Response and Interpersonal Style Selection for an Artificial Suspect
- 2016-40 Christian Detweiler (TUD), Accounting for Values in Design
- 2016-41 Thomas King (TUD), Governing Governance: A Formal Framework for Analysing Institutional Design and Enactment Governance
- 2016-42 Spyros Martzoukos (UVA), Combinatorial and Compositional Aspects of Bilingual Aligned Corpora
- 2016-43 Saskia Koldijk (RUN), Context-Aware Support for Stress Self-Management: From Theory to Practice
- 2016-44 Thibault Sellam (UVA), Automatic Assistants for Database Exploration
- 2016-45 Bram van de Laar (UT), Experiencing Brain-Computer Interface Control
- 2016-46 Jorge Gallego Perez (UT), Robots to Make you Happy
- 2016-47 Christina Weber (UL), Real-time foresight - Preparedness for dynamic innovation networks
- 2016-48 Tanja Buttler (TUD), Collecting Lessons Learned
- 2016-49 Gleb Polevoy (TUD), Participation and Interaction in Projects. A Game-Theoretic Analysis
- 2016-50 Yan Wang (UVT), The Bridge of Dreams: Towards a Method for Operational Performance Alignment in IT-enabled Service Supply Chains

2017

- 2017-01 Jan-Jaap Oerlemans (UL), Investigating Cybercrime
- 2017-02 Sjoerd Timmer (UU), Designing and Understanding Forensic Bayesian Networks using Argumentation
- 2017-03 Daniël Harold Telgen (UU), Grid Manufacturing; A Cyber-Physical Approach with Autonomous Products and Reconfigurable Manufacturing Machines
- 2017-04 Mrunal Gawade (CWI), Multi-core Parallelism in a Column-store
- 2017-05 Mahdiah Shadi (UVA), Collaboration Behavior

- 2017-06 Damir Vandic (EUR), Intelligent Information Systems for Web Product Search
- 2017-07 Roel Bertens (UU), Insight in Information: from Abstract to Anomaly
- 2017-08 Rob Konijn (VU), Detecting Interesting Differences: Data Mining in Health Insurance Data using Outlier Detection and Subgroup Discovery
- 2017-09 Dong Nguyen (UT), Text as Social and Cultural Data: A Computational Perspective on Variation in Text
- 2017-10 Robby van Delden (UT), (Steering) Interactive Play Behavior
- 2017-11 Florian Kunneman (RUN), Modelling patterns of time and emotion in Twitter #anticipointment
- 2017-12 Sander Leemans (TUE), Robust Process Mining with Guarantees
- 2017-13 Gijs Huisman (UT), Social Touch Technology - Extending the reach of social touch through haptic technology
- 2017-14 Shoshannah Tekofsky (UvT), You Are Who You Play You Are: Modelling Player Traits from Video Game Behavior
- 2017-15 Peter Berck (RUN), Memory-Based Text Correction
- 2017-16 Aleksandr Chuklin (UVA), Understanding and Modeling Users of Modern Search Engines
- 2017-17 Daniel Dimov (UL), Crowdsourced Online Dispute Resolution
- 2017-18 Ridho Reinanda (UVA), Entity Associations for Search
- 2017-19 Jeroen Vuurens (UT), Proximity of Terms, Texts and Semantic Vectors in Information Retrieval
- 2017-20 Mohammadbashir Sedighi (TUD), Fostering Engagement in Knowledge Sharing: The Role of Perceived Benefits, Costs and Visibility
- 2017-21 Jeroen Linssen (UT), Meta Matters in Interactive Storytelling and Serious Gaming (A Play on Worlds)
- 2017-22 Sara Magliacane (VU), Logics for causal inference under uncertainty
- 2017-23 David Graus (UVA), Entities of Interest – Discovery in Digital Traces
- 2017-24 Chang Wang (TUD), Use of Affordances for Efficient Robot Learning
- 2017-25 Veruska Zamborlini (VU), Knowledge Representation for Clinical Guidelines, with applications to Multimorbidity Analysis and Literature Search
- 2017-26 Merel Jung (UT), Socially intelligent robots that understand and respond to human touch
- 2017-27 Michiel Joosse (UT), Investigating Positioning and Gaze Behaviors of Social Robots: People's Preferences, Perceptions and Behaviors
- 2017-28 John Klein (VU), Architecture Practices for Complex Contexts
- 2017-29 Adel Alhuraibi (UvT), From IT-BusinessStrategic Alignment to Performance: A Moderated Mediation Model of Social Innovation, and Enterprise Governance of IT"
- 2017-30 Wilma Latuny (UvT), The Power of Facial Expressions
- 2017-31 Ben Ruijl (UL), Advances in computational methods for QFT calculations
- 2017-32 Thaer Samar (RUN), Access to and Retrievability of Content in Web Archives
- 2017-33 Brigit van Loggem (OU), Towards a Design Rationale for Software Documentation: A Model of Computer-Mediated Activity
- 2017-34 Maren Scheffel (OU), The Evaluation Framework for Learning Analytics
- 2017-35 Martine de Vos (VU), Interpreting natural science spreadsheets
- 2017-36 Yuanhao Guo (UL), Shape Analysis for Phenotype Characterisation from High-throughput Imaging
- 2017-37 Alejandro Montes Garcia (TUE), WiBAF: A Within Browser Adaptation Framework that Enables Control over Privacy
- 2017-38 Alex Kayal (TUD), Normative Social Applications
- 2017-39 Sara Ahmadi (RUN), Exploiting properties of the human auditory system and compressive sensing methods to increase noise robustness in ASR

- 2017-40 Altaf Hussain Abro (VUA), Steer your Mind: Computational Exploration of Human Control in Relation to Emotions, Desires and Social Support For applications in human-aware support systems
- 2017-41 Adnan Manzoor (VUA), Minding a Healthy Lifestyle: An Exploration of Mental Processes and a Smart Environment to Provide Support for a Healthy Lifestyle
- 2017-42 Elena Sokolova (RUN), Causal discovery from mixed and missing data with applications on ADHD datasets
- 2017-43 Maaïke de Boer (RUN), Semantic Mapping in Video Retrieval
- 2017-44 Garm Lucassen (UU), Understanding User Stories - Computational Linguistics in Agile Requirements Engineering
- 2017-45 Bas Testerink (UU), Decentralized Runtime Norm Enforcement
- 2017-46 Jan Schneider (OU), Sensor-based Learning Support
- 2017-47 Jie Yang (TUD), Crowd Knowledge Creation Acceleration
- 2017-48 Angel Suarez (OU), Collaborative inquiry-based learning

2018

- 2018-01 Han van der Aa (VUA), Comparing and Aligning Process Representations
- 2018-02 Felix Mannhardt (TUE), Multi-perspective Process Mining
- 2018-03 Steven Bosems (UT), Causal Models For Well-Being: Knowledge Modeling, Model-Driven Development of Context-Aware Applications, and Behavior Prediction
- 2018-04 Jordan Janeiro (TUD), Flexible Coordination Support for Diagnosis Teams in Data-Centric Engineering Tasks
- 2018-05 Hugo Huurdeman (UVA), Supporting the Complex Dynamics of the Information Seeking Process
- 2018-06 Dan Ionita (UT), Model-Driven Information Security Risk Assessment of Socio-Technical Systems
- 2018-07 Jieting Luo (UU), A formal account of opportunism in multi-agent systems
- 2018-08 Rick Smetsers (RUN), Advances in Model Learning for Software Systems
- 2018-09 Xu Xie (TUD), Data Assimilation in Discrete Event Simulations
- 2018-10 Julienka Mollee (VUA), Moving forward: supporting physical activity behavior change through intelligent technology
- 2018-11 Mahdi Sargolzaei (UVA), Enabling Framework for Service-oriented Collaborative Networks
- 2018-12 Xixi Lu (TUE), Using behavioral context in process mining
- 2018-13 Seyed Amin Tabatabaei (VUA), Computing a Sustainable Future
- 2018-14 Bart Joosten (UVT), Detecting Social Signals with Spatiotemporal Gabor Filters
- 2018-15 Naser Davarzani (UM), Biomarker discovery in heart failure
- 2018-16 Jaebok Kim (UT), Automatic recognition of engagement and emotion in a group of children
- 2018-17 Jianpeng Zhang (TUE), On Graph Sample Clustering
- 2018-18 Henriette Nakad (UL), De Notaris en Private Rechtspraak
- 2018-19 Minh Duc Pham (VUA), Emergent relational schemas for RDF
- 2018-20 Manxia Liu (RUN), Time and Bayesian Networks
- 2018-21 Aad Sloomaker (OUN), EMERGO: a generic platform for authoring and playing scenario-based serious games
- 2018-22 Eric Fernandes de Mello Araujo (VUA), Contagious: Modeling the Spread of Behaviours, Perceptions and Emotions in Social Networks
- 2018-23 Kim Schouten (EUR), Semantics-driven Aspect-Based Sentiment Analysis
- 2018-24 Jered Vroon (UT), Responsive Social Positioning Behaviour for Semi-Autonomous Telepresence Robots
- 2018-25 Riste Gligorov (VUA), Serious Games in Audio-Visual Collections

- 2018-26 Roelof Anne Jelle de Vries (UT), Theory-Based and Tailor-Made: Motivational Messages for Behavior Change Technology
- 2018-27 Maikel Leemans (TUE), Hierarchical Process Mining for Scalable Software Analysis
- 2018-28 Christian Willemse (UT), Social Touch Technologies: How they feel and how they make you feel
- 2018-29 Yu Gu (UVT), Emotion Recognition from Mandarin Speech
- 2018-30 Wouter Beek, The "K" in "semantic web" stands for "knowledge": scaling semantics to the web

2019

- 2019-01 Rob van Eijk (UL), Web privacy measurement in real-time bidding systems. A graph-based approach to RTB system classification
- 2019-02 Emmanuelle Beauxis Aussalet (CWI, UU), Statistics and Visualizations for Assessing Class Size Uncertainty
- 2019-03 Eduardo Gonzalez Lopez de Murillas (TUE), Process Mining on Databases: Extracting Event Data from Real Life Data Sources
- 2019-04 Ridho Rahmadi (RUN), Finding stable causal structures from clinical data
- 2019-05 Sebastiaan van Zelst (TUE), Process Mining with Streaming Data
- 2019-06 Chris Dijkshoorn (VU), Nichesourcing for Improving Access to Linked Cultural Heritage Datasets
- 2019-07 Soude Fazeli (TUD), Recommender Systems in Social Learning Platforms
- 2019-08 Frits de Nijs (TUD), Resource-constrained Multi-agent Markov Decision Processes
- 2019-09 Fahimeh Alizadeh Moghaddam (UVA), Self-adaptation for energy efficiency in software systems
- 2019-10 Qing Chuan Ye (EUR), Multi-objective Optimization Methods for Allocation and Prediction
- 2019-11 Yue Zhao (TUD), Learning Analytics Technology to Understand Learner Behavioral Engagement in MOOCs
- 2019-12 Jacqueline Heinerman (VU), Better Together
- 2019-13 Guanliang Chen (TUD), MOOC Analytics: Learner Modeling and Content Generation
- 2019-14 Daniel Davis (TUD), Large-Scale Learning Analytics: Modeling Learner Behavior & Improving Learning Outcomes in Massive Open Online Courses
- 2019-15 Erwin Walraven (TUD), Planning under Uncertainty in Constrained and Partially Observable Environments
- 2019-16 Guangming Li (TUE), Process Mining based on Object-Centric Behavioral Constraint (OCBC) Models
- 2019-17 Ali Hurriyetoglu (RUN), Extracting actionable information from microtexts
- 2019-18 Gerard Wagenaar (UU), Artefacts in Agile Team Communication
- 2019-19 Vincent Koeman (TUD), Tools for Developing Cognitive Agents
- 2019-20 Chide Groenouwe (UU), Fostering technically augmented human collective intelligence
- 2019-21 Cong Liu (TUE), Software Data Analytics: Architectural Model Discovery and Design Pattern Detection
- 2019-22 Martin van den Berg (VU), Improving IT Decisions with Enterprise Architecture
- 2019-23 Qin Liu (TUD), Intelligent Control Systems: Learning, Interpreting, Verification
- 2019-24 Anca Dumitrache (VU), Truth in Disagreement - Crowdsourcing Labeled Data for Natural Language Processing
- 2019-25 Emiel van Miltenburg (VU), Pragmatic factors in (automatic) image description
- 2019-26 Prince Singh (UT), An Integration Platform for Synchromodal Transport
- 2019-27 Alessandra Antonaci (OUN), The Gamification Design Process applied to (Massive) Open Online Courses

-
- 2019-28 Esther Kuindersma (UL), Cleared for take-off: Game-based learning to prepare airline pilots for critical situations
 - 2019-29 Daniel Formolo (VU), Using virtual agents for simulation and training of social skills in safety-critical circumstances
 - 2019-30 Vahid Yazdanpanah (UT), Multiagent Industrial Symbiosis Systems
 - 2019-31 Milan Jelisavcic (VU), Alive and Kicking: Baby Steps in Robotics
 - 2019-32 Chiara Sironi (UM), Monte-Carlo Tree Search for Artificial General Intelligence in Games
 - 2019-33 Anil Yaman (TUE), Evolution of Biologically Inspired Learning in Artificial Neural Networks
 - 2019-34 Negar Ahmadi (TUE), EEG Microstate and Functional Brain Network Features for Classification of Epilepsy and PNES
 - 2019-35 Lisa Facey-Shaw (OUN), Gamification with digital badges in learning programming
 - 2019-36 Kevin Ackermans (OUN), Designing Video-Enhanced Rubrics to Master Complex Skills
 - 2019-37 Jian Fang (TUD), Database Acceleration on FPGAs
 - 2019-38 Akos Kadar (OUN), Learning visually grounded and multilingual representations
-
- 2020
-
- 2020-01 Armon Toubman (UL), Calculated Moves: Generating Air Combat Behaviour
 - 2020-02 Marcos de Paula Bueno (UL), Unraveling Temporal Processes using Probabilistic Graphical Models
 - 2020-03 Mostafa Deghani (UvA), Learning with Imperfect Supervision for Language Understanding
 - 2020-04 Maarten van Gompel (RUN), Context as Linguistic Bridges
 - 2020-05 Yulong Pei (TUE), On local and global structure mining
 - 2020-06 Preethu Rose Anish (UT), Stimulation Architectural Thinking during Requirements Elicitation - An Approach and Tool Support
 - 2020-07 Wim van der Vegt (OUN), Towards a software architecture for reusable game components
 - 2020-08 Ali Mirsoleimani (UL), Structured Parallel Programming for Monte Carlo Tree Search
 - 2020-09 Myriam Traub (UU), Measuring Tool Bias and Improving Data Quality for Digital Humanities Research
 - 2020-10 Alifah Syamsiyah (TUE), In-database Preprocessing for Process Mining
 - 2020-11 Sepideh Mesbah (TUD), Semantic-Enhanced Training Data Augmentation Methods for Long-Tail Entity Recognition Models
 - 2020-12 Ward van Breda (VU), Predictive Modeling in E-Mental Health: Exploring Applicability in Personalised Depression Treatment
 - 2020-13 Marco Virgolin (CWI), Design and Application of Gene-pool Optimal Mixing Evolutionary Algorithms for Genetic Programming
 - 2020-14 Mark Raasveldt (CWI/UL), Integrating Analytics with Relational Databases
 - 2020-15 Konstantinos Georgiadis (OUN), Smart CAT: Machine Learning for Configurable Assessments in Serious Games
 - 2020-16 Ilona Wilmont (RUN), Cognitive Aspects of Conceptual Modelling
 - 2020-17 Daniele Di Mitri (OUN), The Multimodal Tutor: Adaptive Feedback from Multimodal Experiences
 - 2020-18 Georgios Methenitis (TUD), Agent Interactions & Mechanisms in Markets with Uncertainties: Electricity Markets in Renewable Energy Systems
 - 2020-19 Guido van Capelleveen (UT), Industrial Symbiosis Recommender Systems
 - 2020-20 Albert Hankel (VU), Embedding Green ICT Maturity in Organisations
 - 2020-21 Karine da Silva Miras de Araujo (VU), Where is the robot?: Life as it could be

- 2020-22 Maryam Masoud Khamis (RUN), Understanding complex systems implementation through a modeling approach: the case of e-government in Zanzibar
- 2020-23 Rianne Conijn (UT), The Keys to Writing: A writing analytics approach to studying writing processes using keystroke logging
- 2020-24 Lenin da Nobrega Medeiros (VUA/RUN), How are you feeling, human? Towards emotionally supportive chatbots
- 2020-25 Xin Du (TUE), The Uncertainty in Exceptional Model Mining
- 2020-26 Krzysztof Leszek Sadowski (UU), GAMBIT: Genetic Algorithm for Model-Based mixed-Integer Optimization
- 2020-27 Ekaterina Muravyeva (TUD), Personal data and informed consent in an educational context
- 2020-28 Bibeg Limbu (TUD), Multimodal interaction for deliberate practice: Training complex skills with augmented reality
- 2020-29 Ioan Gabriel Bucur (RUN), Being Bayesian about Causal Inference
- 2020-30 Bob Zadok Blok (UL), Creatief, Creatieve, Creatiefst
- 2020-31 Gongjin Lan (VU), Learning better – From Baby to Better
- 2020-32 Jason Rhuggenaath (TUE), Revenue management in online markets: pricing and on-line advertising
- 2020-33 Rick Gilsing (TUE), Supporting service-dominant business model evaluation in the context of business model innovation
- 2020-34 Anna Bon (MU), Intervention or Collaboration? Redesigning Information and Communication Technologies for Development
- 2020-35 Siamak Farshidi (UU), Multi-Criteria Decision-Making in Software Production

2021

- 2021-01 Francisco Xavier Dos Santos Fonseca (TUD), Location-based Games for Social Interaction in Public Space
- 2021-02 Rijk Mercur (TUD), Simulating Human Routines: Integrating Social Practice Theory in Agent-Based Models
- 2021-03 Seyyed Hadi Hashemi (UVA), Modeling Users Interacting with Smart Devices
- 2021-04 Ioana Jivet (OU), The Dashboard That Loved Me: Designing adaptive learning analytics for self-regulated learning
- 2021-05 Davide Dell'Anna (UU), Data-Driven Supervision of Autonomous Systems
- 2021-06 Daniel Davison (UT), "Hey robot, what do you think?" How children learn with a social robot
- 2021-07 Armel Lefebvre (UU), Research data management for open science
- 2021-08 Nardie Fanchamps (OU), The Influence of Sense-Reason-Act Programming on Computational Thinking
- 2021-09 Cristina Zaga (UT), The Design of Robothings. Non-Anthropomorphic and Non-Verbal Robots to Promote Children's Collaboration Through Play
- 2021-10 Quinten Meertens (UvA), Misclassification Bias in Statistical Learning
- 2021-11 Anne van Rossum (UL), Nonparametric Bayesian Methods in Robotic Vision
- 2021-12 Lei Pi (UL), External Knowledge Absorption in Chinese SMEs
- 2021-13 Bob R. Schadenberg (UT), Robots for Autistic Children: Understanding and Facilitating Predictability for Engagement in Learning
- 2021-14 Negin Samaeemofrad (UL), Business Incubators: The Impact of Their Support
- 2021-15 Onat Ege Adali (TU/e), Transformation of Value Propositions into Resource Re-Configurations through the Business Services Paradigm
- 2021-16 Esam A. H. Ghaleb (UM), Bimodal Emotion Recognition from audio-visual cues
- 2021-17 Dario Dotti (UM), Human Behavior Understanding from motion and bodily cues using deep neural networks

- 2021-18 Remi Wieten (UU), Bridging the Gap Between Informal Sense-Making Tools and Formal Systems - Facilitating the Construction of Bayesian Networks and Argumentation Frameworks
- 2021-19 Roberto Verdecchia (VU), Architectural Technical Debt: Identification and Management
- 2021-20 Masoud Mansoury (TU/e), Understanding and Mitigating Multi-Sided Exposure Bias in Recommender Systems
- 2021-21 Pedro Thiago Timbó Holanda (CWI), Progressive Indexes
- 2021-22 Sihang Qiu (TUD), Conversational Crowdsourcing
- 2021-23 Hugo Manuel Proença (LIACS), Robust rules for prediction and description
- 2021-24 Kaijie Zhu (TUE), On Efficient Temporal Subgraph Query Processing
- 2021-25 Eoin Martino Grua (VUA), The Future of E-Health is Mobile: Combining AI and Self-Adaptation to Create Adaptive E-Health Mobile Applications
- 2021-26 Benno Kruit (CWI & VUA), Reading the Grid: Extending Knowledge Bases from Human-readable Tables
- 2021-27 Jelte van Waterschoot (UT), Personalized and Personal Conversations: Designing Agents Who Want to Connect With You
- 2021-28 Christoph Selig (UL), Understanding the Heterogeneity of Corporate Entrepreneurship Programs

2022

- 2022-01 Judith van Stegeren (UT), Flavor text generation for role-playing video games
 - 2022-02 Paulo da Costa (TU/e), Data-driven Prognostics and Logistics Optimisation: A Deep Learning Journey
-



UNIVERSITY OF TWENTE.