AINT512 – Speech Based Text Adventure

Demetrius Zaibo, 10324064, 07/05/2018

# Abstract:

This report outlines a text-adventure based conversational agent, which uses speech instead of monitors and keyboards. Section 1 explains the projects design with justification; section 2 details the design method for the conversational agent; section 3 discusses the results and feedback from play testers; and section 4 summarises the project.

# 1 – Design Overview

## 1.1 – Game Design

To allow for rapid development of the game, along with various game features, a simple game-engine for text adventures was developed. This enabled rapid feedback integration, but also meant certain features became too complex to develop, such as being able to pause or interrupt the narration.

A pre-existing game, ‘The Stanley Parable’, was used as a guide for the design of the computer’s personality, and the general flow of the game. The games layout was designed using tools from the choose-your-own-adventure genre of novels; the story was created from personal experiences; and the tutorial designed iteratively via numerous playtests.

## 1.2 – Speech to Text Design

Grammars were originally used with the speech to text algorithm, as parsing several grammars allows for a direct connection between an expected sentence and the desired action of the user. However, due to the wide variety of possible input phrases a user could generate for a given scenario in game, grammars were deemed infeasible. To generate a suitable textual corpus, a grammar of synonyms was generated for each word. Inflexions, such as ‘umm’ and ‘please’, were inserted between each word as a conditional. The number of possibilities that needed to be scanned by the grammar amounted to several minutes’ worth of processing; an unreasonable amount of time for any user to be expected to wait.

Statistical language models were tested as a substitute. Specifically, CMU-Sphinx was tested as an alternative offline model. CMU-Sphinx also allowed grammars to be tested, and could use custom dictionaries. This allowed for direct comparisons with grammars, and improved processing speeds as a limited dictionary size could be used. The results were usable, but highly dependent on a user’s pronunciation and accent. To alleviate this issue, two online language models are also used in conjunction, Google and Wit.

## 1.3 – Semantic Analysis Design

Semantic analysis is guided by a set of available options, defined by the user’s position within a finite-state model. Each of the three language models mentioned above process user inputs into textual results, along with an associated confidence metric. These results are treated as bags-of-words, where each word is assigned a weight towards a specific narrative option. The weights are calculated via the Term Frequency Inverse Document Frequency (TFIDF) metric. The results were multiplied by the confidence of each result, and all of those summed together. These metrics, one for each narrative selection, are then sorted from largest to smallest. If the largest metric is not sufficiently large enough, then it is likely that a user attempted to select an invalid option. If the two largest metrics are of similar value, then a selection cannot be made. This is likely an issue relating to the word weights, or how the user was guided towards their response. Otherwise, the selection is considered valid.

# 2 – Design Process:

## 2.1 – Basics:

***Task Domain:*** The program domain is to present and guide a user through a custom-made choose your own adventure story.

***Dialogue Specification:*** The dialogue specification and narrative paths overview can be seen in figures 1 and 2.

***Personality Specification:*** The narration used is polite, well spoken, and slightly humorous.

## 2.2 – Grammar Design:

User inputs are expected to be in one of several formats. Yes and no answers either acknowledge or reject the directed narrative. Direct responses, containing a single word or a short phrase, will directly mention some aspect of the directed narrative. This will provide enough context to infer the user’s selection. Due to using language models and the bag-of-words approach, indirect responses should react similarly to the direct responses.

The set of possible user responses was originally generated via several human-to-human trial runs, with casual notes made within relevant story sections. This corpus then began continuous development as prototypes of the game were made, tested, and logged.

Figure 1: Human-Computer Interface Specification



Figure 2: Overview of the game’s Narrative Structure

## 2.3 – Dialogue Design:

The dominant dialogue management technique used is the ‘finite-state model’ approach. However, aspects of the ‘information state update’ approach are used to aid the finite-state model, and a ‘task-based’ approach is used to infer the user’s intentions and progress with the narrative. The dialogue type used was a ‘mixed-initiative’, with a preference towards ‘user-initiative’ input. The dialogue guides the user towards a single valid option, but leaves them to decide whether to take that option or not, and attempts to accept as wide a range of possible inputs as possible so as not to disrupt any immersion the user may have with the game.

The dialogue, or narrative, is built up using a variety of techniques. Specifically scenes, sequels, directed dialogues, user input and computer processing cues, and the three standard options for narratives in choose your own adventures. Scenes are moments of conflict or hardship, experiences by both the character and the reader. Sequels present the character’s reaction to the previous scene, and set up motivations for the next scene. Directed dialogues guide the user towards available narrative options without having to list them. User input and computer processing cues alert the user to internal states of the computer, allowing them to interact at valid times without being frustrated at slow responses. Finally, the three standard options are ‘yes’, following the directed narrative, ‘no’, doing the exact opposite of the directed narrative, and ‘something else’, which are alternative options that a user might try in such a scenario. These alternatives may often be equivalent to asking ‘what else can I do’, such as saying ‘look around’. Deducing valid alternatives comes down to narrative choice and play-tester feedback.

## 2.4 – Error Catching & Handling:

The program distinguishes between four types of errors. The simplest of these is silence, whereby no input is provided by the user. Under this condition, the program displays a note of the silence and continues to wait for user input again. Other responses appeared to interfere with the user’s immersion within the game.

Another error type results from non-vocal, or unintelligible, inputs. Again, to prevent a break in immersion, these errors are simply ignored. However, should the confidence of a translation from such inputs be high enough, the error will become a mistranslation. Incorrect responses would also fall under this category. These errors are difficult to detect due to the nature of language models and the bag-of-words algorithm, and is further compounded with the addition of synonyms for each narrative option. Should no valid selection be identified, a user error is raised, and the user is kindly asked to try again in a variety of different methods. All of this is done within the semantic analysis.

The final error occurs when multiple narrative options are identified given the users input. This is considered a creator error as most inputs will be valid, but poor word weightings within the program prevented a correct assignment. The program humorously apologises and asks the user to rephrase their input in an attempt to prevent the error occurring again. This, also, is caught within the semantic analysis.

The functional limitation of requiring online speech to text converters was resolved with the use of CMU Sphinx, an offline solution. However, checks for a functional microphone or speaker are not incorporated at this time.

## 2.5 – Action / Response Generation:

Responses are entirely pre-scripted. Certain sections can change based on internal states within the program, such as ‘how many times the user has been to this part of the narrative before’. The only exception to this is the ‘user error’, which can help in one of two ways. This provides a confirmation that the computer misheard the user, or verification that the computer heard correctly but the intended path was invalid, by replaying the interpreted result back to the user.

## 2.6 – User Evaluations:

Three evaluation metrics have been generated. The first is a measure of how successfully the program infers the correct narrative selection based on the users input. The second metric categorises user inputs, and the third provides the likelihood of a user entering a particular section of the narrative.

The program can make four possible inference types using two parameters. If the translation is a somewhat accurate representation of what the user actually said, then it is considered ‘true’, otherwise, it is considered ‘false’. If the program selects the intended narrative option it is considered ‘correct’, otherwise, it is considered ‘incorrect’.

User inputs are categorised into three types: a yes-no type, if the users inputs roughly translate to such simple statements; a sentence-based director type, where the user explicitly instructs the character to do something; and finally, a sentence-based actor type, where the user could be immersing themselves into the game.

The third metric is a simple histogram over the different sections of the narrative covered.

# 3 – Play-Tester Feedback

From the log files generated during each play-through, it was possible to collect the relevant pieces of information for assessment with relative ease. In total, eight independent play-testers provided 180 selections from which to assess the systems performance.

## 3.1 – Verbal Feedback:

Feedback for the game changed throughout the development cycle. However, several points were of note:

* There needs to be an explanation, within the game, to notify users that humming corresponds to the computer thinking about their response, and that this humming-time lasts roughly 5-7 seconds.
* A dialogue needs to occur before the game starts to ensure a user’s attention has been attained.
* The tutorial demo interaction should require more than a yes-no response to ensure aid user understanding.
* Being able to pause or interrupt the narrative would be a useful feature.
* The game was engaging and interesting

## 3.2 – Text to Selection Accuracy:

To assess how often a user’s input selected the intended option it is necessary to break the task into two sections. Firstly, how well can the language models convert a user’s speech into text? Secondly, how often does a successfully translated text end up selecting the intended narrative? This distinction is necessary as most of the play-testers had strong accents, which may have reduced the reliability of the speech to text conversion.

Comparing the three text to speech engines (language models), Google, Wit, and CMU Sphinx, we find that their success rates are 72.8%, 59.4%, and 28.3% respectively. A converted text is considered successful if any of the three language models successfully converted the speech into text correctly. Of the 169 remaining samples, 92 selected the intended option. Of the 53 failed selections, 12 were due to issues with the game itself, 25 were instances where the game failed to flag a user error, and 16 were due to incorrect weightings for each word. This corresponds with a 63.5% success rate, a 17.2% failed user error rate, an 11.0% word weighting error rate, and an 8.3% creator error rate.

## 3.3 – User Input Types:

Of the 180 samples, only 156 were interpretable for the purposes of this evaluation. Of those, the most common interface type was the sentence-based director, with 118 instances (75.6%), where the game was directly instructed as to what to do next. Boolean yes-no responses were used 27 times (17.3%), and sentence-based actors, where the player could be immersing themselves, 11 times (7.1%).

## 3.4 – Game State Frequency:

The game can be categorised into five distinct sections. From figure 2, these sections would correspond to the first scene, the “light” option, the “dark” option, endings, and everything else. Of the 180 examples, user’s time was spent as follows: first scene, 50 instances, 27.8%; “light” option, 42 instances, 23.3%; “dark” option, 15 instances, 8.3%; endings 17 instances, 9.44%; other, 56 instances, 31.1%. Of the 56 other instances, it should be noted that 24 of those were allegedly asking for the date. This indicates that the word weightings for that option were set too high.

# 4 – Conclusions:

This report has briefly outlined the design of a conversational agent designed to function as a vocally interactive text adventure. Test-data was collected and analysed, which found the system could correctly translate a user’s input to the correct selection only 63.5% of the time. The sources of errors were assessed and categorised into three types. User commands were also assessed, such that future improvements can hone in on expected inputs more. Finally, the frequency of various narrative states occurring was assessed, pointing towards a potential source for one of the system errors.

# 5 – References:

Dib, Firas. "Online Regex Tester and Debugger." Regular Expressions 101. Accessed May 7, 2018. https://regex101.com/.

Irudaya Regilan, Maria J. "How Can I Convert a .py to .exe for Python?" Stack Overflow. Accessed May 7, 2018. https://stackoverflow.com/a/44433442.

Kalra, Puneet. "Basics of Java Speech Grammar Format ( JSGF )." Puneet Kalra. Last modified April 27, 2011. https://puneetk.com/basics-of-java-speech-grammar-format-jsgf.

Shmyrev, Nickolay. "CMUSphinx Documentation." CMUSphinx Open Source Speech Recognition. Accessed May 7, 2018. https://cmusphinx.github.io/wiki/.

"SpeechRecognition." PyPI. Accessed May 7, 2018. https://pypi.org/project/SpeechRecognition/.

"Text to Speech – Python." Python. Accessed May 7, 2018. https://pythonprogramminglanguage.com/text-to-speech/.

Thesaurus, Power. "Crowdsourced, straightforward, user-friendly thesaurus." Power Thesaurus. Accessed May 7, 2018. https://www.powerthesaurus.org/.

Woodward, Karen. "How To Write A 'Choose Your Own Adventure' Story." Karen Woodward. Accessed May 7, 2018. http://blog.karenwoodward.org/2014/06/how-to-write-choose-your-own-adventure.html.

# Appendix – Getting it to run

The game provided with this report is known to work on Windows 10 machines with an inbuilt microphone, using Python 3.5.4 (v3.5.4:3f56838, Aug 8th 2017, 02:17:05) [MSC v.1900 64 bit (AMD64)] on win32. The code was developed in PyCharm 2017.2.4, and is recommended to be run from there in order to aid installation of the various additional python packages required for this project to function. The file ‘Main.py’ is the file to be run to start the game.

Alternatively, a pre-compiled executable, Main.exe, is also available. Assume that all files are required to run the program, as the executable requires access to the story segments, grammar files, dictionary files, thesaurus files, and libraries to function.

For either version of the game, it is recommended that the game is played with internet access to significantly improve the performance of the speech to text converter.

Should a development timeline be required, it can be found via the commit history on the following GitHub website: <https://github.com/demzai/The-Vaguely-Parable>