Thesis no: MECS-2014-04

Natural Language Generation for descriptive texts in interactive games

A combination of procedural content generation and natural language generation for quests.

Christopher Eliasson

Faculty of Computing
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Engineering. The thesis is equivalent to 20 weeks of full-time studies.

Contact Information:
Author(s):
Christopher Eliasson
E-mail: chel09@student.bth.se

University advisor:
Hans Tap
Dept. of Creative Technologies

Faculty of Computing
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

Abstract

Context. Game development is a costly process and with today's advanced hardware the customers are asking for more playable content, and at higher quality. For many years providing this content procedurally has been done for level creation, modeling, and animation. However, there are games that require content in other forms, such as executable quests that progress the game forward. Quests have been procedurally generated to some extent, but not in enough detail to be usable for game development without providing a handwritten description of the quest.
Objectives. In this study we combine a procedural content generation structure for quests with a natural language generation approach to generate a descriptive summarized text for quests, and examine whether the resulting texts are viable as quest prototypes for use in game development.
Methods. A number of articles on the area of natural language generation is used to determine an appropriate way of validating the generated texts produced in this study, which concludes that a user case study is appropriate to evaluate each text for a set of statements.
Results. 30 texts were generated and evaluated from ten different quest structures, where the majority of the texts were found to be good enough to be used for game development purposes.
Conclusions. We conclude that quests can be procedurally generated in more detail by incorporating natural language generation. However, the quest structure used for this study needs to expand into more detail at certain structure components in order to fully support an automated system in a flexible manner. Furthermore due to semantics and grammatics being key components in the flow and usability of a text, a more sophisticated system needs to be implemented using more advanced techniques of natural language generation.

Keywords: Procedural content generation, quests, text generation, game development.

Contents

Abstract **i**

1 Introduction **1**
1.1 Topic . . . 1
1.2 Background . . . 2
1.2.1 Procedural Content Generation . . . 3
1.2.2 Quests . . . 4
1.3 Problem Statement . . . 4
1.3.1 Research Questions . . . 5
1.3.2 Objectives . . . 5
1.4 Thesis Overview . . . 6

2 Related Work **7**
2.1 Underlying Quest Structure . . . 7
2.2 Common Steps of Natural Language Generation . . . 10
2.3 Three Module Steps . . . 11
2.4 Evaluation of NLG Systems . . . 12
2.4.1 Evaluation through Automatic Systems: BLEU . . . 13

3 Natural Language System **14**
3.1 Quest Structure Generator . . . 15
3.2 WordNet . . . 15
3.3 Motivation and Strategy . . . 15
3.4 The System . . . 16
3.4.1 Text Planner . . . 16
3.4.2 Sentence Planner . . . 17
3.4.3 Linguistic Realiser . . . 20

4 Method **23**
4.1 Validation Tests . . . 23
4.1.1 Motivation . . . 23
4.1.2 Setup . . . 23
4.1.3 Participants . . . 24
4.1.4 Task . . . 24

4.1.5 The Statements . . . 25
4.1.6 Quality . . . 25

5 Results **28**
5.1 Visualisation of Data . . . 28
5.1.1 Collected Data . . . 30
5.1.2 Values . . . 32

6 Analysis **33**
6.1 Generate a Procedural Quest Structure . . . 33
6.2 Quest Corpus Generation System . . . 33
6.3 Quest Corpus Validation . . . 34
6.3.1 Evaluation of the First Action Sequence . . . 34
6.3.2 Synonyms . . . 36
6.3.3 Sentence Aggregation . . . 39
6.3.4 The Worst Action Sequence . . . 39
6.3.5 Other Suggested Improvements . . . 40
6.4 Validity Threats . . . 41
6.4.1 Grammatical Knowledge of Participants . . . 41
6.4.2 Repetitive Corpuses . . . 41
6.4.3 Setup of the Case Study . . . 42

7 Conclusions and Future Work **43**
7.1 Conclusion . . . 43
7.2 Future Work . . . 44

Chapter 1

Introduction

The following sections provides an introduction to the aim of this paper, the areas of natural language generation and procedural content generation, basic terminology within these areas and their respective backgrounds in relation to the aimed topic. The later section provides an introduction to the problem statement, research objectives, and what can be expected in the different chapters throughout the paper.

1.1 Topic

This research experiments with the field of natural language generation, NLG, and procedural content generation, PCG, by combining them to both improve and introduce their use in interactive game development. Procedurally generated quest structures have been integrated with natural language generation in order to produce more complete quests usable for development of interactive games. The focus is on three different areas. The most prominent area is the area of game development since it is this area the results will improve, but the underlying key areas used for this improvement are natural language generation, and procedural content generation.

Natural language generation has had most of its focus in computational linguistics and is often used to produce information from automated systems, e.g. a system used to provide information on train arrivals and departures. Procedural content generation has been used heavily in the field of game development, where the focus is to generate in-game content to improve the replayability of the game and lowering the costs of content production.

The key idea of the research presented in this paper is to combine procedural content generation with natural language generation in order to improve the field of game development by providing more sophisticated procedural generation of quests. Quests have been procedurally generated before Doran and Parberry (2011), but not in combination with natural language generation.

The results of this study aims to expand the uses of natural language generation in game development by providing a method to procedurally generate quests where little or no designing is required. Furthermore the results provides

a method for game developers to either use simplified quests in their games or to generate prototype quests early in development.

1.2 Background

Natural language generation, NLG, is an area within the field of natural language processing, NLP, where the task is to generate natural language from various different machine representation systems, e.g. a to the machine defined knowledge base. The field falls under the category of Artificial Intelligence according to ACM classifications 2012 (ACM, 2012).

The area of NLP has been researched as far back as the 1950s (TURING, 1950). Natural language generation is a bit younger, and has had its main focus area in data analysis where it generates natural language based on analysis performed on data. The earliest system created that could generate natural language was FoG (Goldberg et al., 1994), which was able to generate natural language from data obtained for weather forecasts and was deployed in 1994.

Natural language generation has since then expanded into other uses. In 2007 IBM Research started developing a computer system with the goal of competing with it in the television game show *Jeopardy!*. To successfully compete the computer system, named *Watson* (IBM, 2014), had to interpret questions asked in the form of natural language as well as answering these questions in the same manner. Thus the development of *Watson* required a fair amount of focus on natural language generation. In 2011 *Watson* was put to the test against two former *Jeopardy!* champions in a nationally televised two-game *Jeopardy!* match and won (Ferrucci, 2012).

In later years NLG has been introduced to the development of games as well. In 2007 students from Georgia Institute of Technology introduced natural language generation for personality rich characters in interactive games and suggests a novel template-based system for believable agents (Strong et al., 2007). Furthermore Michael Mateas and Andrew Stern implemented an interactive drama application called Façade (Mateas and Stern, 2003) where the interactor responds to a dialogue between two artificial characters to guide the conversation towards different closures

Natural languages are of high complexity, and can contain an infinite amount of different words, sentences, and their meanings. To successfully generate a natural language sentence there needs to be defined rules and principles in order to generate a sentence that we humans can interpret. These rules are defined by a *grammar*. Like any natural language a grammar defines the set of rules that builds the structure of the language. The grammar needs to understand when a certain word can be used, which can be situational. Words can have different meanings depending on the scenario in which it is used, thus a form of *semantics* needs to be defined. *Semantics* are the examination of the meaning of words

and sentences, and convey useful information relevant to the scenario as a whole. Furthermore the grammar also needs to adjust the *syntax* of words, which is also situational and depends on which language is supposed to be generated.

There are generally six basic activities (Reiter and Dale, 1997) when generating natural language from machine representation data. The tasks included in these activities are sometimes combined with tasks in other activities resulting in fewer activities, but they are all present in some way. These activities contain fundamental ways of planning, creating, and evaluating words, phrases, or sentences, defined by the grammar. These activities also handles the syntax depending on the situational use of the sentence, the adding or removing of singular or plural forms known as *morphology*, and the capitalisation and decapitalisation of words known as *orthograpology*. A more detailed explanation of these activities can be found in the next chapter.

1.2.1 Procedural Content Generation

Another area that can be strongly correlated to NLG is procedural content generation, PCG. PCG generates content, in whatever field it is applied, procedurally and in game development this means less content that needs to be designed and created by a designer. In game development PCG has mainly been used in the areas of level creation, modeling and animation, going as far back as the 1980s with the well known game *Rogue*[1]. In today's games replayability is of high value and there are examples of games where the replayability and procedural elements of the game is what makes it a success. *The Binding of Isaac*[2] is an example where PCG is present for level and enemy generation. *The Binding of Isaac* is a rather one minded game in its core, but due to the replayability, and the simplistic manners of the environment you play in, the game can be played hours on end without losing its interest. According to Edmund McMillen, who is the creator of *The Binding of Isaac*, the game has sold well over 2 million copies[3].

This study continues on previous research on quest generation. Doran and Parberry (Doran and Parberry, 2011) evaluated quests from four different *Massively multiplayer online role-playing games*, MMORPG, and found strong similarities in quest structure between the different games despite having different developers and being different genres. They also proposed a generic quest structure corresponding to these similarities. They further implemented an application which produced entire quest chains from their designed quest grammar. In this study a combination of their proposed structure and natural language generation is proposed in order to add a descriptive text, known as *quest corpus*, to these generated quests.

[1] http://web.archive.org/web/20080715035939/http://roguelikes.sauceforge.net/pub/rogue/index.html
[2] http://store.steampowered.com/app/113200/
[3] http://www.youtube.com/watch?v=NxhIDco3sXo

1.2.2 Quests

In many different genres of games there are quests, or missions, that the player has to complete for the game to progress forward. In role-playing games, RPGs, these quests are commonly obtained and completed through interaction of a non playable character, NPC. The NPC has a specified reason for why it wants the quest to be completed, and informs the player how this reason should be satisfied, i.e what types of actions the player needs to perform for the quest to be completed. This information is often conveyed to the player through a quest corpus. The quest corpus is simply the informative text given by the quest, or NPC, that states what the player needs to do to successfully complete the quest.

The type of player-to-NPC-interaction for quests, and the structure of the quests, described above is common in general, not only in RPGs. World of Warcraft, Everquest, the Diablo series, and Eve Online are some examples of games where this quest style, or structure, is present.

There have been several studies aiming to generate these types of quests procedurally and to successfully do this there has to be a generalisation of the quest structure. Doran and Parberry (Doran and Parberry, 2011) proposed such a generalisation which made it possible to procedurally generate quest structures. Lee and Cho (Lee and Cho, 2012) continued on Doran and Parberry's research by combining their proposed structure with Petri Net Planning (Desel and Juhás, 2001) to generate the plot structure for the quests. Even if there have been accomplished ways of generating the base structure of quests no quest corpus has been attempted for generation, which would make the generation of quests more complete.

1.3 Problem Statement

Quests are well structured missions that are often connected to a form of story and progress which dictates the information presented to the player. To generate quests procedurally is a complex task since a newly generated quest needs to be generated with the course of events from a previously generated quest in mind. This type of problem is too complex for the span of this study and thus is not covered. Another issue, which is within the coverage of this study, is that a quest have to have some set of parameters that can be used in order to define a grammar that connects natural language with a specific quest. Thus the integration of natural language in quests needs a generic structure, and a way to use this structure to further set a grammar for natural language generation, in order to procedurally generate quests and their corresponding quest corpus.

From an economic standpoint in game development creating content is a costly and time consuming endeavour. Even if generating quest structures procedurally is a step forward, it does not eliminate the fact that a designer has to write

the quest corpus for those generated quests. Using PCG in combination with NLG creates more complete quests and increases the replayability of games in certain genres while at the same time open up more developing time and money in other areas of game development. The combination of these two areas holds the potential to produce quests procedurally which are playable at an early level in development and can have a lot of uses when prototyping gameplay mechanics or the quest integration itself. When implementing a quest system in a game there needs to be quests in order to test the functionality of the system. Even if these quests probably are written swiftly and does not end up in the final release of the game, it is still time consuming to write them. With procedurally generated quest structures and quest corpuses these quests can not only be created instantly for the purpose of prototyping, but also be used as a basic starting point to continue on when designing the actual quests shipped with the game.

By providing this method the economic cost of developing a game can be lowered significantly since there are games that require a vast amount of text in various aspects. As an example Activision Blizzard Entertainment recently released statistics with interesting data on their big hit game *World of Warcraft* in which as of their latest expansion, *Mists of Pandaria*, contains around 6 million words[4]. There are also around 9 500[5] different quests in the game with their own quest corpus.

1.3.1 Research Questions

The primary aim of this study is to explore a way to incorporate natural language generation with procedural content generation of an automated quest structure, and evaluate if this incorporation is feasible for game development in the aspect of automatic generation of quests. There are two questions that are to be answered throughout the study which can be viewed below.

- Can natural language be generated in conjunction with a procedurally generated quest structure to produce a corresponding quest corpus describing the generated quest?
- Will the quest corpus generated for this automated quest structure have a high enough quality to be usable in game development?

1.3.2 Objectives

There were three objectives to be reached in order to obtain the results necessary to address the aim of this research. These objectives were achieved in chronological order and are shown below.

[4]http://media.wow-europe.com/infographic/en/world-of-warcraft-infographic.html
[5]http://www.wowwiki.com/Quest

1. *Generate a procedural quest structure*
   Implementation of a suitable quest grammar structure to build the natural language system upon.

2. *Build quest corpus generation system*
   Implementation of the system that generates the quest corpuses with the quest grammar structure as a foundation.

3. *Quest validation*
   Validation of the quest corpuses done by evaluating them in a user case study through a set of evaluation statements.

1.4 Thesis Overview

The remaining sections are presented in different chapters where each chapter is a main section of the paper and provides an introduction to its content at the beginning. The chapters are briefly described in chronological order below to provide the reader with an indication of what to expect in the form of content throughout the paper.

Related Work provides important research done previously which this research is built upon, and describes advanced preliminaries in the topic areas, and theory used behind the solutions proposed.

Natural Language System describes the implemented natural language generation system in detail and provides a step by step explanation of the major components.

Method describes the the chosen method for evaluating the results of the NLG system which is done through a *user case study*.

Results provides the results acquired from the implemented systems and the validation process of the results.

Analysis analyses the results in accordance with research objectives.

Conclusion and Future Work provides a closure to the research performed and proposes future work in the same line of direction.

Chapter 2

Related Work

The sections in this chapter provides related research and papers on which this paper is built upon. The underlying grammar structure of the corpuses are explained in detail, as well as a simplistic approach to implementing a natural language generation system.

2.1 Underlying Quest Structure

In games it is common that the player interacts with an NPC in order to obtain, execute, and successfully complete a quest for a reward. Doran and Parberry evaluated more than 750 quests from four different *massively multiplayer online role-playing games*, MMORPGs, in order to find similarities in the quests between these games for the purpose of defining a general structure for such quests (Doran and Parberry, 2011). The evaluation of these quests revealed strong similarities between them and they were able to propose a general quest structure corresponding to these similarities. According to their study an NPC has to have a logical motivation to why it wants the quest to be completed in order for the quest to make sense to the player executing it. Analysis of the different quests categorized them into nine different motivations, where each motivation is further divided into different strategies which are ways for this motivation to be satisfied, i.e. different ways of completing a quest. Each strategy is composed of a set of atomic actions and non-atomic actions, further referred to as an *action sequence*. These actions signifies what the player is supposed to do, and in what order, to finish the specified quest. These non-atomic actions can in turn be composed of a set of other actions, creating longer quests or even long chains of quests. The motivations with their respective strategies, and the actions composing these strategies, can be viewed in table 2.1.

Motivation	Strategy	Sequence of Actions
Knowledge	Deliver item for study	<get> <goto> give
	Spy	<spy>
	Interview NPC	<goto> listen <goto> report
	Use an item in the field	<get> <goto> use <goto> give
Comfort	Obtain luxuries	<get> <goto> give
	Kill pests	<goto> damage <goto> report
Reputation	Obtain rare items	<get> <goto> give
	Kill enemies	<goto> <kill> <goto> report
	Visit a dangerous place	<goto> <goto> report
Serenity	Revenge, Justice	<goto> damage
	Capture Criminal (1)	<get> <goto> use <goto> give
	Capture Criminal (2)	<get> <goto> use <capture> <goto> give
	Check on NPC (1)	<goto> listen <goto> report
	Check on NPC (2)	<goto> take <goto> give
	Recover lost/stolen item	<get> <goto> give
	Rescue captured NPC	<goto> damage escort <goto> report
Protection	Attack threatening entities	<goto> damage <goto> report
	Treat or repair (1)	<get> <goto> use
	Treat or repair (2)	<goto> repair
	Create Diversion (1)	<get> <goto> use
	Create Diversion (2)	<goto> damage
	Assemble fortification	<goto> repair
	Guard Entity	<goto> defend
Conquest	Attack enemy	<goto> damage
	Steal stuff	<goto> <steal> <goto> give
Wealth	Gather raw materials	<goto> <get>
	Steal valuables for resale	<goto> <steal>
	Make valuables for resale	repair
Ability	Assemble tool for new skill	repair use
	Obtain training materials	<get> use
	Use existing tools	use
	Practice combat	damage
	Practice skill	use
	Research a skill (1)	<get> use
	Research a skill (2)	<get> experiment
Equipment	Assemble	repair
	Deliver supplies	<get> <goto> give
	Steal supplies	<steal>
	Trade for supplies	<goto> exchange

Table 2.1: Strategies for each of the motivations (Doran and Parberry, 2011).

In total there are 28 different actions, where 8 are non-atomic and 20 are atomic, that can compose a quest or chains of quests. The atomic actions are final actions that are not composed of any other actions, while the non-atomic actions can represent a sequence of other actions. Table 2.2 shows all the atomic actions. Table 2.3 illustrates the different non-atomic actions and which different combinations of actions they can be composed of.

	Action
1.	ε
2.	capture
3.	damage
4.	defend
5.	escort
6.	exchange
7.	experiment
8.	explore
9.	gather
10.	give
11.	goto
12.	kill
13.	listen
14.	read
15.	repair
16.	report
17.	spy
18.	stealth
19.	take
20.	use

Table 2.2: All atomic actions (Doran and Parberry, 2011)

	Action
1.	<subquest> ::= <goto>
2.	<subquest> ::= <goto> <QUEST> goto
3.	<goto> ::= ε
4.	<goto> ::= explore
5.	<goto> ::= <learn> goto
6.	<learn> ::= ε
7.	<learn> ::= <goto> <subquest> listen
8.	<learn> ::= <goto> <get> read
9.	<learn> ::= <get> <subquest> give listen
10.	<get> ::= ε
11.	<get> ::= <steal>
12.	<get> ::= <goto> gather
13.	<get> ::= <goto> <get> <goto> <subquest> exchange
14.	<steal> ::= <goto> stealth take
15.	<steal> ::= <goto> <kill> take
16.	<spy> ::= <goto> spy <goto> report
17.	<capture> ::= <get> <goto> capture
18.	<kill> ::= <goto> kill

Table 2.3: Action rules in Backus-Naur Form (Doran and Parberry, 2011).

2.2 Common Steps of Natural Language Generation

In the field of natural language generation there are, to some extent, a consensus that natural language generation is done through six steps (Reiter and Dale, 1997). How these steps are implemented or designed varies from system to system and the practices in the field are targets of debate, but each step is present at some level. The steps in chronological order is briefly described below.

1. **Content Determination** is the task of determining what information should be communicated in the end resulting corpus. This is done by creating messages from the input data available for the system and is generally highly domain dependent. Reiter and Dale (Reiter and Dale, 1997) explains that the messages created are often expressed in a formal language and labels *entities, relationships, and concepts* in the application domain.

2. **Discourse Planning** takes the data provided by the previous step and structures it into a tree where the leaves are messages. The messages in turn contains the data and also how messages are connected to other messages. In other words this step takes the data and structures it logically depending on how the desired result is to be presented.

3. **Sentence Aggregation** further combines these messages into sentences. It can also combine several sentences into larger ones in order to make the corpus more readable or to create a better flow of the text. As input it takes a tree-structure, with messages as nodes, and further develops a new tree-structure with more composed messages.

4. **Lexicalization** focuses on which words should be selected to describe the tree-structure provided by the sentence aggregation step.

5. **Referring Expression Generation** is commonly viewed as a description task where the task is to include enough information in the description to unambiguously identify the target entity.

6. **Linguistic Realisation** is the final step and focuses on applying the rules of the grammar to produce a corpus which is syntactically, morphologically, and orthographically correct, i.e. this step creates the actual sentences to communicate the data representation in previous steps.

Morphology is a broader area than just removing or adding correct syntax for singular and plural forms of words. Ahmad labels area as a stage that includes identifying words in a sentence or text that interact together. The information gathered by performed morphology is used in later stages to change the syntactical presentation of words (Ahmad, 2007).

Orthography is the process of adding punctuation, word breaks, and capitalization to name a few. In this study orthography is used in the most simplistic manner and mainly handles the capitalization of words in sentences.

2.3 Three Module Steps

There are different practices to choose from when developing a natural language generation system, and which practice to be used varies depending on how sophisticated and complex the system should be. For basic generation of natural language Reiter and Dale propose a three step module approach where the previously described steps are combined into three modules (Reiter and Dale, 1997). This approach is adopted in this study since this is one of the most common approaches and the target corpus, which is only a few sentences long, does not require a too complex or sophisticated generation system. Below is a brief description of the modules.

1. **Text Planning** is the first step and combines the first two steps described previously: *Content Determination* and *Discourse Planning.*

2. **Sentence Planning** is the middle step which combines the steps of *Sentence Aggregation*, *Lexicalization*, and *Referring Expression Generation.*

3. **Linguistic Realisation** is the final step and is the step described previously.

2.4 Evaluation of NLG Systems

To ensure the quality of a natural language processing system it needs to be evaluated somehow. Galliers and Jones discusses that it is necessary to recognise that a system's evaluation criteria may vary depending on if the system is being evaluated in itself, or in the environment in which the system is to be used (Galliers and Jones, 1993). An NLP system is often evaluated in performance and normally falls under *intrinsic* and *extrinsic* criteria. Intrinsic criteria are related to the system's objective, e.g. the speed of generation, and extrinsic criteria are related to the system's function, e.g. usability of the generated corpus. There are indeed several different aspects to consider when evaluating a natural language processing system.

The evaluation of an NLG system is also of great importance for several reasons, e.g. to demonstrate the progress for fellow researchers. Mellish and Dale suggests that there are three main categories to choose from when evaluating NLG systems (Mellish and Dale, 1998).

1. Evaluating properties of the theory, where the focus is to evaluate the underlying theory of the NLG system.

2. Evaluating properties of the system, focusing on assessing chosen characteristics of the NLG system.

3. Application potential, where the focus is to evaluate the system's potential from a specific environmental aspect. This could, for instance, determine if the system is a better solution than alternative solutions in this environment (Mellish and Dale, 1998).

The NLG system produced in this paper is evaluated with the third approach in mind, however the resulting corpus generated by the system is the focus of evaluation and not the system itself. One could argue that evaluating the corpus is part of evaluating the NLG system, but the system produced in this research is quite simplistic and extracting data other than action sequence and the generated corpus is not feasible.

Furthermore there are different ways of performing evaluation within these three main categories. Mellish and Dale discuss a few common approaches where

the one with most resemblance to the evaluation done in this paper is *fluency/intelligibility evaluation* (Mellish and Dale, 1998). This kind of evaluation focuses on assessing the readability of the text in three different steps: *measuring comprehension time, measuring the time taken to post-edit the text to a state of fluency,* and *asking human subjects directly to assess the readability of the text.* The assessment done in our research is, as previously stated, performed solely on the generated corpuses and is done through a user case study where participants were tasked with rating each corpus through a set of statements.

2.4.1 Evaluation through Automatic Systems: BLEU

Another common approach to evaluate corpuses are through an automated system, where one more successful evaluation system is BLEU (Papineni et al., 2001). BLEU uses n-gram-based evaluation metrics by comparing the output of a machine translation system to a defined perfect translation standard. BLEU has over the past years been used as a standard to rate a machine translation system by, or certain optimizations within different systems. However, BLEU is only relevant if it correlates with human judgement as well (Belz and Reiter). A study performed by Callison-Burch, Osborne, and Koehn concludes that BLEU is, however, not as reliable as once thought (Callison-Burch et al., 2006). They argue that BLEU is insufficient due to there being a wide variety of texts that are not grammatically or semantically reasonable, but which obtains the same BLEU score but where the human evaluations differ. The evaluation process for BLEU is done by evaluating the output corpus of a system with other reference texts, which are usually human written. In this study this is not a plausible solution since the corpuses generated are of a much lower complexity than the usual systems evaluated by automatic systems. It is also not plausible due to the vast amount of reference texts needed to perform this evaluation. Belz and Reiter suggests that human evaluation is still the most prominent way of evaluating texts, which is done in this paper.

Chapter 3

Natural Language System

The following chapter describes the implementation and labeling process for the natural language generation structure, and the quest structure generation system based on Doran and Parberry's research presented in the previous chapter. Each of the three modules described in the previous chapter is explained in detail. In Figure 3.1 a brief overview of the natural language generation system is presented.

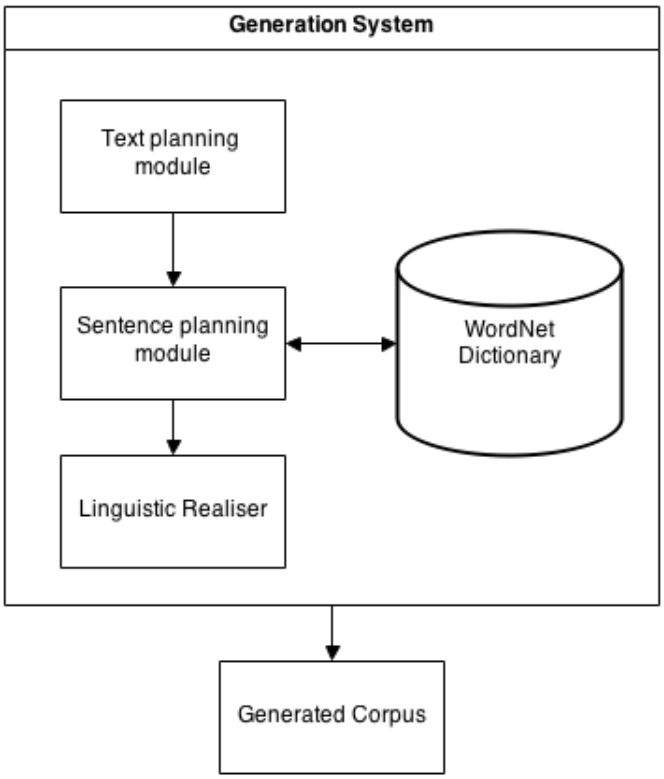

Figure 3.1: An overview of the natural language generation system.

3.1 Quest Structure Generator

The underlying grammar used for the generation of the quest corpuses was, as stated in previous chapter, a grammar Doran and Parberry suggested in their research. They also presented a prototype generator to generate quest structures using this grammar, however the source code for the generator is not available online. Due to the unavailability of the generator a new one had to be implemented in order to build the natural language generation system. The implementation of the quest structure generator was done in C++ and supports almost all action sequences presented in Table 2.1 and Table 2.3. The action sequences excluded from our version of the quest generator is the *<subquest>* and *<QUEST>*. The decision to exclude these sequences was taken to narrow down the scope of the natural language generation system. It was also logical in an interactive game perspective since an NPC is never aware of what the next NPC will ask of the player, i.e. the structure of a later quest or subquest. Unless the aim of this research was to generate natural language for whole quest chains it would not be necessary to support these sequences, and even then the natural language generated from each action sequence would still be quite separate since each quest, or subquest, is usually highly unaware of each other in a game environment.

The quest generation system generates an action sequence by choosing a motivation, and a corresponding strategy for that motivation, randomly and builds a quest structure according to Table 2.1 and Table 2.3. The structure is saved to file for later input to the natural language generation system.

3.2 WordNet

In an attempt to add more versatile sentences and a more varying table of possible synonyms for each word in the corpus, an online vocabulary was integrated called WordNet (Miller et al., 1990). WordNet can be used both online or as a downloadable asset to your project. There are a few different ways of incorporating this database of words into your project and the one used in this implementation was the C++ API. However, it was hard to find any code examples regarding their API and they state on their homepage that they do not provide support on programming issues. With that stated, the WordNet API was integrated in the natural language system, but to a highly limited use. The WordNet API was integrated in the second module and is discussed later in this chapter.

3.3 Motivation and Strategy

From Table 2.1 there are two columns titled motivation respectively strategy. The information that strategy is coupled to motivation is not passed on and handled

by the implemented system. The reason for this is that there is no concrete data or structure conveyed other than a keyword for the motivation, and a sentence for the strategy. There is thus no efficient way in generating the context that this keyword or sentence implies that the action sequence is a part of. The only way to currently convey information about the context of the action sequence is to add a type of variable or concept to each strategy and somehow connect this to the action sequence by a set of chosen parameters. Due to this complication motivation and strategy was not conveyed to the system, and thus not handled in this study.

3.4 The System

The next sections describes the implementation process of the natural language system in detail showing the journey of an action sequence through each module.

3.4.1 Text Planner

As previously stated natural language generation is the task of generating natural language from data in a defined system. The first step towards this goal is the text planner module which is responsible for creating a *text plan*. An overview of the text planning module can be viewed in Figure 3.2.

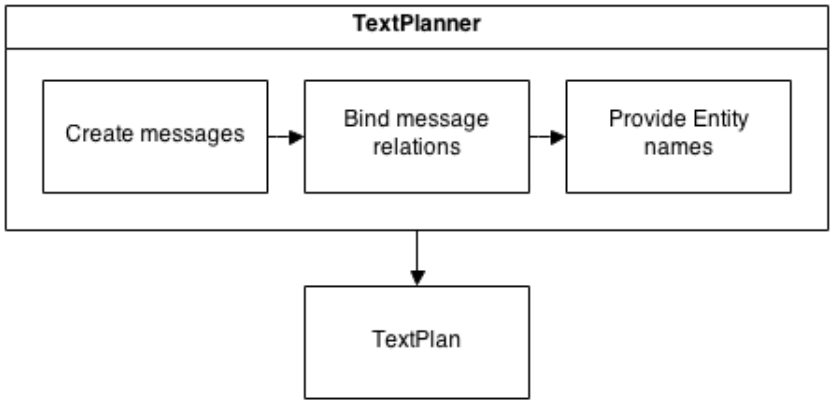

Figure 3.2: An overview of the text planning module.

The text plan consists of *messages* with all the necessary data that should be communicated in the end resulting corpus. This data includes the action sequence, each action's relation to another action, their different parameters, and entities to which the actions refer to. For explanatory simplicity we will use *<get> <goto> give* as an example sequence for the remainder of this chapter. The first step to creating the text plan is to create more composed messages from the input

message. The input message is simply a structure of actions which are divided into one message per action. In this case there will be three composed messages in the text plan - one for each action. Since the system is built upon an already defined grammar, and the grammar is used to provide the information to be communicated in the text, this is all there is for the *content determination* part.

The other part of this module is to bind each message's relation to other messages. To do this each message contains information on what relations to have to other messages, depending on the type of its contained action. Looking at the example sequence each action is referring to one or several entities. Seeing as the actions are all of the same sequence it is reasonable to think that *<get>* and *give* are referring to the same object, or objects. *<goto>* can refer to either an *NPC* or a *location*, but for the sake of simplicity we decide it to be a location. The last action, *give*, refers to more than one entity. It refers to an *item*, but also to an *NPC* to which this *item* should be delivered. The <get>-message is of the type *GET* and has one parameter to build relations to, an *item* which in this case is *item(1)*. The *give*-message is created similarly, but has two parameters to build relations upon. Seeing as one of these parameters is the *item(1)* it will construct a relation of the type *IDENTITY* with the <get>-message.

Once the relations are created for each message in the text plan, each entity is provided with a randomized name to add more diversity and personality to the text plan. Table 3.1 shows the example action sequence with its resulting text plan. This text plan is forwarded to the next module, *Sentence Planner*, for further processing.

Action sequence	Text plan
<get> <goto> give	*<**get**>item(1)<**goto**>location **give** item(1)-NPC(1)*

Table 3.1: An example of an action sequence and the resulting text plan.

3.4.2 Sentence Planner

From the text plan, forwarded by the text planner, the sentence planner performs different tasks to transform the text plan into sentence plans, which contains more explicit information of the messages and also aggregates these messages into sentences. An overview of the sentence planning module can be viewed in Figure 3.3.

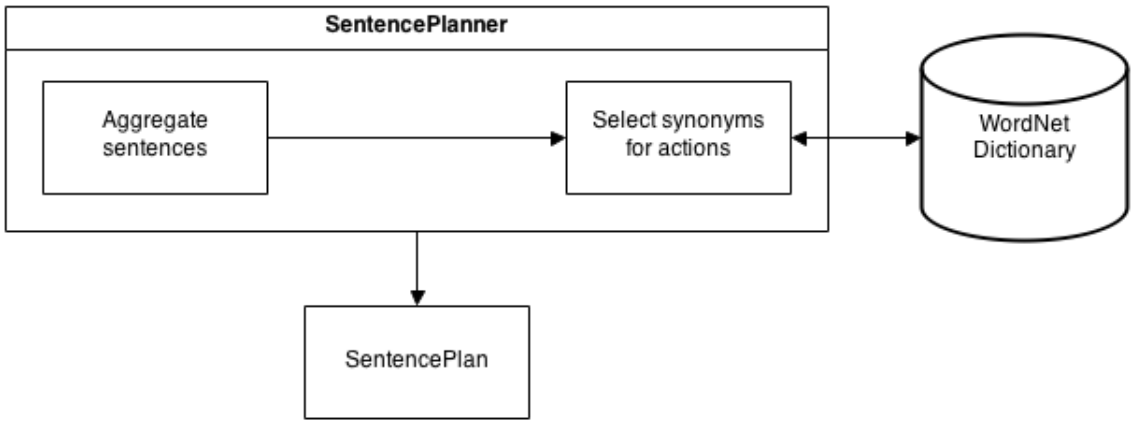

Figure 3.3: An overview of the sentence planning module.

The sentence aggregation is done using the most simplistic of approaches. Since the output corpus is short one could argue if the aggregation into sentences should be done for the entire text plan and result in one sentence. However, the text plan can contain as much as 6 different actions, even if the general text plan is between 2-4 actions. If we consider a sentence in general it can be as small as one single word, or longer than 20. Since the output corpus will contain entities as well as the descriptive actions and corresponding prepositions aggregation was decided to be done from 1 to 3 actions in sequence. Table 3.2 shows the three different outcomes of our previous example sequence after the sentence aggregation has been performed.

Aggregation	Sentences
1 sentence	<***get***>*item(1)*, <***goto***>*location* ***and give*** *item(1)-NPC(1).*
2 sentences	<***get***>*item(1)* ***and*** <***goto***>*location.* ***give*** *item(1)-NPC(1).*
3 sentences	<***get***>*item(1).* <***goto***>*location.* ***give*** *item(1)-NPC(1).*

Table 3.2: Three possible aggregations of messages performed on the example text plan forwarded from the text planner.

Looking at the generated quest structure each action is, more or less, a self explanatory word which dictates what the player needs to execute as a next step towards completion of the quest. To decide which words to properly describe the messages with is the next step of the sentence planner. This is done by attaching a *keyword*, which is a better descriptive synonym to what the action infers the player to do, to each action in the sequence. The synonym selected requires the action to be unambiguous to work, i.e. a <get>-action must always refer to obtaining something or someone, and not e.g. refer to the cognitive meaning of

the word *get*. This is one reason to why it is important that a grammar must be unambiguous. Table 3.3 shows the chosen keywords for the action sequence in the text plan.

Action	Keyword
<get>	Obtain
<goto>	Travel
give	Give

Table 3.3: Some actions with their respective keywords.

A problem that occurs with this approach is that keywords attached to the actions are always the same, which produces monotone sentences. This is solved using a vocabulary, *WordNet*, where synonyms to the keywords of the actions are also connected to that action. The result is sentences where the actions are described by an array of different words chosen at random among the synonyms for that particular action. Some keywords do not have synonyms connected to them. The reason for this is that the WordNet framework does not provide any example code or declaration on how to extract certain synonyms based on the situational use of the word, and there was simply not enough time to dig into this framework any further. The words that have synonyms attached simply had a small subset of synonyms which made it easier to control, and the words that does not have synonyms attached had too large subsets which made it impossible to filter. The word *give* had over a 100 synonyms depending on situational use, whereas *travel* had only seven. For the sake of simplicity words with too many synonyms was not provided with any.

In table 3.4 the corpus structure is shown using the keyword and different synonyms to that keyword.

Corpus structure	
Keyword	***Obtain*** *item(1)* ***Travel*** *location* ***Give*** *item(1)-NPC(1)*
Synonyms (1)	***Get*** *item(1)* ***Go*** *location* ***Give*** *item(1)-NPC(1)*
Synonyms (2)	***Find*** *item(1)* ***Journey*** *location* ***Give*** *item(1)-NPC(1)*

Table 3.4: Corpus structure using keywords and synonyms instead of the action type.

The final step of the sentence planner is to provide information to unambiguously identify the target entity. To do this the sentence planner considers the relations between messages created by the text planner. The only supported relation expression so far is on item entities. Going through each message's relation the

name of an item is changed to its personal pronoun if it has been referred to previously in the sentence plan. This is done to make it less repetitive in the output corpus. The only referring generation done other than that of items is to make sure that a location where the player should already be positioned in, following a previous <goto>-action, is removed from the sentence plan. Also occurrences of the same NPCs is only done once as well, simply because killing an NPC and then reporting back to the same NPC is a bit illogical. The sentence plan is thus done creating sentence plans for each sentence and sends the tree of sentences to the *Linguistic realiser*, which is the final module, for further processing. Table 3.5 shows the sentence plans, for two sentences using our example sequence and their selected names and keywords, forwarded to the final module.

Sentence	Sequence
1.	***Obtain*** *Horn of Valere and* ***Travel*** *Caemlyn.*
2.	***Give*** *Horn of Valere-Rand al'Thor.*

Table 3.5: The sentence plans forwarded to the Linguistic realiser.

3.4.3 Linguistic Realiser

The linguistic realiser is the last module of the generation system. An overview of the module is depicted in Figure 3.4.

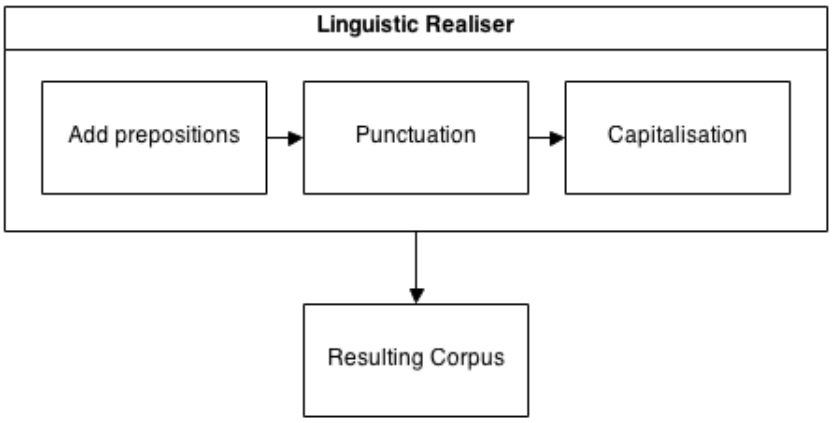

Figure 3.4: An overview of the linguistic realiser module.

There is not a defined grammar used for the linguistic realiser other than one for prepositions. The corpus is short and the structure grammar provided by Doran and Parberry is already structuring the information communicated in the corpus well. The prepositions needs to be handled though and this is done depending on what type of entity the actions are referring to. Ambiguities in atomic actions

is a problem here. Doran and Parberry's categorization of atomic actions is not enough to use as they are. It becomes clear if we consider the atomic action *give*. This action refers to two entities, an item or NPC that should be delivered to someone and this someone is the other entity. However the atomic action *listen* is referring to only one entity, an NPC to which the player should listen. In the grammar there is nothing that tells *give* and *listen* apart from each other which creates a problem for the generation of prepositions.

Categorization of atomic actions needs to be done in order for the prepositions to be generated correctly. This categorization is done by labeling the entities to different preposition types depending on the action in which the entity is used. Let us look at the atomic action *give* as an example. This action is labeled as *ItemToTarget*, which specifies that this action is related to an item and a target, which is either introduced by a previous action in the sequence or this action itself. The prepositions generated for this action will thus be decided to be *the* and *to*, corresponding to an item which is given to a target. Table 3.6 shows some different types of labels on atomic actions used.

Action	Type	Produced prepositions
give	ItemToTarget	*the, to*
listen	ToTarget	*to*
experiment	OnTarget	*on*

Table 3.6: Atomic actions, their labels, and their produced prepositions.

Furthermore the linguistic realiser takes care of punctuation and capitalization of letters in the beginning of a sentence. When each sentence plan has been iterated through each task of this module they are all added to the output corpus as the finished text for this quest. Table 3.7 shows a few example action sequences and their corresponding output corpus.

Action sequence	Output corpus
<get> <goto> give	Find the Staff of Imprisonment, travel to Shayol Ghul and give it to Lews Therin Telamon.
<goto> <kill> <goto> report	Journey to Caemlyn, kill Mordeth and go to Almoth Plain. Report to Artur Hawkwing.
<goto> listen <goto> report	Move to Almoth Plain and listen to Perrin Aybara. Journey to Altara and report to Rand al'Thor.
<get> <goto> use <capture> <goto> give	Find the Half moon axe, travel to Altara, use it and capture Matrim Cauthon. Travel to Tarabon and give it to Ingtar.

Table 3.7: Actions sequences with their respective output corpuses.

Chapter 4

Method

This section provides the chosen method for evaluation of the generated corpuses provided by the NLG system described in the previous chapter. The following sections provides information on how the chosen method was set up and performed.

4.1 Validation Tests

To validate the usefulness of the quest corpuses that the natural language generation system provides testing was performed on *grammatics*, *semantics* and *usability*. The tests were performed in a user case study and is thoroughly explained in the following sections.

4.1.1 Motivation

A case study is used to examine a certain event or phenomenon and includes a limited number of cases (Berndtsson et al., 2008). This also allows that each case can be examined, evaluated and analysed independently on the other cases. Each action sequence presented is seen as its own case. This is convenient since the results of one action sequence does not necessarily apply to the results of another action sequence. Furthermore this approach makes it easy to evaluate each action sequence individually.

4.1.2 Setup

For the user case study there were corpuses generated from a total of 10 different action sequences picked from strategies in table 2.1. The decision on which 10 sequences to pick among the different 39 was done by evaluating the originality of each sequence. Several of the sequences were similar, or in some cases exactly the same. Deciding on a total of 10 sequences also narrowed down the time the tests would take and at the same time provided the most varying corpuses for testing.

1.	<get> <goto> give
2.	<goto> listen <goto> report
3.	<goto> <kill> <goto> report
4.	<goto> use <goto> report
5.	<get> <goto> use <capture> <goto> give
6.	<goto> take <goto> give
7.	<goto> damage escort <goto> report
8.	<goto> <steal> <goto> give
9.	<goto> experiment
10.	<goto> repair

Table 4.1: 10 different action sequences used.

Seeing as each action in the action sequence is described by a keyword and randomizes synonyms connected to this keyword there could be synonyms that do not have good synergy with other synonyms in the same context. This could prove a corpus to be bad solely due to the chosen synonyms, thus each sequence was generated into three different corpuses. Thus this user case study was performed on 30 different corpuses generated from ten different action sequences.

In total there were 15 people participating in the study. Seeing as 30 corpuses were too many for each participant to read through and evaluate, the participants were divided into three groups with five participants in each. The corpuses were divided into these groups such that each group had one corpus generated from each action sequence, thus a total of ten corpuses. The ten action sequences selected for this user case study are shown in table 4.1.

4.1.3 Participants

The participants all had experience with games where quests were present in some way. This was a requirement to make sure that the participants knew what a quest was and how it is usually used in different games and genres. The participants were also required to have a good understanding of English in order to judge the grammatical and semantic aspects of the corpuses they read.

4.1.4 Task

The participants were presented with 10 corpuses, one from each action sequence, and where participants from the same group were given the same 10 corpuses. Their task was to rate these texts individually through a set of statements. The participants answered each statement as an seven-level Likert item (Jamieson, 2004), where seven was the best obtainable score for the corpus and implied that the participant fully agreed with the statement, and one as the worst and implied that the participant completely disagreed with the statement. The two most

common scale sizes used for Likert scales are 5-scale and 7-scale. In the case study the 7-point Likert-scale was used since this gives the user more choices, without adding too much choice, which is slightly better than a 5-point Likert scale (Nunnally and Bernstein).

4.1.5 The Statements

If a statement was answered with a low score on the scale the participant was asked to motivate their answer and suggest improvements. This was a measure taken not only to find areas in the corpus where improvements could be made, but also to make sure that the participant had understood the corpus and were enough educated in English grammar to be able to judge and criticize it correctly. There were a total of three statements presented to the participants where each statement aimed to evaluate the corpus from the different aspects of *grammatics*, *semantics*, and *usability*. These aspects are further referred to as *aspects of validation*.

Grammatics is the aspect where the participant rates the corpus grammatically. This is to ensure how grammatically correct the corpus is, while at the same time judging the participants grammatical knowledge of the English language.

Semantics is the aspect where the participant rates the corpus semantically. This is to ensure that the corpus convey useful information to the reader, while also making sure that the participant has successfully understood the quest.

Usability is the aspect where the participant rates the corpus on usability in the context of how usable the quest corpus is from an interactive game perspective. This is a measure to evaluate the quality of the corpus.

The questionnaire used for the user case study is depicted in Figure 4.1.

4.1.6 Quality

The quality needed for the corpus to be usable in game development is not explicitly defined, but rather done from an analysis perspective. Since the question conveyed to the participants for the case study is if the corpus can be used in an interactive game, and not in game development, there is no need to explicitly define it. If the quest is rated as being usable in an interactive game that would suggest that the corpus is good enough to be used in development for such a game

as well. It is therefore more suitable to define what quality is needed to be usable in an interactive game. The quality needed for the quest corpus to be usable in an interactive game is set to the max obtainable score on the Likert scale, i.e. 7. The logic behind this is that there can be no inconsistencies in grammatics or semantics when conveying information to a user in an interactive game, since this could confuse the user which in turn harms the gameplay experience.

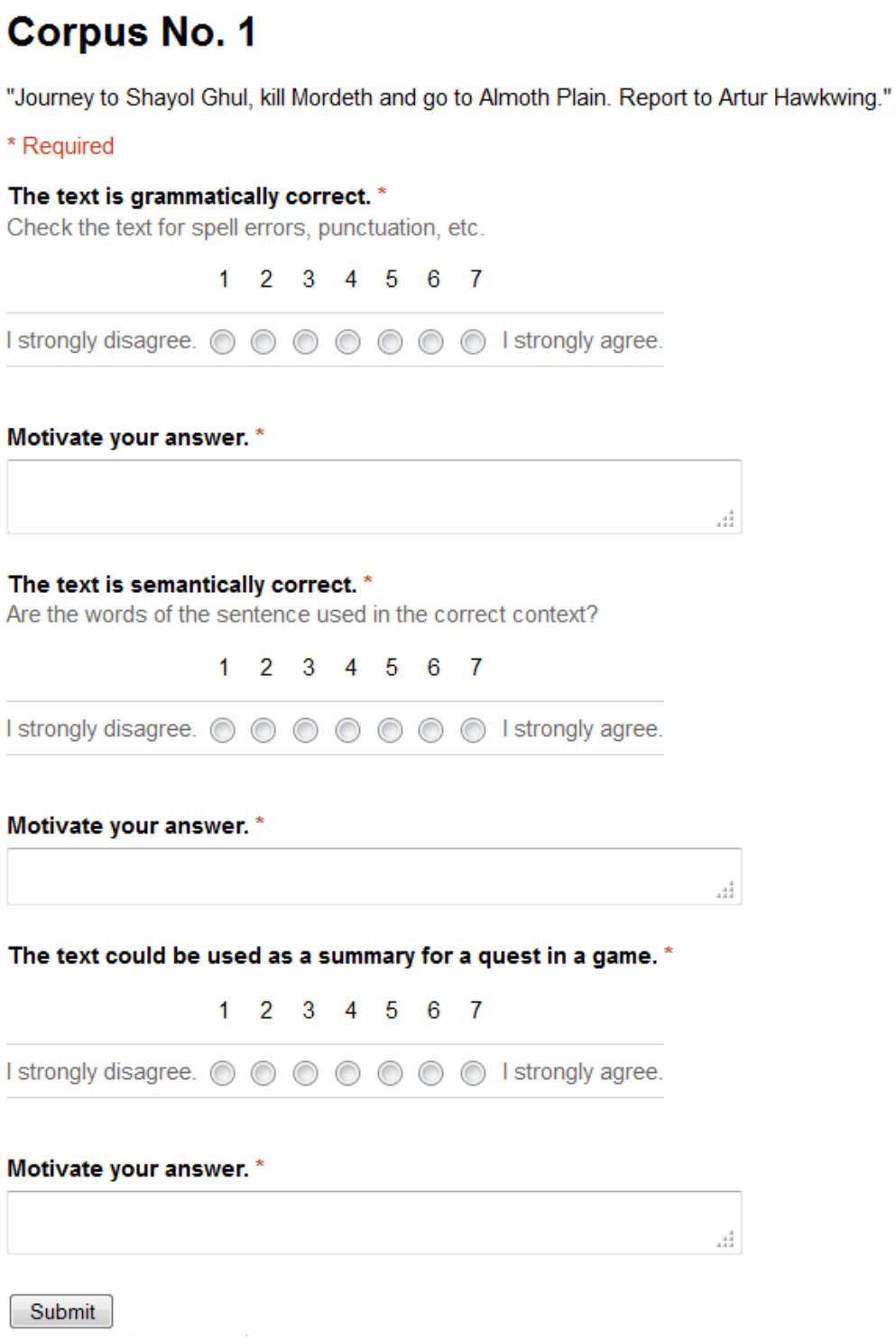

Figure 4.1: The provided form for the user case study.

Chapter 5

Results

In this chapter the results of the user case study is presented. The data is only presented here, the analysis is performed in the consecutive chapter.

5.1 Visualisation of Data

Each action sequence with its three corresponding resulting corpuses evaluated in the user case study is presented in a Table 5.1. Each action sequence is provided with an index and each index contains three different corpuses generated from the same action sequence. The corpus *Find the Horn of Valere and move to Illian. Give it to Mordeth*, for instance, can be viewed in the **A** row as the first corpus in Table 5.1 and is thus referenced as **Corpus A#1**.

Index	Sequence	Corpus #1	Corpus #2	Corpus #3
A	<get> <goto> give	Find the Horn of Valere and move to Illian. Give it to Mordeth.	Find the horn of Valere, travel to Andor and give it to Lan Mandragoran.	Find the Staff of Imprisonment, travel to Shayol Ghul and give it to Lew Therin Telamon.
B	<goto> listen <goto> report	Move to Cairhien, listen to Padan Fain and travel to Caemlyn. Report to Ba'alzamon.	Move to Emond's field, listen to Hurin and move to Fal Dara. Report to Hurin.	Move to Ghealdean. Listen to Agelmar Jagad and travel to Whitebridge. Report to Tam al'Thor.
C	<goto> <kill> <goto> report	Move to Saldaea. Kill Thom Merrilin. Go to Cairhien and report to Elyas Machera.	Go to Arafel, kill Loial and go to Baerlon. Report to Mordeth.	Journey to Shayol Ghul, kill Mordeth and go to Almoth Plain. Report to Artur Hawkwing.
D	<goto> use <goto> report	Journey to Tremalking. Use the Heron marked blade and go to Altara. Report to Tam al'Thor.	Go to Tar Valon, use the Half moon axe and report to Egwene al'Vere.	Travel to Tear, use the One Power and go to Andor. Report to Lan Mandragoran.
E	<get> <goto> use <capture> <goto> give	Obtain the Shield of Redemption. Travel to Tarabon, use it and capture Egwene al'Vere. Go to Tar Valon and give it to Lan Mandragoran.	Find the Half moon axe , travel to Altara, use it and capture Matrim Cauthon. Travel to Tarabon and give it to Ingtar.	Find the Staff of Imprisonment, go to Tremalking, use it and capture Lan Mandragoran. Journey to Almoth Plain and give it to Nynaeve al'Meara.
F	<goto> take <goto> give	Journey to Illian. Take the Blade of Sorcery and go to Whitebridge. Give it to Ingtar.	Travel to Fal Moran and take the Longbow. Journey to Emond's field and give it to Bayle Domon.	Journey to Altara, take the Dagger of Shadar Logoth and travel to Isle of the Sea Folk. Give it to Ingtar.
G	<goto> damage escort <goto> report	Journey to Isle of the Sea Folk. Damage Nynaeve al'Meara. Escort Ingtar, journey to Tarabon and report to Matrim Cauthon.	Travel to Ghealdean, damage Agelmar Jagad and escort Bayle Domon. Go to Tarabon and report to Loial.	Travel to Cairhien. Damage Ba'alzamon. Escort Loial and journey to Isle of the Sea Folk. Report to Tam al'Thor.
H	<goto> <steal> <goto> give	Move to Almoth Plain, steal the Quarterstaff of Ogier and go to Fal Dara. Give it to Perrin Aybara.	Travel to Caemlyn, steal the Dagger of Shadar Logoth and journey to Andor. Give it to Thom Merrilin.	Journey to Elmora. Steal the Dagger of Shadar Logoth, go to Andor and give it to Matrim Cauthon.
I	<goto> experiment	Travel to Tarabon and experiment the Fork of Unseeing on Tam al'Thor.	Journey to Fal Moran and experiment the Staff of Imprisonment on Lews Therin Telamon.	Journey to Illian and experiment the Fork of Unseeing on Moiraine Aes'sedai.
J	<goto> repair	Travel to Baerlon and repair the Disc of History.	Journey to Tremalking and repair the Ta'angreal.	Go to Cairhien and repair the Heron marked blade.

Table 5.1: All the action sequences and their corresponding resulting corpuses.

5.1.1 Collected Data

The data collected from the case study is presented in Table **??**. The corpuses generated from each action sequence is referenced by the index value and the Corpus number from Table 5.1. Each corpus was evaluated separately through a set of statements, where these statements are the three aspects of validation presented earlier. Each statement is presented with two values obtained from the case study. The values presented are average value and standard deviation for each aspect of validation. In Table **??** these values are denoted as **Avg**, and **SD**. These values corresponds to the acquired score the corpus was given by the participants in the respective aspect of validation. As an example the data presented for corpus *Find the Horn of Valere and move to Illian. Give it to Mordeth*, which is **Corpus a#1**, can be viewed under the column *Corpus* in the row called **A#1**.

	Grammatics		Semantics		Usability	
Corpus	Avg	SD	Avg	SD	Avg	SD
A #1	6.2	0.8	5.6	1.3	5.6	1.1
A #2	7	0.0	7	0.0	7	0.0
A #3	6.8	0.4	6.2	1.1	6.4	0.5
B #1	6.8	0.4	6	1.4	6	1.4
B #2	6	1.4	5.4	0.9	5.2	2.2
B #3	5.2	1.3	5	1.9	4.6	2.3
C #1	6.8	0.4	6	1.2	5.8	1.3
C #2	6.6	0.5	6.4	0.5	6	1.2
C #3	6.2	0.8	6.4	0.9	6.4	0.9
D #1	6.2	1.1	6.4	0.9	5.8	2.2
D #2	5.8	0.4	5.8	0.8	5.6	1.7
D #3	5.8	1.3	5.8	0.8	5.4	1.5
E #1	6	1.4	5.6	1.1	5.6	1.7
E #2	5.8	1.3	5.4	1.1	5.4	1.1
E #3	5.2	1.3	4.6	1.5	5.8	1.1
F #1	7	0.0	6.4	0.9	6.8	0.4
F #2	7	0.0	6.8	0.4	7	0.0
F #3	4	1.2	6.4	0.9	6.6	0.5
G #1	6.6	0.9	3.8	2.2	4	2.3
G #2	6.8	0.4	5.2	1.1	5.8	1.1
G #3	3.8	1.3	3.8	1.1	4.6	1.5
H #1	6.8	0.4	5.8	1.1	6.4	0.9
H #2	7	0.0	6.6	0.9	6.8	0.4
H #3	6	1.0	6.4	0.9	6.6	0.5
I #1	4.6	2.3	4	2.2	3.8	2.3
I #2	4.8	1.8	4.2	1.5	4.8	1.8
I #3	4.6	1.5	4.8	1.1	4.6	1.5
J #1	6.8	0.4	7	0.0	7	0.0
J #2	7	0.0	7	0.0	7	0.0
J #3	6.6	0.5	6.6	0.5	6.8	0.4

Table 5.2: Each corpus' obtained average value and its standard deviation.

5.1.2 Values

The values chosen for evaluation are the average values for each aspect of validation. The standard deviation for each aspect of validation is also presented in order to see the distribution of answers obtained in order to analyse the results easier. The decision behind standard deviation and average values are quite simple. The average value provides information on how a corpus performed in each aspect of validation in general. The standard deviation further adds information to the average values, which is of high interest since the analysis can be done more critically. A high standard deviation can potentially render an average score useless or simply inform when the participants were disagreeing on a certain aspect of validation. This information is crucial since it helps us evaluate how different the participants answered and thus provides us with the data to criticially judge their understanding of each statement. Lastly, the standard deviation also pinpoints corpuses where participants did not agree too well, thus suggesting that the corpus needs more work or needs to be analysed in more depth.

Chapter 6

Analysis

This section is discussing and evaluating the results of not only the user case study, but the research as a whole. A more thorough explanation and discussion is performed in the light of the specific objectives regarding problems, solutions, and the influence they had on the results.

6.1 Generate a Procedural Quest Structure

As described in chapter 3 this was done using Doran and Parberry's defined grammar structure for generation of quests where a decision was made to exclude *<subquest>* and *<QUEST>*. This exclusion prevented the action sequences to be too long which would result in a complex solution in order to generate appropriate natural language corresponding to the sequences.

6.2 Quest Corpus Generation System

This system has been the main implementational work undertaken during this research and its resulting corpuses were evaluated through a user case study as described in previous chapter. While the quest structure generator had long action sequences generated from Table 2.1 and Table 2.3 as output, only the sequences from Table 2.1 was used, i.e. the sequences representing the strategies. There are two reasons for this. The first one being that the grammar structure proposed by Doran and Parberry states what actions the player needs to do in order to fulfill the quest. Several of these actions are often actions that the quest giver does not know or care about, which means that these actions are generally not conveyed to the player by the NPC - which makes the natural language generation of these actions pointless.

Let us look at the action *<goto>* presented in Table 2.3. This action can be subdivided into another action sequence, e.g. *<learn> goto* as shown in the table. This indicates that to go somewhere the player first needs to learn where to go and then go there. However, the NPC will generally only instruct the player to go somewhere and not convey the information on how, or where this place is.

Of course there are situations where the NPC actually conveys this information, but it is not necessary from a gameplay point of view to always convey this information to the player.

The other reason is that to fully support the entire grammar of Doran and Parberry's research, it would require more time and effort than available for this research, hence the scope had to be narrowed down.

The influence these decisions had on the final corpus is that the corpuses are heavily summarized and concise, and since the aim of this research was to generate a summarized corpus from an underlying quest grammar structure, the decisions pointed the study in the right direction.

The choice of not handling motivations and strategies has also impacted the results in a subtle manner. Looking in Table 2.1 we can see that there are several action sequences that are exactly the same, but where the motivation and strategy data is the one thing telling their context apart. Since motivations and strategies was not handled there were action sequences that could not be tested since they appeared more than once and would obtain the same results as the one sequence that it is identical to. This narrowed down the available action sequences to test and to some extent there are quests that can not be expressed by the implemented system. This is mainly due to the lack of context though, and not the fact that an action sequence is not currently testable.

6.3 Quest Corpus Validation

The quest validation process is done by evaluating the results from the user case study. The analysing process of the results are done by starting with action sequences where the corpuses generated from the same sequence differs in score.

6.3.1 Evaluation of the First Action Sequence

The data shown in Table 6.3.1 shown below was obtained from action sequence *<get> <goto>* give. Looking at the average values for each corpus for this sequence we can clearly see that the values differs quite remarkably between them. The first corpus had an average of 6.2 in grammatics whereas the second corpus obtained the highest possible score, with the third corpus almost reaching the highest. Furthermore the average in both semantics and usability was quite low for the first corpus compared with the other two, despite being generated from the same action sequence. The standard deviation shows that the first corpus had the highest distribution of scores in each aspect, ranging from 0.8 to 1.3. Looking at the second corpus the participants were completely unanimous of the score, since the standard deviation is 0.0. The third corpus had a low standard deviation value as well, but was a bit higher in the aspect of semantics compared

to the other two aspects. Let us compare each quality separately.

	Grammatics		Semantics		Usability	
Corpus	Avg	SD	Avg	SD	Avg	SD
A #1	6.2	0.8	5.6	1.3	5.6	1.1
A #2	7	0.0	7	0.0	7	0.0
A #3	6.8	0.4	6.2	1.1	6.4	0.5

Table 6.1: The data obtained from the corpuses generated by *<get> <goto> give.*

Grammatics
The only aspect which grammatically differs between the corpuses are sentence aggregation. In the first corpus the action sequence is aggregated into two sentences in total, whereas the other two corpuses are aggregated into only one sentence, which can be viewed in Table 5.1.

None of the participants provided any suggestions for improvement of the grammatics, but rather argued that the chosen synonyms for the first corpus is what resulted in a bad rating. Synonyms are rather an evaluation topic of semantics rather than grammatics, which did not seem clear to several of the participants.

Semantics
Semantically every single participant suggested improvement on the synonym *move*, unless the corpus is suggesting that one should actually move your home somewhere. Generally this is quite uncommon since you, as a player, do not have a place of living in a game setting. The participants concluded that *move* was simply a synonym used for *going* somewhere in order to do the next task described in the corpus, thus making the synonym *move* wrongly used. Semantically this made the corpus confusing.

If we compare the three corpuses the only thing semantically telling them apart are the synonyms, places, and names used. Which further strengthens the theory that the synonym *move* is the culprit of this confusion, since none of the participants even remotely suggested that the names or places were the issue of the corpus.

Usability
Participants further stated that the questionable use of *move* breaks the illusion of the corpus and thus brings down the usability of the corpus. This would suggest that synonyms are of great importance when it comes to both semantics and usability in quest corpuses. Nevertheless, the average score in the aspect of

usability is high for both the second corpus and the third, with their respective standard deviation values low as well. The issue of synonyms is discussed in detail in the next section.

6.3.2 Synonyms

To further investigate the hypothesis of synonyms playing a huge role in the results of the individual corpuses let us investigate the corpuses of the other action sequences as well. Looking at the tables below, acquired from the previous chapter, we can see the presence of the synonym *move* in Table 6.2, 6.4, and 6.6. Looking at the corpuses of Table 6.2 each uses the synonym *move*.

Index	Sequence	Corpus #1	Corpus #2	Corpus #3
B	<goto> listen <goto> report	**Move** to Cairhien, listen to Padan Fain and travel to Caemlyn. Report to Ba'alzamon.	**Move** to Emond's field, listen to Hurin and **move** to Fal Dara. Report to Hurin.	**Move** to Ghealdean. Listen to Agelmar Jagad and travel to Whitebridge. Report to Tam al'Thor.

Table 6.2: The corpuses generated by the action sequence *<goto> listen <goto> report.*

Furthermore looking in Table 6.3 the average values of this action sequence is also spread out on the scale a bit. The first corpus of this action sequence had the best score in all three aspects, but the standard deviation values were quite high in both semantics and usability. The grammatics reach almost a top score, and was also high for the second corpus, while the third corpus obtained a much lower score. Overall the standard deviation shows that the distribution of scores given on the scale by the participants were high, especially in the aspect of usability.

The suggestions of improvement shows that 13 out of 15 participants stated that the synonym *move* made the corpus confusing. One participant also noticed that in the second corpus you are supposed to listen to an NPC, then go somewhere else to report to the same NPC, which could be illogical depending on the context of the quest given.

	Grammatics		Semantics		Usability	
Corpus	Avg	SD	Avg	SD	Avg	SD
B #1	6.8	0.4	6	1.4	6	1.4
B #2	6	1.4	5.4	0.9	5.2	2.2
B #3	5.2	1.3	5	1.9	4.6	2.3

Table 6.3: The data obtained from the corpuses generated by *<goto> listen <goto> report.*

Looking at Table 6.4 only the first corpus is using *move*, but the average score of the corpus is close to the average score of the others, which can be viewed in Table 6.5. The aspect of semantics was given a lower average score for the first corpus, and also the highest standard deviation value. The usability score was also the lowest for the first corpus compared to the others, where the standard deviation was also slightly higher for the first corpus.

The only suggestion of improvement given for the first corpus was to change *move* to another synonym, which was given by every participant.

Index	Sequence	Corpus #1	Corpus #2	Corpus #3
C	<goto> <kill> <goto> report	**Move** to Saldaea. Kill Thom Merrilin. Go to Cairhien and report to Elyas Machera.	Go to Arafel, kill Loial and go to Baerlon. Report to Mordeth.	Journey to Shayol Ghul, kill Mordeth and go to Almoth Plain. Report to Artur Hawkwing.

Table 6.4: The corpuses generated by the action sequence *<goto><kill> <goto> report.*

	Grammatics		Semantics		Usability	
Corpus	Avg	SD	Avg	SD	Avg	SD
C #1	6.8	0.4	6	1.2	5.8	1.3
C #2	6.6	0.5	6.4	0.5	6	1.2
C #3	6.2	0.8	6.4	0.9	6.4	0.9

Table 6.5: The data obtained from the corpuses generated by *<goto><kill> <goto> report.*

Looking at the corpuses shown in Table 6.6, there is one corpus that shows the synonym *move*. The test data for this action sequence is shown in Table 6.7 and shows great average values, where the first corpus, is given the worst score in

the aspects of semantics and usability. Every participant suggested the change of *move* here as well, which results in a lower score for semantics and usability.

Index	Sequence	Corpus #1	Corpus #2	Corpus #3
H	<goto> <steal> <goto> give	**Move** to Almoth Plain, steal the Quarterstaff of Ogier and go to Fal Dara. Give it to Perrin Aybara.	Travel to Caemlyn, steal the Dagger of Shadar Logoth and journey to Andor. Give it to Thom Merrilin.	Journey to Elmora. Steal the Dagger of Shadar Logoth, go to Andor and give it to Matrim Cauthon.

Table 6.6: The corpuses generated by the action sequence *<goto> <steal> <goto> give.*

	Grammatics		Semantics		Usability	
Corpus	Avg	SD	Avg	SD	Avg	SD
H #1	6.8	0.4	5.8	1.1	6.4	0.9
H #2	7	0.0	6.6	0.9	6.8	0.4
H #3	6	1.0	6.4	0.9	6.6	0.5

Table 6.7: The data obtained from the corpuses generated by *<goto> <steal> <goto> give.*

The corpuses generated from the sequence *<goto> damage escort <goto> report* shown in Table 6.8 below also had uneven average values in comparison, which can be viewed in Table 6.9. In each of the corpuses the participants answered that the word *damage* did not make any sense in the context it is used, even if it is grammatically correct. Suggestions were made to either rewrite the entire sentence, or to change synonyms and use *wound* or any other synonym more suiting. A general opinion was also that the word *escort* further added to the confusion of the corpus since it did not say where you were supposed to escort the referred NPC, suggesting a gap of missing information.

Index	Sequence	Corpus #1	Corpus #2	Corpus #3
G	<goto> damage escort <goto> report	Journey to Isle of the Sea Folk. **Damage** Nynaeve al'Meara. **Escort** Ingtar, journey to Tarabon and report to Matrim Cauthon.	Travel to Ghealdean, **damage** Agelmar Jagad and **escort** Bayle Domon. Go to Tarabon and report to Loial.	Travel to Cairhien. **Damage** Ba'alzamon. **Escort** Loial and journey to Isle of the Sea Folk. Report to Tam al'Thor.

Table 6.8: The corpuses generated by the action sequence *<goto> damage escort <goto> report.*

	Grammatics		Semantics		Usability	
Corpus	Avg	SD	Avg	SD	Avg	SD
G #1	6.6	0.9	3.8	2.2	4	2.3
G #2	6.8	0.4	5.2	1.1	5.8	1.1
G #3	3.8	1.3	3.8	1.1	4.6	1.5

Table 6.9: The data obtained from the corpuses generated by *<goto> damage escort <goto> report.*

The interesting data to look at here is that the first and third corpus had terrible average scores in the aspects of semantics and usability, while the second corpus had remarkably higher scores. Looking at the standard deviation, the second corpus also had the lowest distributions of scores. The second corpus seem to have a better sentence aggregation which probably provides a better flow when reading, but the reason for this remarkable difference in score is hard to pinpoint.

6.3.3 Sentence Aggregation

In several of the corpuses participants suggested that the sentence aggregation should have been done in another way. The third corpus in Table 6.8 was one of the worst corpuses in the aspect of sentence aggregation. The vast majority of participants felt that the corpus was written more like a todo-list, which made the corpus lose its flow and resulted in that it felt like each sentence did not necessarily have any connection to any other sentence in the corpus. In fact a few even felt that the corpus could just as easily have been two or several quests. In general the quest corpus generated for this sequence had major issues with synonyms which could have been one contributing reason to the segregation of the coherency of the corpus. The last corpus in particular further pushed this incoherence due to poor sentence aggregation.

In general there were suggested improvements addressing sentence aggregation scattered around several of the action sequences and their generated corpuses, but was not the common opinion in any of the corpuses. It is thus hard to pinpoint exactly in what situation sentence aggregation should be improved, other than when an action sequence's every action is provided with its own sentence, as in the third corpus of Table 6.8 described above.

6.3.4 The Worst Action Sequence

The overall worst score given to corpuses was provided by the action sequence *<goto> experiment* and is shown in Table 6.10. The issue with each of the

corpuses generated by this action sequence was that the natural generation system did not generate the word *experiment* with correct prepositions. This resulted in that each corpus was generated grammatically incorrect, and this also had a huge affect on the semantics of the entire corpus. Participants suggested two different solutions where the first one was to simply change the *experiment* with another more fitting synonym, i.e. *try*. The other solution was proposed by the majority and was to add the preposition *with* after *experiment*. Since *experiment* was so poorly used the corpus was rendered useless by most of the participants, and thus obtained low score in every category for each corpus. The scores provided for this action sequence can be viewed in Table 6.11.

Index	Sequence	Corpus #1	Corpus #2	Corpus #3
I	<goto> experiment	Travel to Tarabon and **experiment** the Fork of Unseeing on Tam al'Thor.	Journey to Fal Moran and **experiment** the Staff of Imprisonment on Lews Therin Telamon.	Journey to Illian and **experiment** the Fork of Unseeing on Moiraine Aes'sedai.

Table 6.10: The corpuses generated by the action sequence *<goto> experiment.*

	Grammatics		Semantics		Usability	
Corpus	Avg	SD	Avg	SD	Avg	SD
I #1	4.6	2.3	4	2.2	3.8	2.3
I #2	4.8	1.8	4.2	1.5	4.8	1.8
I #3	4.6	1.5	4.8	1.1	4.6	1.5

Table 6.11: The data obtained from the corpuses generated by *<goto> experiment.*

Something worthy of note is that the standard deviation values are high in each corpus, and in each aspect of validation. Since the standard deviation values were so high, the average value is rendered useless as a metric. The only thing that can be taken into consideration from these values are that the participants did not agree on the level of severity the grammatical incorrectness had on the corpuses as a whole. This could even suggest that one, or several, participants did not even notice the grammatical error.

6.3.5 Other Suggested Improvements

There were a few suggestions for improvements scattered around different corpuses as well, e.g. some participants stated that it was hard to know what places or persons the names provided in the corpus was referring to. They did, however,

realise that this kind of information is rarely conveyed in a quest description in an actual game. When a player plays a game it is implied in the context of the game itself that the player knows a certain place or NPC.

Another issue was that there was no motivation presented to why a certain task was to be performed. The motivation is rarely not conveyed in the quest description, but seeing as the generated corpuses evaluated in this research are merely a summarized set of actions the quest wants to be performed, the motivational information is more suited to be conveyed in a longer description, thus it can be assumed that the motivation for the quest is described elsewhere.

6.4 Validity Threats

There are certain aspects of this research that has not been taken into deeper consideration which could have affected the results. The specific areas are explained below as its own section.

6.4.1 Grammatical Knowledge of Participants

The results from the varying tables presented in earlier sections show that there are a occasions where the grammatical score of corpuses are varying between participants. There was no method presented in this research to ensure the grammatical knowledge of the participants. The grammatical ensurement was done by forcing the participants to motivate their choice of rating in the aspects of validation, thus being forced to provide their grammatical knowledge when rating that particular aspect. This was, however, not taken under consideration when presenting the results, but was taken under consideration when analysing the data for action sequences where the data differed among its corpuses, since this is not the primary evaluation value sought after in this research.

6.4.2 Repetitive Corpuses

Looking at the corpuses one can argue that they are a bit repetitive. This is simply because the generation of text is only using the basic grammar structure proposed by Doran and Parberry. This basic grammar structure does not contain any other data than the actions to be performed, which limits the possibilities for variation. This is a potential issue, which needs to be taken under consideration. In this study changing synonyms and names of entities has been a measure to try and lower the repetitive pattern in the corpuses.

6.4.3 Setup of the Case Study

Each participant of the case study was given the same corpuses for evaluation as every other participant in his/hers assigned group. The corpuses were also presented to each participant in order. This could have affected the results since the corpuses are of different length, grammatical quality, and so on. If a participant was shown the worst generated corpus first, his or hers point of view would probably have been different compared to be shown a good one first instead. This is hard to analyse in hindsight, and if this had a negative or positive (or any) effect on the results is unclear.

Chapter 7

Conclusions and Future Work

The following two sections are the closing sections for this paper. The first concluding the analysis of the results and the second identifying future work in the direction of the topic presented in this research.

7.1 Conclusion

This study aimed to answer if natural language can be generated in conjunction with a procedurally generated quest structure to produce the descriptive text for a quest, referenced in this paper as quest corpus. Furthermore the question whether the quest corpus reach a high enough quality to be used in game development is also to be answered.

The general results show that there are indeed potential in combining natural language generation with procedural content generation in order to generate more complex quests for interactive games. The focus in this research was to generate a summarized version of a quest description, and thus the results of this research is depended upon there being a more thorough description of the quest tested. The combination of natural language generation, procedural content generation, for quests seems to be a viable approach and would benefit from further research.

The focus of the tests were to see if the summarized quest corpus could be used in an interactive game, which was not the research question to be answered. The reason for this twist of the question was simply because the participants of the case study did not have experience with the process of game development, but rather with the process of playing the games where such quests are often used. If the results indicate that the quality of the corpuses are high enough to satisfy its potential customers, it must also be high enough for use in the development of such a game.

In general the data collected from the case study, and the analysis performed on this data, suggests that corpuses can be used to some extent as a summarized version of a quest description in an actual interactive game. There were several action sequences where the result scores were either the highest obtainable, or

close to the highest obtainable. At the same time there were some questionable action sequences that did not work at all, or action sequences where the results were mediocre.

It is hard to tell if the corpuses reached a high enough quality to be used in game development. From the results we see that there are several corpuses that obtain the highest available score, or close to the highest available, which would indicate that they are good enough to at least be used in development to some extent. The combination of natural language generation and procedural content generation can probably be used for early prototype of quests, provide the base structure for a quest, or to be used to generate template quests to test functionality of quest systems implemented for the game at hand. To use the summarized corpuses in a game directly, as they are now and with no longer description available, is not possible.

To fully answer if the quality of the corpuses were high enough to be used in game development there needs to be more tests performed, and the current natural language generation system implemented for this purpose must be implemented more carefully to provide support for the two broken action sequences. To further implement more varying data structures and more creative corpuses as a whole, would require a lot of work in several areas and would be complex. In the next section some of the areas that needs to be worked upon to reach such a complex system is proposed.

7.2 Future Work

The most prominent future work to be done is to improve the implemented system presented in this research and perform more tests on a much larger scale. The improvements to be done are many, but could be to add a thorough classification or labeling of atomic actions, which was touched upon in this paper. Each atomic action can refer to any number of entities and there is, as of now, no way of telling them apart from each other. This is obviously a problem when generating corpuses procedurally where the referring entities are conveying the basis of information to the player. This labeling process can be done by identifying the different entities, and their relations, each atomic action refers to.

Defining each motivation, and its corresponding strategies, to convey more explicit data to each action sequence can provide the corpus with more diversity and, more importantly, a context to which the sequence can be customized to. Such a definition could potentially provide a more detailed quest corpus for each action sequence, and where there are significant differences between action sequences are identical, thus providing a wider array of usable action sequences than provided in this study. This will also help counter the potential issue of repetitive corpuses.

Another approach to further expand this area is to attempt to generate the more thorough quest description. Just as the quest grammar structure used is the foundation for this summarized description of the quest, this summarized version could be used as the foundation for the more descriptive version.

It could also be of interest to perform research on exactly what kind of, and how much, information is needed to be conveyed to the player in the context of interactive games in order to provide a more precise definition on what needs to be generated when it comes to PCG and NLG for quests. Sentence aggregation also seemed to be of high value when it comes to generate the flow of a corpus. A study to see exactly how much this affects and contributes to corpuses for quests in interactive games can further define the generation of quests.

Bibliography

ACM, . ACM Computing Classification System toc, 2012. URL `http://dl.acm.org/ccs\_flat.cfm`.

Ahmad, S. A Natural Language Processing Tutorial. *Language*, 810, 2007. doi: 10.1.1.100.4660.

Belz, A. and Reiter, E. Comparing Automatic and Human Evaluation of NLG Systems. *EACL*, pages 313–320.

Berndtsson, M.; Hansson, J.; Olsson, B., and Lundell, B. *Thesis Projects - A Guide for Students in Computer Science and Information Systems*. Springer, second edi edition, 2008. ISBN 9781848000087.

Callison-Burch, C.; Osborne, M., and Koehn, P. Re-evaluating the Role of BLEU in Machine Translation Research. In *Proc. 11th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2006*, pages 249–256, 2006. ISBN 1932432590. doi: 10.1145/1083784.1083789. URL `http://www.iccs.inf.ed.ac.uk/~osborne/papers/eacl06.pdf`.

Desel, J. and Juhás, G. "what is a petri net?" informal answers for the informed reader. In Ehrig, H.; Padberg, J.; Juhás, G., and Rozenberg, G., editors, *Unifying Petri Nets*, volume 2128 of *Lecture Notes in Computer Science*, pages 1–25. Springer Berlin Heidelberg, 2001. ISBN 978-3-540-43067-4. doi: 10.1007/3-540-45541-8_1.

Doran, J. and Parberry, I. 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, pages 1:1–1:8, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0872-4. doi: 10.1145/2000919.2000920.

Ferrucci, D. A. Introduction to "this is watson". *IBM Journal of Research and Development*, 56(3):1, 2012.

Galliers, J.R. and Jones, K. Sparck. Evaluating natural language processing systems, 1993.

Goldberg, E.; Driedger, N., and Kittredge, R.I. Using natural-language processing to produce weather forecasts. *IEEE Expert*, 9(2):45–53, April 1994. ISSN 0885-9000. doi: 10.1109/64.294135.

IBM, . IBM Watson Ecosystem Program, 2014. URL `http://www-03.ibm.com/innovation/us/watson/`.

Jamieson, S. Likert scales: how to (ab)use them. *Medical education*, 38(12):1217–8, December 2004. ISSN 0308-0110. doi: 10.1111/j.1365-2929.2004.02012.x.

Lee, Y-S. and Cho, S-B. Dynamic quest plot generation using petri net planning. In *Proceedings of the Workshop at SIGGRAPH Asia*, WASA '12, pages 47–52, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1835-8. doi: 10.1145/2425296.2425304.

Mateas, M. and Stern, A. Façade: An experiment in building a fully-realized interactive drama. *Game Developers Conference*, 2, 2003.

Mellish, C. and Dale, R. Evaluation in the context of natural language generation. *Computer Speech & Language*, 12(4):349–373, October 1998. ISSN 08852308. doi: 10.1006/csla.1998.0106.

Miller, G. A.; Beckwith, R.; Fellbaum, C.; Gross, D., and Miller, K. J. Introduction to wordnet: An on-line lexical database*. *International Journal of Lexicography*, 3(4):235–244, 1990. doi: 10.1093/ijl/3.4.235.

Nunnally, JC. C. and Bernstein, I. *Psychometric Theory*. ISBN 0070474656. doi: 10.1037/018882.

Papineni, K.; Roukos, S.; Ward, T., and Zhu, W-J. BLEU: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics - ACL '02*, pages 311–318, 2001. doi: 10.3115/1073083.1073135.

Reiter, E. and Dale, R. Building applied natural language generation systems. *Journal of Natural Language Engineering*, 3:57–87, 1997. ISSN 13513249. doi: 10.1017/S1351324997001502.

Strong, C.; Mehta, M., and Mishra, K. Emotionally driven natural language generation for personality rich characters in interactive games. *In proceeding of: Proceedings of the Third Artificial Intelligence and Interactive Digital Entertainment Conference*, pages 98–100, 2007.

TURING, A. M. I.—computing machinery and intelligence. *Mind*, LIX(236):433–460, 1950. doi: 10.1093/mind/LIX.236.433.