DEGREE PROJECT, IN COMPUTER SCIENCE , FIRST LEVEL
STOCKHOLM, SWEDEN 2015

Natural Language Interfaces in Computer Games

A STUDY OF USER AND COMPUTER-GENERATED INPUT IN A TEXT-BASED GAME USING A NATURAL LANGUAGE INTERFACE

MARK HOBRO AND MARCUS HEINE

KTH ROYAL INSTITUTE OF TECHNOLOGY

CSC SCHOOL

KTH Computer Science and Communication

Natural Language Interfaces in Computer Games

A study of user and computer-generated input in a text-based game using a natural language interface

MARCUS HEINE - mheine@kth.se
MARK HOBRO - mhobro@kth.se

Degree Project in Computer Science - DD143X
Supervisor: Richard Glassey
Examiner: Örjan Ekeberg

CSC, KTH 2015-04-20

Abstract

Natural language processing is a complex area of computer science which has been under discussion for more than forty years. During recent years natural language interfaces have been established in conjunction with speech recognition. This report will cover the theory behind natural language processing and evaluate the weaknesses and strengths of implementing and using a natural language interface in a text-based game environment using the Natural Language Toolkit for Python. The results show that the Natural Language Toolkit has great potential for implementing a natural language interface for a text-based game, but the library alone is not sufficient to get good results when the scope of language is increased.

Referat

Natural language processing är ett komplext område inom datalogin, som det har forskats om i mer än fyrtio år. På senare år har natural language interfaces tagit större plats i samhället i och med att röststyrningsteknologin har utvecklats. Denna rapport går in på teorin som ligger till grund för natural language processing och reflekterar över styrkor och svagheter i både implementation och användande av ett natural language interface i en textbaserad spelmiljö. Detta görs med det språkteknologiska biblioteket Natural Language Toolkit till Python. Resultatet visar att Natural Language Toolkit har stor potential för att implementera natural language interfaces för textbaserade spel.

Contents

Chapter 1

Introduction

Relying on a computer to complete a task without any errors is inherently difficult. When a user's ability to express its intentions is crucial for an interface between user and computer, it immediately becomes difficult to develop a system with high precision. A Natural Language Interface (NLI) is an interface with the purpose of interpreting text written in a natural language, evaluate this data and then perform an appropriate action.

In the 1980's, computer games based entirely on text (text-based games), rose to popularity. Due to the difficulty of developing a game based entirely on natural language input, predefined commands were used to narrow down the variety of the input and simplify the task. The idea of using a well optimized NLI is that the user can freely write what functionality they want the game to perform, as far as it makes sense in the scope of the game or application. This means that the user should be able to express his or her intention using a natural, everyday language.

NLI's are today a highly relevant subject. Direct speech and text communication, where the computer has to take complex decisions out of sometimes ambiguous input, is used widely in electronic devices. The focus of this text will be on the parsing and interpretation of text, which can be viewed as a slighter easier task compared to interpreting text that has been converted from speech.

1.1 Motivation

Natural language processing has been a well discussed subject the last century and it had its prime during the 1980's with the growing computer game industry. After that, the research of natural language processing had a period where the discussions were less frequent, mainly because of the complexity of language technology and finding a place for it in society. Since the start of the 21st century NLI's have become more interesting again, mainly because of the integration of speech recognition in computers and other devices. It is therefore interesting to further investigate the technology that is available today and research how well it performs.

1.2 Problem Statement

The main goal of this study is to evaluate to which extent it is possible to interpret natural language in a small application, a text-based game, and get results that comply with the user input. The rules of English grammar are complex, and it is of interest to see how an NLI would handle inputs to a game. The Natural Language Toolkit (NLTK) for Python is a widely used tool, and one of the goals is to see how it will perform in a game and whether it has the necessary functions to complete the appropriate tasks. The second part of the study is regarding users of text-based games. How accurate are their commands? What types of sentences are used? What are their general opinions on text-based games?

Chapter 2

Background

This report will focus on studying natural languages and the use of natural language in computer games. In order to do this, a thorough research of language was carried out to understand the logic and the linguistic behind the language.

The modern computer has been under continuous development and has evolved during the last 70 years. Since the beginning of the 1940's computers have gone from doing simple arithmetic to performing billions of calculations per second [1]. Over the course of the next three decades computers were greatly improved, and in 1976 a text-based game called Colossal Cave Adventure (CCA) was released by Will Crowther. It is a simple game, written in FORTRAN and is played in a terminal. CCA leads the user on an "adventure" based on the inputs. The user would enter either one or two words and the parser that the game uses looks at the first five letters of each word and determines a course of action based on what it receives [2].

This is where Natural Language Technology becomes relevant. In the original CCA, Crowther wrote approximately 1400 lines of code, and among those, 193 vocabulary words. This is comparable to a database of strings, and CCA would then compare the input to the words in the database. When this program was written, no major thought was given to handling errors [3]; if the wrong word was typed, the program would reply that it could not parse the input. CCA was designed to run on an early and primitive computer system, and therefore it was not possible to include any larger wordlists.

Since then, technology has improved in a number of different areas. The point of interest in this report is the way humans program computers to parse input generated by users. A recent example of this would be in 2011, when Apple Inc. released the mobile phone iPhone 4s. The flagship feature was the introduction of Siri, a responsive interface created to answer questions and perform certain actions on the phone. Siri uses voice recognition via an NLI to interpret user input. Siri can be divided into two major components, the parsing of the input and the interpretation of said input. The actual handling of the input is not of major importance in this report, since it is done by voice recognition. The focus in this report concerns how Siri and other similar software handle the input they receive. Fundamentally, Siri works similarly to most other parsing services; it tries to find the best suitable match based on stored keywords, and uses these calculations to perform the most probable task [4]. A major feature of Siri is the "correcting" feature; if Siri is unsure about the correct course of action, it will attempt to gain more information by prefacing the reply with "Did you mean", in an attempt to confirm the users intentions. This feature was a big step towards fully understanding what the user really meant.

Another recent use of this technology surfaced in the form of Façade [5], an interactive story posing as a game, which was released in 2006. Façade quickly gained international recognition, partly because of simple graphics, but mostly because of it's unique gameplay. The developers stated that "a fully-realized interactive drama has not yet been built", and they wanted to "integrate believable characters, natural language conversation, and dynamic storyline, into a small but complete [and] playable [...] experience" [5]. Façade uses a number of technologies to power the game, and the one that is of interest is the use of ABL (A Behaviour Language).

ABL is a programming language that was invented with the intention of fully supporting "the creation of life-like computer characters", so-called believable agents. Compared to the "standard" imperative programming languages such as Java or C++, ABL makes it simple for the programmer to specify how the agent, or character, should act and react. The response of the agents can easily be altered based on external input, which means that an agent can "perform local, context-specific reactions to a player's actions" [6].

2.1 Natural Language Processing

Natural language processing is the theory and practice behind the translation of natural language to a language that computers can understand. The research of natural language in a mathematical manner started taking place in the early 1900's [8]. At first the focus was to construct mathematical reasoning using logic. Three later developments have laid a foundation to what natural language processing is today. These three developments are formal language theory, symbolic logic and the principle of compositionality [8].

2.1.1 Theoretic foundation of language technology

Formal language theory defines a language consisting of a set of symbols, also called an alphabet, which under a set of rules defines a formal language [9]. As an example, the natural numbers can be represented as follows:

$$A = \{0,1,2,3,4,5,6,7,8,9,+,=\}$$

A language L defines a set of rules to an alphabet which together defines a formal language [9]. In this case, addition and the equals sign is in the alphabet A, since every set of natural numbers can be added up to a greater value. The formal language for the natural numbers is defined as L and consists of the set of rules below to meet the requirements of the language.

- All the nonempty strings of the alphabet that does not start with "0" or contains "+" or "=" is in L.
- A string only containing "0" is in L and could be considered an empty string.
- A string is in L if it contains exactly one "=" which separates two valid strings that is in L.
- A string is in L if it contains "+", but not "=", and all the "+" separates two valid strings that is in L.

This type of symbolic logic introduced a formal way to complete logical proofs, i.e propositional logic. Having a formal calculus language makes it possible to associate the meaning of natural language in a calculus formula by translating the natural language into the syntax of symbolic logic [8].

The dog catches the ball.

The sentence above can be translated into the relation:

catch(dog, ball).

The verb catch/catches is interpreted as the relation and dog/ball is the subject/object. Symbolic logic is a good way to interpret natural language in a computational manner. It is also important to have in mind that the decomposition steps from natural language to a logical language can be complex.

The principle of compositionality is a concept that states that every expression or sentence is a direct composition of its containing words or symbols, and how these parts of the expressions relate to one another. This principle gives a correspondence between the semantics and syntax of a sentence, phrase or expression. This means that it is possible to translate a natural language expression into symbolic logic and compute it recursively [8]. If there is a relation which says that "It is not true that ("expression")", that translates it into not("expression"). Negating the example above can then be written as:

not(catch(dog, ball)).

This expression, which in natural language could be written as "it is not true that the dog catches the ball", is here written in symbolic logic.

These three developments have built a foundation of what natural language processing is today. The developments were a good starting point for the research of natural language processing and a basis that different scholars could agree upon [8].

2.1.2 Levels of linguistics

It is possible to divide natural language processing into several levels of linguistic, since natural language is versatile and can be split into different parts. Humans use every aspect of the linguistic levels to understand a language, but the interpretation process is difficult for computers. A more capable NLI would utilize aspects from several of the levels of language mentioned below, whereas an interpreter that only focuses on one area of linguistics could be successful for a specific purpose [10].

Phonology

The phonology level focuses on the interpretation of human speech. When recognizing human speech an interpreter would first of all want to look at phonetic rules, which is sounds within words (how a person would pronounce words or letters). Secondly the interpreter should look at phonemic rules. These rules describe pronunciation differences when words are spoken together. Lastly the interpreter should look at prosodic rules, which are variations in stress across a sentence [10].

Morphology

The morphology level handles the composition of words. Words can be decomposed into morphemes, which are words that cannot be decomposed into a smaller entity and still have a meaning. For example, take the word "packed". This can be decomposed into two morphemes, "pack" and the suffix "ed". Suffixes and prefixes are also morphemes [10].

Lexical

The lexical level looks at words individually. Lexical analysis is one of the most common methods used when creating an NLI. For example, the Natural Language Toolkit for Python has a way to tag words with a range of different grammatical determiners (see Appendix A). This technology is still flawed, because there are words that have more than one sense or meaning. A dictionary could be a good addition to the tagging of words to solve the case with ambiguous words. Otherwise the usual approach is to implementing the more advanced linguistic layers mentioned below [10].

Syntactic

The focus in the syntactic level is on analyzing whole sentences by investigating the grammatical contexture with the goal to ascertain the context of it. In an NLI it is common to implement decomposition of a sentence and extracting the valuable words that define the context. It is not only crucial to be able to extract the right words, but also to be able to interpret the sentence structure correctly [10].

Semantic

The semantic level is the interpretation of sentences by looking at dependencies between words, meaning looking at words or phrases which describe the subject. To complement the problem with ambiguous words that the lexical level does not cover, the semantic level can disambiguate words in its interpretation process. This can be done by statistically looking at a word in other contexts and evaluating the result [10].

Discourse

The discourse layer handles text longer than sentences. The length varies but most commonly this level analyzes a set of components and evaluates the global context by looking at dependencies. This can be done by investigating pronouns and to which entity they refer [10].

Pragmatic

This is the most complex level which looks at ambiguous context of sentences. Some phrases is even impossible for us humans to understand, if they are built in a way that creates a double-edged meaning. Anaphoric terms, such as "they", can be used in a way in which it is possible that it points back to two different words and it is difficult to evaluate which word "they" is supposed to point back to [10].

2.2 Natural Language Interpretation

Taking the underlying theoretics mentioned in section 2.1 into consideration will help to evaluate what parts of language to focus on when implementing an NLI. The goal of programming an NLI is to get an understanding of the language by simplifying the context and trying to find patterns in short phrases or sentences [7]. In a text-based computer game, the player wants to turn text into an action. The interpreter's task is to examine and parse the text so that it becomes possible to evaluate which action the player wants to accomplish. To accomplish this, the linguistics in section 2.1 defines how text should be evaluated in a computational manner. This section covers concrete examples of how the linguistics applies to phrases that could relate to situations in a text-based game.

2.2.1 Disambiguate words and interpret context

The area of potential actions in a text-based game is quite narrow compared to, for example, Apple Inc.'s Siri, which has a larger scope of phrases to evaluate. In the case of a text-based game, the words that could be considered ambiguous will be easier to evaluate since the context of the phrase is more predictable (the computer is aware of the context scope of the input). Many words are ambiguous, therefore it could be crucial to find several words that fit into the same context [7].

As an example, consider the two words "pass" and "play". "Pass" and "play" are difficult to interpret if they exist alone in a short sentence with the absence of a clear context. Compare these two sentences:

- "The pass was the start of the play"
- "That pass from the centre back's right foot was the start of an incredible play by the Tigers and made the spectators go crazy"

The first sentence is not that clear and is, even for a human, difficult to understand. One thing that could lead a computer in the right direction to interpret this sentence is to look at semantics and possible contexts that the words fit in. The other sentence gives more information to help with the evaluation of its context. A human should understand the context of the sentence and come to the conclusion that it is about sports. A text parser is given more information and words, than in the first case, that can be bound to the verbs. Adding rules to the parser will make it possible to extract the words of interest and analyse the context correctly. Parsing ambiguous words is difficult but can be done by looking at semantics and adjacent words, but if one word has more than one meaning, the evaluation immediately turns into a more complex task [7].

Continuing on this approach it is possible to look at words that fit into a defined context. An example of this is conjunctions, and it is common that they in a sentence are followed by a word that explains a location or has an agentive or temporal meaning. The interpreter can get valuable information about the context of the input by evaluating succeeding words of conjunctions. An example is the word "by". It can be used in various ways, for example:

The keys were located by the dog.

"The dog" has an agentive meaning. The words succeeding "by", in this case "the dog", is the grammatical agent that performs the action of the verb. Extracting this word gives a hint of the context of the sentence, since knowing the subject of a phrase is helpful. An example of a sentence with a locative subject is:

The keys were located by the desk.

"The desk" explains the location of the event. As a last example, a temporal meaning:

The keys were located by sunset.

"Sunset" explains when the event took place. Finding the subjects of the sentences is not a difficult task, and it would be of great use if it is possible to bind it together with the object that gives the event its meaning. Finding both objects and subjects of a verb is a more linguistic approach than the aforementioned methods.

In a game the subject is usually the player itself, and detecting the objects and possibly what kind of action the subject is performing, is important in a text-based game. It is also an important part of the parsing, since this type of input is all about different events taking place in the scope of the game. However, finding the objects and subjects and binding them together can be difficult [7]. As an example:

The scientists approached the animals.
They eventually left the area.

The second sentence is ambiguous and difficult to interpret. The problem here is to figure out which object the pronoun is referring to and in this case that is impossible because of the verb in the second sentence being suited for both the scientists and the animals. Fortunately text-based games are usually turn based, so the player/players execute one task at a time. Since this report will focus on text-based games, played from a first-person perspective, this kind of text parsing will not be of great importance [7].

2.3 Natural Language Toolkit

Most of the examples mentioned in section 2.2 can be decomposed and evaluated using natural language software. For this project the Natural Language Toolkit for Python has been chosen as the language parser.

NLTK was used in the study for three reasons, many previous similar studies have used both NLTK and Python with success, extensive documentation for the library is available, and because it was recommended by the supervisor of this project.

One of the most important parts of Natural Language Processing and what NLTK is used for is to recognize and analyze patterns [11]. An example of a pattern is the recognizable prefix "re-", and when encountered it can be assumed that the verb is something that has happened before, and is now happening again. The grammatical rules may seem trite to English-speaking people; humans are taught the rules and the exceptions at an early age, and from that point on it comes almost by instinct [12]. For computers it is not that simple, the rules need to be clarified and written down in a way that enables the computer to classify words and text. This is where the Natural Language Toolkit (NLTK) for Python comes in.

The idea for the NLTK project was thought of in 2001 by Steven Bird, an Associate Director at the University of Pennsylvania, and one of his students, Edward Loper. Loper wrote a plan for developing a software infrastructure for natural language processing, and both of them started working on it from that point onwards. Since then, NLTK has been under continuous development and is now "a suite of open source Python modules, data sets and tutorials supporting research and development in Natural Language Processing." [13]

One of the most frequently used features of NLTK is word tagging. What this does is that it attaches a "tag" to a word, which describes what "word class" the word belongs to. The official book on NLTK describes word tagging as "the process of classifying words into their parts of speech and labeling them accordingly is known as part-of-speech tagging." [11]. Appendix A describes the majority of the labels used by NLTK.

Chapter 3

Method

This section will cover the implementation of the NLI and how it relates to the theory behind natural language processing. The knowledge behind the code written relies on a thorough literature study.

The method is split into a number of steps, being the creation of the game, the parsing of user input and the gathering of the testing data.

3.1 Thoughts and hypothesis

From our initial research we have gathered enough information to build an early opinion of what this project might lead to. We believe that it is possible to design an NLI that performs well in a text-based game environment. This is mainly the case because of the variety of possible input that should be handled is limited. With a big dataset of possible input, such as Siri's case, it is complex if not impossible today to build an interpreter with perfect accuracy. We also believe that it is possible to create an automated dataset, by creating user-like sentences, which could be used to test the accuracy of the NLI.

3.2 Gameplay

A simple adventure game was created, based on Phillip Johnson's game titled "AdventureTutorial" [13]. It is a relatively simple game programmed using Python 3. The gameplay in itself is basic, the user find the character in a fictitious cave with exits in up to four directions. There are a number of different rooms to which the user can go, and one of the rooms has an exit that leads to safety, which is the victory condition. One room in the cave has an exit to a "pit of deadly snakes", entering this room is the failure condition.

In the rooms the user can find different types of monsters, such as ogres and spiders; weapons, such as rocks and daggers; and items, such as coins. The aspect that makes the game simple is that the user can in theory complete the game in 5 commands. The victory condition does not depend in any way on how many coins the user has or how many enemies have been defeated. The only condition needed to win is to go in the correct direction.

3.3 Input parsing

In order to start collecting and analyzing input, the NLTK library was installed. The most important components used were the NLTK tagger and model, to analyze the input, and the WordNet Lemmatizer, to ensure all input was translated to the correct tense and the correct part of the sentence. The input was made into lowercase letters and punctuation was disregarded.

All commands entered by the user are passed to the parser, which is where the main use of NLTK occurs. The parser tags all words in the input sentence with a "word class tag", and gives a response based on the words.

Listing 3.1. Excerpt from Parser.py

```
def classifyWords(listOfWords):
  for index, (word, tag) in enumerate(listOfWords):

    if tag.startswith('N'): #Word is tagged as a noun
      parseNoun(word)

    elif tag.startswith('VB'): #Word is tagged as a verb
      parseVerb(word)

    elif tag.startswith('JJ'): #Word is tagged as an adjective
      parseAdj(word)

    elif tag.startswith('DT'): #Word is tagged as a determiner
      parseDT(word)

    elif tag.startswith('RB'): #Word is tagged as an adverb
      parseAdv(word)

    elif tag.startswith('.'):
      print('No need for punctuation or questions here.')

    else
      logUnrecognizedTag();
```

Listing 3.1 describes an extraction of the program used to handle user input. Each of the word classes of interest have their own parsing method which in turn sets a global variable to said word. The parser is only meant to handle one command at a time, meaning that a sentence such as:

"First I want to go north, pick up the dagger and kill the ogre."

would be too complicated. The decision was made to only perform the last thing the user said, meaning that the sentence above would be parsed as:

"kill ogre"

In the example below the word has been tagged as a delimiter by our NLI, and is sent to the "parseDT()" method.

```
def parseDT(dt):
  global determiner
  determiner = dt
```

This method sets the global determiner variable to what the current word is. Because only the last delimiter is the one of importance, there is no need to use a list or vector to keep track of any other potential delimiters in the input sentence.

3.3.1 Word tagging

Word tagging has a central part of the implementation. It is used to distinguish words from one another and gives information that is used to make decisions about how to further process the strings involved. The word tags that the NLI in the game uses are:

- N - Noun
- VB - Verb
- RB - Adverb
- JJ - Adjective
- DT - Determiner

A more extensive list of tags can be found in appendix A.

An analysis of the possible inputs for this type of text-based computer game is the basis of the choice of which tags to use. The tags mentioned above are considered crucial to be able to parse the input the game receives. But the amount of tags used is considered a minimal state to possibly get decent results.

Worth noting is that in the context of this game, it is not of relevance what sort of noun or verb it is. For example, it does matter if it is a plural proper noun (NNPS) or a singular noun (NN), since all nouns will be handled in the same way by the parser, see Listing 3.1.

The word tags are used in such way that NLTK sets them to a value that the program then can evaluate. This divides the different words of the input into "classes" and the classes should theoretically give a correct grammatical representation of the context. The game class is built as a command-based interpreter and therefore expects a code word as input. The NLI's task here is to translate the tagged natural language words into one of the code words and send it to the game class. To do this the parser taking the user input needs help to interpret what kind of code word is associated with respective word; this is usually done by using a corpus.

To make the results of the tagging more accurate, a custom model was added. A model in this case is a precomputed list of words with their respective tags. As an example, the model would contain the item:

'attack':'VB'

This would tell our word tagger that the word attack should always be tagged as a verb. If this is not added to the model, the NLTK word tagger assumes that "attack" is a noun, as in:

"They carried out an attack".

However, in the scope of the game, there would be no instance where the word "attack" would be a noun. Because the NLTK tagger has difficulty correctly tagging verbs without context (such as the suffix "-ing" or the word "to" before the verb) it was necessary to create a model which could handle verbs, even with little or no surrounding context.

3.3.2 Corpus

If the input has been categorized into the different word tags, the words subsequently need to be divided into a category that corresponds to a code word in the game. In this case, the range of commands that are useful in the game is small. That means that it is sufficient to save synonyms in the code in separate lists and verify the tagged words and see if they correspond to any of the words. Words that describe a direction in the game should then correspond to the following words:

```
direction = ["north","east","west","south","up","down","left","right"]
```

If the words do not correspond to any of the words above, it is determined that it is not valid direction and is then discarded for that case.

In the case of verbs the method was similar. A list of synonyms was created and the input was compared to the list of acceptable words. For example, the following verbs would indicate that the user wanted to attack a monster

```
attacking = ["attack", "kill", "hit", "punch", "kick"]
```

Some expressions come in pairs, such as "go north", where the parser needs to bind the verb and adverb together. To do this, a combination between the NLI's word tagging and the corpus lists are used to verify if the parser found both a verb and an adverb and if these are valid strings.

3.4 Testing

The results are obtained by testing the game in two different ways. The goal of the testing is to give information about how well the NLI performs in different situations. The result of the testing should help with the analysis of grammatical errors, sentence structure and expose weaknesses and strengths of the natural language processing that the NLI performs. The user testing also reveals useful information about the general capability of the game. To gain useful information from continuous tests a system for logging during runtime is used.

3.4.1 Automated tests

An automated "user" was implemented for two reasons. Firstly fewer human users than anticipated were able to test the program due to time constraints, and secondly it was of interest to see if computer generated input could be used to play the game. This option was researched in order to get a big number of tests, which will in turn produce a bigger dataset with more accurate results.

Dataset

As user testing is time consuming and might not always give the results expected, a precomputed set of data has been used to test the NLI as mentioned above. This study has been done under a limited time slot and therefore the precomputed input is added to make up for the possibly lacking user testing.

The automated user would create a sentence by randomly choosing a verb from a predefined list of acceptable words. Based on the verb, either a noun, such as an item, or an adverb, such as a direction, would be created from a similar list of acceptable words. In an attempt to make the sentence more structurally similar to a real sentence, there was a 60 percent chance that "filler words" would be added to the sentence. These filler words include "I want to", "Can I", and "I would like to". The lists of accepted words were intentionally kept small as to reduce the chance of creating too many incoherent sentences. See Appendix B for an example of a complete log of sentences created by the automated user.

Some occurrences of the precomputed sentences are not making sense, which is a good thing. The goal with having a precomputed dataset is to obtain useful information about scenarios where flaws and strengths can be detected in the NLI. The predicted result is to see if the NLI can handle sentences which do not make sense contextually, but might be interpreted incorrectly in the game. For example:

"I want to dagger"

In the scope of the game, the word "dagger" makes sense. But in the case above, the sentence is incomprehensible and is supposed to be discarded accordingly.

The precomputed dataset also covers sentences that makes sense but are interpreted incorrectly. Sentences that are not making sense according to the logic of the natural language processing are discarded by the NLI. But a sentence that is discarded could be evaluated by the human eye that it should make sense in the aspect of the game.

The automated user is able to play the game and either fail or complete it. But it is no mean a bot that is created to successfully complete the game. Due to the complexity of creating bots which act like a human, the automated user is only used as a way to produce interpretation results.

3.4.2 User testing

User tests were carried out to give an increase of variety of the input and to get the user perspective on text-based games based on an NLI. The game will receive input that otherwise is difficult or impossible to automatically compute due to the uniqueness of human users.

A total of 13 users tested the game. At the start of each test, they would be asked to enter their name, and were informed that their input would be logged, but that this information was to be confidential. The user was then given the following introductory sentence:

This game is written using a Natural Language Interface, meaning
that it will try to analyze the sentences/phrases you enter.
The goal of this simple text-based game is to escape from a cave.
In each room there are a number of exits, and perhaps other surprises.
Good luck!

To ensure fairness and equal circumstances for everyone, all users testing the program were given basic background information about the game. The users were told that the game was created using an NLI, but no stress was made on making sure that the users used a "natural language". The user was told that the program should, in theory, be able to handle all types of input. During testing, the user could ask any question and was given a corresponding answer, excluding questions regarding how to complete the game. If a user entered two similar inputs in a row with minor faults (e.g. incorrect spelling, incorrect grammar or multiple commands in a single sentence) the user would then be told what was wrong with the input. No user was allowed to look at the code either prior to or after the testing was completed, but users could, if they wanted to, look at their own log.

3.4.3 Test logs

All tests were logged in the same way. The file would begin with the name of the test user, and each following line followed the format of:

timestamp: command – Hotkey:'letter'

The timestamp is the time when the command was parsed, the command is the user input before parsing, and the hotkey is the abbreviation of the command, which was used in the game to proceed to the next step.

The logs gives information about all details happening in the game that is useful for the end result. Examining the logging data gives a statistical perspective of how well the NLI performs to interpret the input. The logs only focus on the connection between input and outcome, and to get a deeper understanding of errors that may occur, further analysis is carried out.

An extraction of one of the automated test looks like this:

```
15:04:33: I want to attack – Hotkey:a
15:04:33: Can I proceed west – Hotkey:w
15:04:33: I would like to proceed right – Hotkey:r
15:04:33: exit – Hotkey:?
15:04:33: I would like to run – Hotkey:f
```

The input to the game is logged as well as the translation of the sentence written at that point.

3.4.4 Questionnaire

A questionnaire was created and presented to the user after the user had completed or failed the game. The questions were designed to give an insight in how proficient the user was with computers, how well they felt the game was designed and how they thought text-based games could be improved in general. Although not paramount to our report, the questionnaire also asked for age and gender, this to see if there was a correlation between language used, or time taken to complete the game, and gender or age.

The questionnaire was supposed to help to get an understanding of how well the NLI behaved from the user perspective and if it is suitable to use an NLI to play text-based games.

Chapter 4

Result

In this section results from the user- and automated tests and the questionnaire are demonstrated to show to which extent the implemented NLI is useful in a text-based game environment.

4.1 Automated tests

The automated tests underwent several phases of program runs which led to small fixes to eliminate bugs in the program code. One test consists of several automatically computed phrases that are sent as input to the game. The outcome of every phrase can be placed in one of the following categories:

- **Correct** - The interpretation of the sentence was correct and the corresponding function in the game was executed.
- **Incorrect** - The interpretation of the sentence was incorrect and thus the wrong function in the game was executed.
- **Unable to parse** - The NLI was not able to interpret the sentence, conceptually similar to a syntax error.
- **Incomprehensible** - The NLI interprets the sentence as something that does not correlate with the games context.
- **Correctly marked as incomprehensible** - The sentence did not make sense and was correctly discarded.
- **Incorrectly marked as incomprehensible** - The sentence did make sense according to the context of the game and was incorrectly discarded.

The testing results in table 4.1 consist of twenty full runs of the game. The automated tests eventually complete or fail to complete the game, which is noted in as either S (success) or F (failure). To represent the data from the tests statistically, an analysis of the data was performed. This was needed to evaluate the outcome and inspect ambiguous sentences more closely.

The amount of data was sufficient to get useful information about the parsing the NLI performs. Each individual test would parse between approximately 100 to 1000 sentences, and a total of 7210 sentences have been generated to analyse the correctness of the NLI.

Figure 4.1. Results from 20 of the automated tests.

Log	Number of parsed phrases	Correct parsing	Incorrect parsing	Unable to parse	Incompre -hensible	Correctly marked as incompre -hensible	Incorrectly marked as incompre -hensible	Failure / Success
1	15	15	0	0	0	0	0	F
2	82	59	3	5	15	5	10	F
3	89	61	4	5	19	0	19	F
4	807	526	33	37	211	30	181	S
5	226	158	9	14	59	7	52	F
6	63	47	3	3	10	1	9	F
7	537	345	23	22	147	17	130	F
8	436	271	19	19	127	15	112	F
9	91	59	3	5	24	4	20	F
10	385	253	18	21	93	19	74	F
11	352	237	14	11	90	11	79	F
12	1140	720	70	46	304	33	171	S
13	124	87	7	3	27	2	25	F
14	175	122	14	4	35	3	32	F
15	542	340	20	37	145	18	127	S
16	680	452	26	38	164	19	145	F
17	242	158	8	6	70	11	59	F
18	393	248	19	22	104	8	96	F
19	734	467	37	38	192	19	173	F
20	97	62	2	6	27	3	24	S

The automated tests have varied results on the amount of input-phrases needed to either complete or fail the game. The first and third runs noticeably differ from the average result when it comes to discarding incomprehensible sentences correctly. The results of the remainder of the tests have approximately the same percentage of correctness.

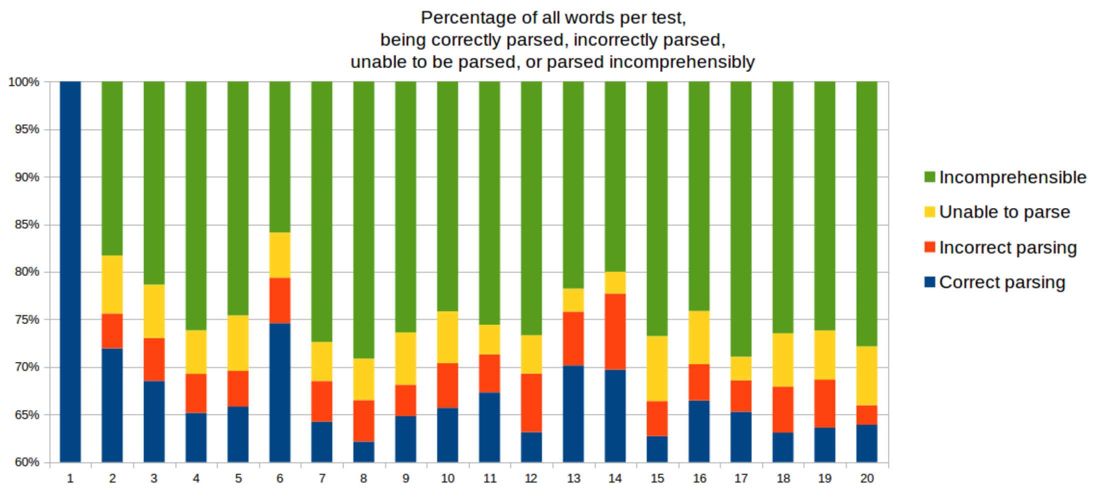

Figure 4.2. Results from 20 of the automated tests in a graph, listing correctness by percentage. Note that the scale on the y-axis starts at 60%.

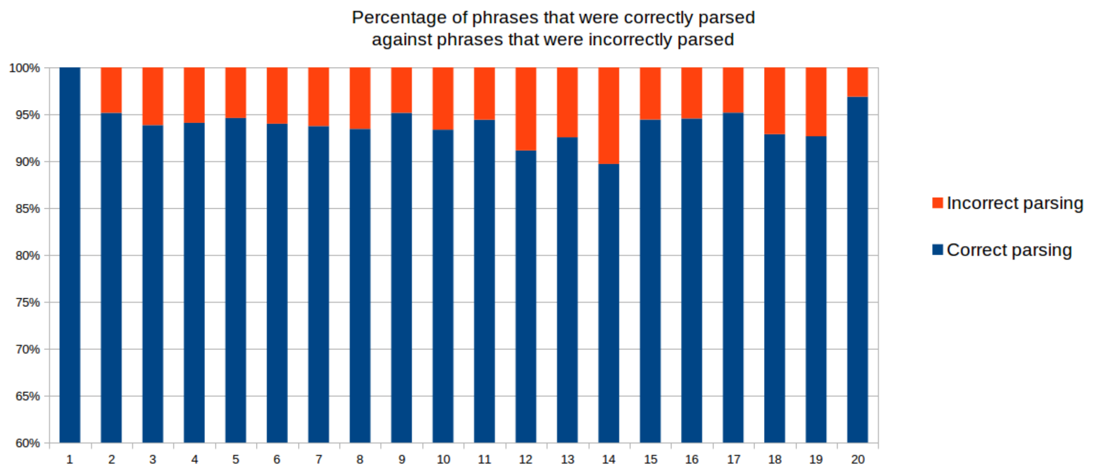

Figure 4.3. Sentences interpreted correctly and incorrectly by percentage. Note that that the scale on the y-axis starts at 60%.

The anomalies are not prevalent. This is due to the small differences in the automated data used. The first test had a significant deviation from the other tests in the amount of sentences needed to reach an exit of the game. From a theoretic perspective figure 4.1.1 does not give much more useful information than that the average correctness of the NLI is about 70 percent and the incorrectly parsed sentences are uncommon. This is more clearly displayed in figure 4.1.2, and shows that the amount of incomprehensible sentences is high. Since this graph does not give information about the correctness of the incomprehensible sentences, it is interesting to display the result of that in a separate graph.

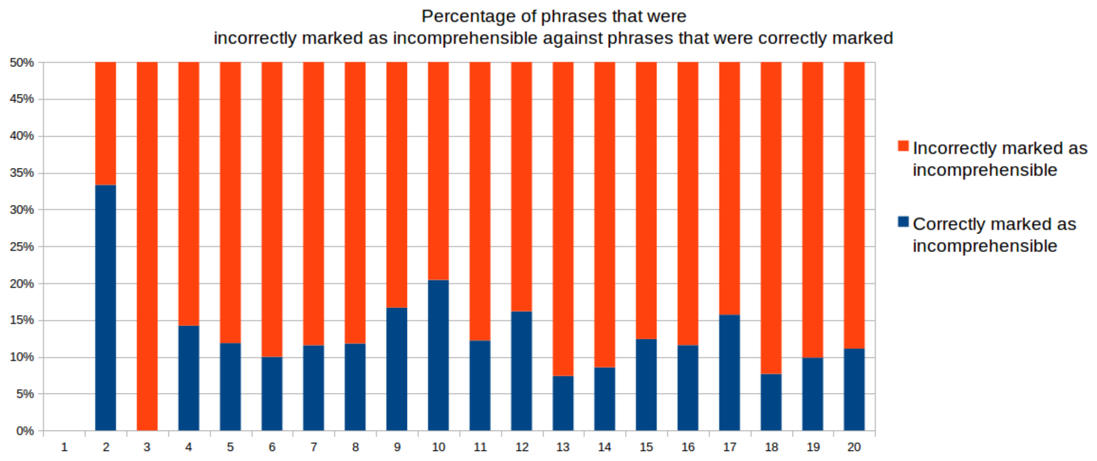

Figure 4.4. Incomprehensible sentences by percentage. Note that the scale on the y-axis stops at 50%.

Figure 4.1.3 shows that the NLI incorrectly marks the majority of the incomprehensible words, which is undesirable. All of the incorrectly interpreted sentences are supposed to make sense to the NLI. Examples of incomprehensible sentences which should not be put in that category are:

10:51:46: look item – Hotkey:?
10:51:57: walk north – Hotkey:?

The errors above happen because of how NLTK parses words. In the first two cases, the verbs "look" and "walk" are interpreted as nouns, as is "take a look" and "take a walk" respectively, and this happens because of the lack of determining words, such as prepositions and conjunctions.

4.2 User tests

The automated tests gave a sufficient amount of information about how well the NLI behaves. The user tests did not perform that well in that manner, but instead gives a more general view on NLI-based games from the users perspective. The most interesting information obtained is whether the user is satisfied with the performance of the NLI.

The theoretical outcome of the parsing is not a focus in the results of the user testing, since the automated tests covered that part more accurately. Instead the table below focuses on how well the users performed and if there were any phrases that were incomprehensible.

Figure 4.5. Results from the user testing.

	User	C	Min. WPC	Max. WPC	Avg. WPC	Time (m:s)	?	W	R	S/F
	1	9	1	2	1.78	00:00:59	1	0	8	S
	2	13	2	4	2.23	00:04:04	1	0	12	F
	3	15	1	2	1.67	00:03:28	0	0	15	S
	4	15	1	2	1.8	00:01:37	2	0	13	S
	5	31	1	2	1.77	00:03:57	3	0	28	S
	6	42	1	3	1.9	00:03:59	2	0	40	F
	7	14	1	2	1.92	00:00:58	0	0	14	S
	8	16	1	2	1.69	00:01:52	3	0	13	S
	9	17	1	9	3.35	00:01:42	2	3	12	S
	10	21	1	9	2.38	00:05:49	5	1	15	S
	11	21	1	5	2.14	00:04:22	4	1	16	S
	12	14	1	3	2	00:02:30	2	0	12	S
	13	24	1	4	2.16	00:03:59	8	0	16	S
Avg:		19.38	1.08	3.77	2.06	00:03:01	2.54	0.38	16.46	

- **C** - Total number of commands/sentences written by the user
- **Min. WPC** - The amount of words written to build the shortest command/sentence
- **Max. WPC** - The amount of words written to build the longest command/sentence
- **?** - The total amount of incomprehensible words
- **W** - Number of words interpreted incorrectly
- **R** - Number of words interpreted correctly

The majority of the users that performed the testing had a good knowledge of computer science and therefore they had a tendency to pull towards a more formal language consisting of a few code words. The scope of the input that the NLI accepted was not appealing enough to the average user. The users which had little experience of computer science tried to speak more naturally to the game, but this also led to long sentences which tried to do several operations in one go, and that is not possible to do in the game.

4.2.1 Questionnaire results

The questionnaire mainly consisted of check questions ranging from one to five, to be able to represent the results as statistics. The users' answers are displayed in figure 4.6.

Figure 4.6. Results from the questionnaire.

	User	Age	E	D	I	O	CG
	1	22-25	5	1	1	5	5
	2	18-21	5	2	4	5	5
	3	22-25	5	1	4	5	4
	4	18-21	1	4	4	5	4
	5	18-21	1	4	4	5	4
	6	18-21	4	2	3	4	4
	7	18-21	5	3	4	4	3
	8	22-25	5	3	4	3	5
	9	18-21	5	3	4	5	4
	10	22-25	3	1	5	5	5
Avg:			3.9	2.4	3.7	4.6	4.3

- **E** - General experience of programming and computers, ranging from 1 to 5, where 5 is good experience (studying Computer Science)
- **D** - The user's opinion on the difficulty of the game, ranging from 1 to 5, where 1 is easy, 5 is difficult
- **I** - The user's opinion on the amount of information given to successfully play the game, 1 is not sufficient amount of information, 5 is enough information
- **O** - The user's opinion on the amount of in game options given to successfully complete the game, 1 is not sufficient amount of options, 5 is enough options
- **CG** - The user's opinion on the how well the game interpreted their input, 1 is bad, 5 is good

Chapter 5

Discussion

The displayed results and evaluation of the incomprehensible words demonstrates that word tagging is significantly the most important part of the NLI to get good results in a text-based game environment. Text-based games often end up having a preset and limited amount of operations and users also tend to keep it simple when writing input to the game. This means that the range of the natural language processing levels is restricted to a certain area. The NLI developed has parts which work with lexical-, syntactic- and semantic analysis, where the lexical analysis (word tagging) is the core of the NLI.

The incomprehensible words that were incorrectly interpreted were mostly not explanatory enough, an example of an incomprehensible input is:

10:51:46: exit – Hotkey:?

"Exit" is an ambiguous word in the context of the game if sent alone as input. This is the case since it is possible to flee or escape a zone whenever the player wants to. However, the code word "exit" could also be used to quit the game. This relates back to the example mentioned in section 2.2.1, where a phrase could be considered ambiguous. Because the game is played from a first-person perspective, it was always safe to assume that the commands entered were to be carried out by the player. This also relates to the second example mentioned in 2.2.1. Because of the limited scope of the game, no major thought had to be given to analyzing the semantics of the subject and the object. In the case of the game, the ambiguity was not created because of a lack of subject, but because of a words multiple meanings or interpretations. This leads to the NLI tagging words incorrectly. If it happens that there is a determining word preceding the word, this greatly increases the chances that the word would be tagged correctly. Enhancing semantics helps tagging the cases where there is a determining word. Otherwise the built in tagging has to handle ambiguous words, which could result in incorrect word tagging.

Another thing that helps to get correct interpretation results is morphology-theory. Prefixes and more likely suffixes help to distinguish verbs from other type of tags. The word "look" is interpreted incorrectly by the NLI too such an extent that it causes problems. If the input instead would consist of the word "looking", the NLI would interpret it correctly as a verb.

What makes natural language processing difficult is how big of a dataset that has to be handled. For a quite small text-based game, it is easy to evaluate the variety of words that could be used to try to communicate with the game. For the type of game that has been implemented, the outcome should ultimately be branched into a category of words that makes sense or one that does not. This leads to a fixed dataset and a reasoned conclusion that it should not be impossible to build an NLI in a text-based game environment which handles the majority of the input it gets. One problem with natural language processing technology lies in the uncertainty of what the input might be. For example Apple's Siri is supposed to answer any type of question, which is a complex task, since the dataset to be handled is big if not infinite.

5.1 Analysis of testing results

The user testing shows that users tend to figure out how to play to game with minimal input, rather than using a natural language. The user logs show that when a user realized they could type a short sentence instead of a long sentence, they would always write the shorter version of the sentence. This means that the users' input would not follow the more advanced grammatical rules of a natural language. The extent of possible operations in the game was limited and therefore it probably was not appealing enough to write longer more natural sentences since it did not give the user any extra information to complete the game. The results from the questionnaire also show that most of the users were pleased with how the NLI interpreted the input.

The information given to the user was intentionally slightly more vague than necessary. This was done in order to try and get the user to use a natural language. If exact instructions had been given, it is possible that the input received would have been less natural, and would have followed the specified guidelines, and this was something to be avoided. But this did not go as planned, since most users used a formal language.

Analysis of the user tests also showed interesting results about the users. A majority of the users did not attempt to use the in-game "help" option; only 38 percent of users entered a command to receive help. As an additional result, the study showed that 61 percent of the users chose to move north as their first recognizable command. A more detailed study of user input could have been carried out to analyze the reasoning for why the users wrote what they did, but that is beyond the scope of this report.

The results also show that the users' general consensus was that there were more than enough options to complete the game. This shows that text-based games such as the one implemented might not be well suited for use with an NLI, because of the limited amount of options needed to complete the game. Since the game only takes a certain number of commands, all the NLI has to do is to see if the input "matches" one of the predefined commands; if not, it is disregarded. Compare this to a speech recognition application, which needs to be more versatile, and as such is better suited for use with an NLI.

The automated tests gave useful information about the NLI. The result indicated which parts of the NLI that showed on weaknesses. The NLI could be improved by implementing better semantic analysis and a system which handles morphology (see section 2.3.2). For example, verbs without the suffix "-ing" are interpreted incorrectly most of the time. This is because the built in word tagging has to work with an ambiguous word and therefore tags it incorrectly for this context. Since NLTK is not handling this kind of linguistics, it needs to be implemented separately. The verb can be more thoroughly evaluated by trying to bind it together with another subject and object. A corpus for prefixes and suffixes can also be added to partition words and look at differences in word tagging.

The expectation of the NLI created was to give interpretation results which made the game playable and this expectation was partially met. In the automated tests, the accuracy was between 65 to 75 percent. In the user tests, the accuracy was between 66 to 100 percent, with an average of 84.9 percent. This can be attributed to the fact that no form of machine learning was implemented; meaning that if a faulty input was generated, it could potentially be generated again, whereas a human would recognize that a mistake was made, and not perform the same action again.

5.2 Improvements

Our results could have been improved in a few ways. When creating our NLI, more focus could have been put on trying to make the parser handle verbs better. A custom model was developed, but was not advanced or accurate enough. More time could have been spent on making the automated tests seem more human-like, and more time could have been spent gathering user testing data.

It is also interesting to state that out of the linguistic levels mentioned in section 2.2, the correctness of the parsing that the NLI carries through, relies mostly on the lexical analysis that word tagging entails. Other than that, special cases have been handled using syntactic and semantic analysis. But to be certain of getting almost perfect interpretation of sentences for the scope of data used, it could have been enough to implement an NLI which takes morphology-, lexical-, syntactic- and semantic theory in consideration.

The results also indicate that using an NLI to control a computer game might not be the best alternative. The idea of using a natural language instead of a formal one is overlooked by most users. In the end this leads to all the parsing and interpretation steps being excessive. With that said, the scope of information used in computer games should generally not be too complicated and an NLI should still be highly relevant to use as a feature in a game.

Chapter 6

Summary

The automated tests gave us enough information to establish that the tools NLTK provides are not sufficient alone to build a simple NLI. The environment a text-based game describes consists of several special cases that have to be handled separately. The limited amount of time given in the course restricted us from carrying through thorough error handling. We can also conclude that it is not always that efficient to use natural language as an interface to control a game. But NLI's are viable and could be of good use in a gaming environment. With this said, we still believe that building an NLI for a text-based game of this size that can perform perfect interpretation, is possible.

Bibliography

[1] Bird, S., "Timeline of Computer History", Computer History Museum, CA, 2006. Available at: *http://www.computerhistory.org/timeline/?category=cmptr* [Accessed 2015 February 26]

[2] Johnson, P. (2014), "AdventureTutorial", [Computer Program], Available at *https://github.com/phillipjohnson/text-adventure-tut* [Accessed 2015 March 31]

[3] Source code to the computer game Colossal Cave Adventure, Accessed from *http://www.icynic.com/~don/jerz/advdat.77-03-11*

[4] About Siri, Apple Inc., Accessed from https://support.apple.com/en-us/HT204389

[5] Façade, a one-act interactive drama, a computer game developed by Procedural Arts, 2005-2012, accessed at *http://www.interactivestory.net/technology/*

[6] Mateas, M and Stern, A., "A Behavior Language: Joint Action and Behavioral Idioms", College of Computing and School of Literature, Communication and Culture Georgia Institute of Technology, Available at: *http://egl.gatech.edu/ mateas/publications/MateasSternLifelikeBook04.pdf* [Accessed 2015 February 26]

[7] Bird, S., Klein, E., and Loper, E., "Language Processing and Python", O'Reilly Media Inc. 2009, "Natural Language Processing with Python", Available at: *http://www.nltk.org/book_1ed/ch01.html* [Accessed 2015 February 18]

[8] Bird, S., Klein, E., and Loper, E., "Afterword: The Language Challenge", O'Reilly Media Inc. 2009, "Natural Language Processing with Python", Available at: *http://www.nltk.org/book/ch12.html* [Accessed 2015 February 18]

[9] Jiang, T. et al, "Formal Grammars and Languages", Department of Computer Science, University of Rhode Island, Available at: *http://www.cs.ucr.edu/~jiang/cs215/tao-new.pdf* [Accessed 2015 March 15]

[10] Liu, X., "Natural Language Processing", School of Information Studies at Syracuse University, Accessed from *http://surface.syr.edu/cgi/viewcontent.cgi?article=1043&context=istpub*

[11] Bird, S., Klein, E., and Loper, E., "Learning to Classify Text", O'Reilly Media Inc. 2009, "Natural Language Processing with Python", Available at: *http://www.nltk.org/book/ch06.html* [Accessed 2015 February 18]

[12] Pinker, S., "The Language Instinct: How the Mind Creates Language", William Morrow and Company, 1994

[13] Bird, S., Natural Language Toolkit Github. Available at: *https://github.com/nltk/nltk/wiki/FAQ* [Accessed 2015 February 26]

Appendix A

NLTK Word tags

Abbreviation	**Name of clause**	**Examples**
CC	Co-ordinating conjunction	and, but, for, or, yet
CD	Numeral (cardinal)	forty, twenty, 98, .05
DT	Determiner	all, both, each, either, much
FW	Foreign word	ich, fille, Herr, pas
IN	Preposition or conjunction	atop, behind, towards, inside
JJ	Adjective	first, oiled, separable, ill-mannered
JJR	Comparative adjective	braver, calmer, darker, faster
JJS	Superlative adjective	cheapest, calmest, deadliest
MD	Modal auxiliary	can, might, ought, should
NN	Noun (singular)	artichoke, bard, casino, duster
NNS	Noun (Plural)	products, designs, clubs
NNNPS	Proper noun (plural)	Americans, Animals, Andes
NNP	Proper noun (singular)	Motown, Sawyer, Liverpool
PDT	Pre-determiner	all, both, half, many
POS	Genitive marker (possessive)	’s
PRP	Personal pronoun	ours, him, self, theirs
PRP$	Possessive pronoun	her, mine, your, thy
RB	Adverb	occasionally, swiftly
RBR	Comparative adverb	harder, better, faster, stronger
RBS	Superlative adverb	best, first, biggest, worst
RP	Particle	about, across, over, upon
UH	Interjection	Gosh, Oops, Wow
VB	Verb (base form)	ask, behold, build
VBD	Verb (past tense)	dipped, pleaded, aimed, wore
VBG	Verb (present participle)	stirring, focusing, erasing
VBN	Verb (past participle)	initiated, dubbed, experimented
VBP	Verb (present tense)	wrap, sue, postpone, sever
VBZ	Verb (3rd person singular)	bases, mixes, seduces, speaks
WP	"WH"-pronoun	what, whatsoever, which, whom

Appendix B

Automated test run

```
10:51:52: I want to flee – Hotkey: f
10:51:52: attack – Hotkey: a
10:51:52: I would like to flee – Hotkey: f
10:51:52: go left – Hotkey: l
10:51:52: exit – Hotkey: ?
10:51:52: I would like to go up – Hotkey: u
10:51:52: hit – Hotkey: a
10:51:52: I want to move south – Hotkey: s
10:51:52: I want to attack – Hotkey: a
10:51:52: inspect dagger – Hotkey:
10:51:52: I would like to go south – Hotkey: s
10:51:52: I want to hide – Hotkey: f
10:51:52: I want to proceed up – Hotkey: u
10:51:52: attack – Hotkey: a
10:51:52: I would like to view inventory – Hotkey: i
10:51:52: I would like to look inventory – Hotkey: i
10:51:52: hit – Hotkey: a
10:51:52: Can I punch – Hotkey: a
10:51:52: Can I kick – Hotkey: a
10:51:52: inspect item – Hotkey: i
10:51:52: I want to kill – Hotkey: a
10:51:52: I would like to leave – Hotkey: f
10:51:52: crawl right – Hotkey: ?
10:51:52: Can I look item – Hotkey: c
10:51:52: Can I inspect inventory – Hotkey: c
10:51:52: I would like to flee – Hotkey: f
10:51:52: Can I move down – Hotkey: d
10:51:52: I want to flee – Hotkey: f
10:51:52: look item – Hotkey: ?
10:51:52: I want to leave – Hotkey: f
```

```
10:51:52: punch – Hotkey: ?
10:51:52: travel up – Hotkey: ?
10:51:52: Can I inspect item – Hotkey: c
10:51:52: jump east – Hotkey: e
10:51:52: Can I hide – Hotkey: f
10:51:52: I want to punch – Hotkey: a
10:51:52: go left – Hotkey: l
10:51:52: hit – Hotkey: a
10:51:52: view dagger – Hotkey:
10:51:52: I want to leave – Hotkey: f
10:51:52: I want to move east – Hotkey: e
10:51:52: I would like to hit – Hotkey: a
10:51:52: look inventory – Hotkey: ?
10:51:52: crawl south – Hotkey: ?
10:51:52: I want to move north – Hotkey: n
10:51:52: proceed left – Hotkey: l
10:51:52: I want to attack – Hotkey: a
10:51:52: I want to view item – Hotkey: i
10:51:52: I want to cross west – Hotkey: ?
10:51:52: I want to travel east – Hotkey: ?
10:51:52: jump west – Hotkey: w
10:51:52: cross left – Hotkey: ?
10:51:52: Can I flee – Hotkey: f
10:51:52: show dagger – Hotkey:
10:51:52: I would like to attack – Hotkey: a
10:51:52: Can I go left – Hotkey: l
10:51:52: Can I hide – Hotkey: f
10:51:52: I would like to crawl east – Hotkey: e
10:51:52: proceed north – Hotkey: n
10:51:52: I want to flee – Hotkey: f
10:51:52: I would like to look item – Hotkey: i
10:51:52: I would like to cross south – Hotkey: s
```