401I - Final Year Project

Final Report

Voice Recognition RPG

Baron Khan (bak14)

Contents

1 Abstract **1**

2 Introduction **2**

3 Background **4**
3.1 Voice Recognition in Games 4
3.1.1 In Verbis Virtus 4
3.1.2 Skyrim Kinect 4
3.1.3 Star Trek Bridge Crew 5
3.1.4 Classic Zork on Alexa 6
3.2 Limitations of Zork 6
3.3 Voice Recognition Implementations 7
3.3.1 Tazti - Speech Recognition for PC Games 7
3.3.2 PocketSphinx for Unreal Engine 4 8
3.3.3 Houndify 8
3.3.4 Watson Conversation Engine 9
3.4 Natural Language Processing 11
3.4.1 Stages of Natural Language Processing 11
3.4.2 Part-of-Speech Tagging 13
3.4.3 Slot Filling 13
3.4.4 Synonyms and Hypernyms 14
3.4.5 Semantic Similarity 15
3.4.6 Generating Sentences 16
3.5 WordNet 17
3.6 Object Properties 17

4 Design **18**
4.1 Target Platform 18
4.2 Java Programming Language 18
4.3 System Overview 18
4.4 System UML Class Diagram 19
4.4.1 `MainActivity` Class 20
4.4.2 `VoiceControl` Class 21
4.4.3 `VoiceProcess` Class 21
4.4.4 `GlobalState` Class 21
4.4.5 `ContextActionMap` Class 21
4.4.6 `Semantic Similarity` Class 22

5 Appendix **23**
5.1 Skyrim Kinect Command List 23
5.2 Penn Treebank Tagset 25
5.3 System UML Class Diagram 26

References **27**

1 Abstract

When adding voice commands to a video game, a developer may find themselves hard-coding text strings for phrases that a player can say, and mapping them to intents within the game. It becomes more work to include varying versions of the same intent as the developer would have to include every permutation of the phrase (for which there could be infinitely many).

This project presents a text-based role-playing game (RPG) on Android which allows the player to issue actions within the game using their voice. The game uses a new voice recognition system that aims to make it easier for the developer to add voice commands without hard-coding the phrases.

We explore and evaluate various methods for improving the voice recognition system with the aim of decreasing the developer workload. These include using semantic similarity methods, a user-controlled learning mechanism, and various other NLP techniques.

2 Introduction

In recent years, video games have explored various forms of input that diverge from the traditional control schemes of a keyboard and mouse, or an analogue stick and buttons on a controller. Touch-screen controls and motion controls have had varying degrees of success over the years, and provide new forms of interactions with games. A recent form of input used in video games has been voice controls; a user utters a sentence or phrase and this would execute an action or intent within the game.

Voice recognition and control schemes have been used in various ways in the medium, whether in tandem with traditional control schemes, or as the only form of input. Most games which feature a voice control scheme are typically programmed to work with a specific set of keywords or phrases, hard-coded by the developer. This is usually the case with games where the voice control interface is entirely optional to user, as it requires little to no input from the programmer; if the speech processing is handled by a separate library, the developer just has to map a text string (e.g. "open the door") to the function that executes the intent (e.g. the function that opens a nearby door in the game).

It becomes increasingly difficult for the developer to add more phrases that can be accepted in the game and map to the same intent (e.g. "push the door open") as they would have to hard-code each possible input that the user could possibly say: having too few phrases would mean that the user could become frustrated when their preferred way of stating an intent is not accepted by the game, while adding too many acceptable phrases would increase development time.

Natural Language Processing (NLP, also referred to as Natural Language Understanding) has been used to improve inference of what users were trying to say, and to extract meaning from the user's phrases and sentences. For instance, if the user says, "push the door open", NLP can be used to infer that the user's intent is to open a door. Using NLP, several user phrases can be mapped to the same intent without having to hard-code each possible phrase.

NLP is already used to improve voice recognition in popular personal assistants such as Siri, but these systems offload the NLP workload to the server to reduce the amount of local processing required, as NLP requires a sizeable amount of machine learning and pattern recognition [1]. Most video games do not require an internet connection to play, and therefore can be played anywhere, so it may not be possible to use these cloud services for adding voice recognition to games.

The goal of this project is to apply NLP to a voice control scheme used within a role-playing game (RPG), in order to reduce the amount of work required by a

developer to add a flexible voice control interface, and to give more freedom of expression to the player while they play the game.

Role-playing games are a genre of video games that involve the player controlling a character in a world featuring exploration and/or battle mechanics, along with a progression system, where the character gains new abilities or items as the game progresses. These include turn-based games such as Pokemon and Final Fantasy, or more action-based games such as Dark Souls or Skyrim. In these games, the player possesses several items that they acquire during the game, and accesses them by navigating menus.

If the player has many different items, they may need to spend a lot of time searching the menus for a specific item, which can be quite tedious. It can also become quite repetitive if the user is constantly only pushing the "Attack" button over and over again with no variation in the attack used. Using voice recognition and NLP, a player could just simply say a phrase such as, "use a potion" or, "attack with axe", and the item will be used without having the player navigate several menu screens.

Another motivation for this project is to explore how voice controls can allow games to become more accessible to people with certain disabilities that prevent them from using traditional controllers or a keyboard and mouse. Very few popular games support voice recognition as a form of control, whether it is consoles, PC or mobile.

A text-based RPG will be created that will use a flexible voice control interface. It will feature a simple exploration mechanic, with the user manipulating objects and environments using voice commands, and a simple battle mechanic. These will be designed to fully demonstrate the power of the voice-control system to make the game easier to play, as opposed to using traditional mouse clicks and button presses. The target platform will be Android to allow for hands-free voice control input.

Note that this project is not concerned with the real time digital processing required to convert speech to text; it is assumed that the speech from the microphone is already converted to a text string using an existing tool such as Google's Speech-to-Text API. The focus is on the integration and usability of voice control schemes.

The proposed design for the RPG and system aims to achieve the following goals:

- Allow a developer to add voice recognition to their game with little effort.
- Create a simple natural language processing system that can work offline.
- Explore NLP usages in RPGs (for text generation, item descriptions, etc).

3 Background

3.1 Voice Recognition in Games

Below is a list of notable recent games that have incorporated voice control functionality as a major component of the interaction experience.

3.1.1 In Verbis Virtus

In Verbis Virtus [2] is an independent 3D fantasy adventure game developed by Indomitus Games for Windows. It requires the player to solve puzzles in a 3D environment and battle enemies using magic.

When a microphone is connected, the player is able to cast spells using their voice by saying specific phrases defined by the game. For example, to cast a spell that produces a floating light source to brighten a room, a user must say, "let there be light", or if the user wishes to shoot an energy beam from their hand, they must say, "Beam of Light", and so. These phrases appear to be hard-coded into the game.

There are several limitations with this implementation of voice control. Firstly, since the phrases are hard-coded, there is no lenience in variations of the phrases, e.g. "create a light" or "Laser of Light", etc. This can seem mundane and no different to a user pressing the same button repeatedly to cast a spell.

Another limitation is that the user is still required to use a keyboard and mouse to perform other actions in the game, such as moving around, looking around, and navigating menus and message boxes. The voice control interface cannot be used to perform these actions. The voice control interface itself does not seem integral to the experience, as the user could just simply press a key that performs the spell instead of saying the same phrase over and over again.

This voice recognition interface could be improved by allowing the player to perform any action within the game using their voice, as well as allowing for more variation in what the user can say for each action.

3.1.2 Skyrim Kinect

The Elders Scrolls V: Skyrim is first-person action role-playing game developed by Bethesda [3]. The Xbox 360 version of the game supports the Microsoft Kinect peripheral which allows the player to execute voice commands [4]. The user can use over 200 hundred pre-configured voice commands, and these can be used simultaneously with the traditional control scheme.

There are voice commands available that allow the user to navigate the game's menu screens. For instance, the player would say, "quick items" to open their inventory menu. Afterwards, the user can open menus for specific categories of items such as "potions", "books", etc. However, it doesn't seem to be possible to actually use any of the items selected with only a voice command from the menu; this is only possible if you assign the item to a 'hotkey' (e.g. by saying, "assign health potion" once you have highlighted a potion item), and then saying "equip health potion" during the game. See the Appendix for a full list of available voice commands.

Figure 1: Voice commands that are available within the items menu in Skyrim.

This system, despite being optional to the player, helps to reduce the time spent searching through the menu screens. However, it appears that this system also hard-codes the phrases that can be said by the player with no flexibility.

3.1.3 Star Trek Bridge Crew

Star Trek Bridge Crew is a virtual reality game developed by Ubisoft. This game allows the player to issue voice commands to AI crew members on a spaceship [5]. In order to develop a more interactive and realistic experience, the development team used IBM Watson's interactive speech capabilities. This API allows commands to be delivered in a variety of ways, as the speech is parsed for its meaning using Watson's Conversation service. For example, a player could say, "show me the ship's status report" or they could say something drastically different such as "damage report"

to execute the same command [6]. This gives the user more freedom in how they convey their request to the AI, giving the player a "new level of sense of presence" [7]. IBM's Watson API is described in more detail later in this section.

The voice recognition design used in this game is similar to the proposed design outlined in this report. The user's intent should be extracted from the phrase that they speak, so similar phrases should map to the same intent. Unfortunately, Watson's API is a paid cloud service; so the game requires a constant internet connection to play.

3.1.4 Classic Zork on Alexa

Zork is a trilogy of classic text-based role-playing games that support a free-form style of input. The player is provided with an input scenario (e.g. "You are standing in an open field west of a white house, with a boarded front door. There is a small mailbox here."), and then types commands that they want to execute (e.g. "open mailbox" or "attack troll with sword") [8]. The exploration part of the proposed game in this report will be loosely based off Zork's exploration mechanics of interacting with objects in the surroundings.

These commands could be delivered using a speech-to-text interface to add voice control to Zork. A group of developers created a port of Zork running on Amazon's Alexa Skills Kit [9]. This allows the player to deliver commands using their voice as if they were typing the phrases into a terminal. However, this implementation still suffers from the limitations of Zork's free-form input.

3.2 Limitations of Zork

The exploration mechanics of the proposed design in this report will be based on Zork's general gameplay of interacting with objects in the environment to solve puzzles, and the limitations of Zork's free-form input are discussed here.

Zork supports a number of text commands, which usually take the form of a verb-noun structure, such as "use <noun>" or "take <noun>", but it also supports some more sophisticated commands with complex structures, such as "give all but the pencil to the nymph" or "drop all except the dart gun" [10].

Zork's free-form input still has some limitations. For example, it still cannot accept all variations of specific intents. For instance, below is a list of accepted and rejected commands if the user wishes to open a mailbox in front of them:

- "open mailbox" - Accepted

- "open the small mailbox" - Accepted
- "can i open the mailbox" - Rejected
- "check the mailbox" - Rejected
- "find out what's in the mailbox" - Rejected

Below is a list of results for closing the mailbox:

- "close mailbox" - Accepted
- "shut mailbox" - Rejected
- "closing mailbox" - Rejected
- "close the red mailbox" - Rejected

Clearly the rejected text commands could be valid phrases that express an intent to open some sort of mailbox. Zork has a pre-defined list of verbs and sentence structures that it can accept, and sticks to those without much flexibility.

Since most of the supported commands take the form of a verb-noun structure, this could be used in the proposed design as the general structure of commands expected by the user.

3.3 Voice Recognition Implementations

There are many tools and services available that allow developers and users to add voice recognition to video games, with some discussed below:

3.3.1 Tazti - Speech Recognition for PC Games

Tazti is a keyboard mapping tool available to players to be used with any type of PC game that uses the keyboard [11]. The user sets up a profile for each game separately by mapping speech commands to one or more keys. For instance, the user could say "fire" or "shoot fire" and this would map to the keystrokes required to cast a fire spell.

While this tool is not directly integrated into games by the developer, it has the advantage of being able to work with almost any PC game that uses a keyboard, such as role-playing games, first-person shooter games, and even platforming games. However, the voice commands are limited to being hard-coded, so there is no way of varying the speech commands slightly, even if the meaning is the same. It is also not freely available and requires a license to be purchased.

3.3.2 PocketSphinx for Unreal Engine 4

Sphinx-UE4 is a speech recognition plugin for the Unreal Engine 4 game engine [12], based on the PocketSphinx library which provides offline speech-to-text [13]. The plugin allows the developer to specify keywords and the commands that they map to (e.g. "turn right", "enable sprint", "kick the ball", etc).

The plugin also supports grammar files. A grammar contains the grammatical structure that an accepted input can take, such as, $< digit >< operation >< digit >$. This will accept input such as, "three add five", "six minus four", "two times twelve", and so on.

3.3.3 Houndify

Houndify is a paid cloud service developed by SoundHound that allows developers to add voice recognition and control to any application they wish [14]. The API takes as input either a text string or audio samples and returns a JSON string containing the response to the input. This response can range from anything: from home automation control, to weather updates, and more. The response would be in the form of an intent - a single word representing the action to be executed. For example, any attacking phrase would map to an intent called, "ATTACK".

As well as having built-in voice commands already for specific domains (such as all weather commands or sports update commands), Houndify also supports the creation of custom voice commands by the developer. To create a custom command, the expression to be said by the user is specified using a syntax similar to regular expressions, but instead specifies the general phrase structure (and with very different syntax), and is mapped to an intent [15].

Below is an example of a possible expression for attacking a troll:

"attack" . "troll"

Here, the player would have to say, "attack troll" to execute the intent. This can be expanded to have more variation in how the user can say this command:

("attack" | "hit") . ["the"] . "troll"

Here, the player can say phrases such as, "attack the troll" or "hit troll". The expression can be further expanded. However, the expression will eventually become long and confusing:

("attack" | "hit" | "damage") . ["the"] . ["big" | "large"] . "troll" . [("with" | "using") . ["a"] . ("sword" | "hammer" | "axe")]

Even though this expression allows for more variation in what the user can say, it still doesn't cover all the possible variations of specifying an attack, and this long expression is only for one intent; many more expressions would have to be written for other intents, and this can become tedious for the developer.

While this API would be useful for building a personal assistant similar to Apple's Siri or Amazon's Alexa, it does not seem feasible for this project due to the time required to create a flexible voice recognition scheme. This is also a paid service, so users are charged for each query they make.

3.3.4 Watson Conversation Engine

IBM's Watson API features a Conversation service that allows developers to build voice recognition interfaces that understand natural language input [16]. The developer defines the training data to be used by the API (in order to train a natural language classifier) using 'intents' and 'entities' [17].

'Intents' are the goals that a user will have as they interact with the application. For example, in a voice-controlled car, an intent could be 'wipers_on', in order to activate the windscreen wipers. This intent would be paired with example utterances in the form of text strings, such as "turn on the wipers", "switch on the windscreen wipers", and so on. The more examples given, the more accurate the trained model will be.

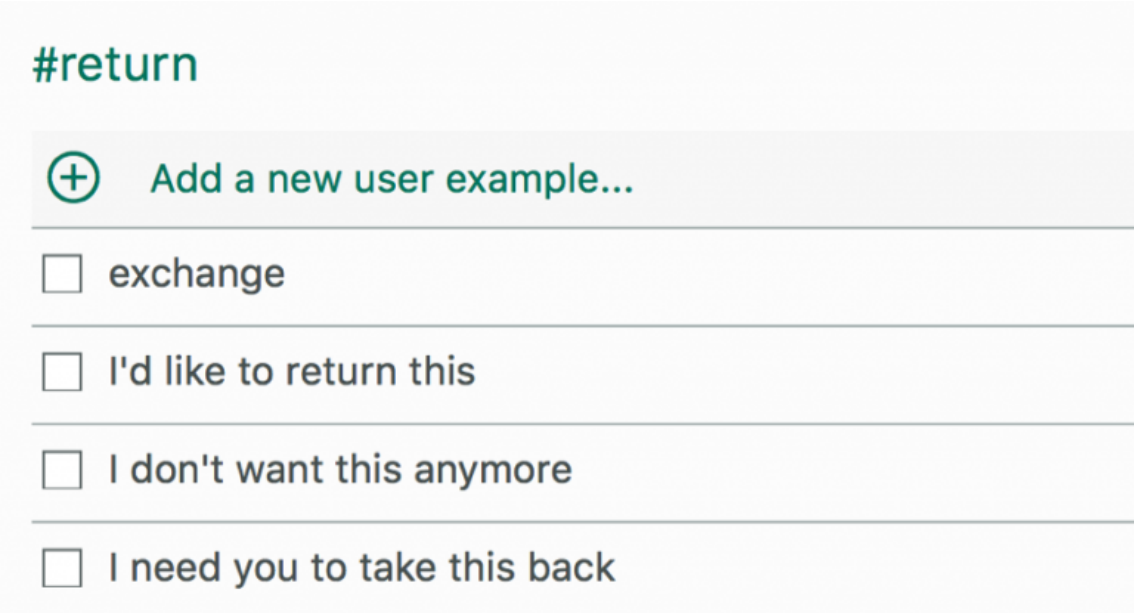


Figure 2: An example of an intent to return something, along with examples of phrases for the intent. Image re-used from an IBM blog post [18].

'Entities' are classes of objects that help to provide context to intents. For example, in the above example, it is not clear whether the user is referring to the wipers on the front of the car or the wipers on the back of the car (the API will assume the front by default). If the entity is included in the phrase spoken by the user, then the context of the intent becomes clear.

Figure 3: An example of entities for items that can be returned. Entities that have been grouped together (usually synonyms) are placed on the same row [18].

Once the classifier is trained, it will be able to use the examples of intents provided to classify whether new examples of phrases have the same meaning as those. For example, if the user says, "activate the windscreen wipers at the front of the car", the classifier will return the 'wipers_on' intent, even though this input is very different in structure to the examples provided above. A demo is available to test this [19].

This API is very flexible and matches the first objective of this project; only a few example of possible inputs need to be provided for each intent, and then the classifier can infer whether any new input strings will match the same intent. Unfortunately, despite being a very powerful system, this still remains a paid cloud service, so users are charged per API call. It also requires a lot of processing power (which is offloaded to a cloud server) for the many machine learning algorithms being used here. However, a simpler system may possibly be implemented that uses a similar concept to the intents and entities framework, and the proposed design in this report uses a similar concept that runs locally.

There is a similar API called DialogFlow [20], which works very similarly to IMB's Watson Conversation engine. However, it suffers from the same issues as above (e.g. cloud-based, pay-per-request, etc).

3.4 Natural Language Processing

Natural Language Processing (NLP) forms a major component of this project. This will give the developer the ability to take an arbitrary string of text and infer its meaning, and map it to an intent. Below are some explanations on NLP concepts related to this project.

3.4.1 Stages of Natural Language Processing

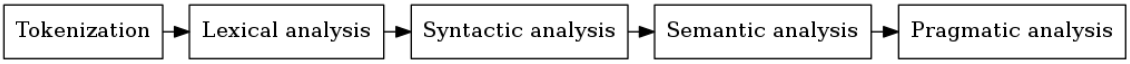


Figure 4: The general stages of processing a text string for its meaning using NLP.

NLP can be broken down into six stages [21]. The first stage is the tokenisation, where a raw text string is broken down into words (usually separated by spaces).

The second stage is the lexical analysis, where the output of the tokenisation process is improved upon by looking at words that can be taken apart even further (or rejoined if needed) to uncover more information. These include words such as:

clictic contractions (e.g. "what're" would tokenise to "what" and "are",. while "we're" would break down to "we" and "are"); removing end-of-line hyphens that split whole words into parts when using a justification alignment; and abbreviations (e.g. Dr., U.S.A., etc.) [22].

The third stage is syntactic parsing, where the grammatical structure of the sentence is determined. This is usually achieved by generating a syntax tree of each sentence, where each word is broken down into a terminal from a given grammar (e.g. verbs, nouns, adjectives, determiners, etc.) [23].

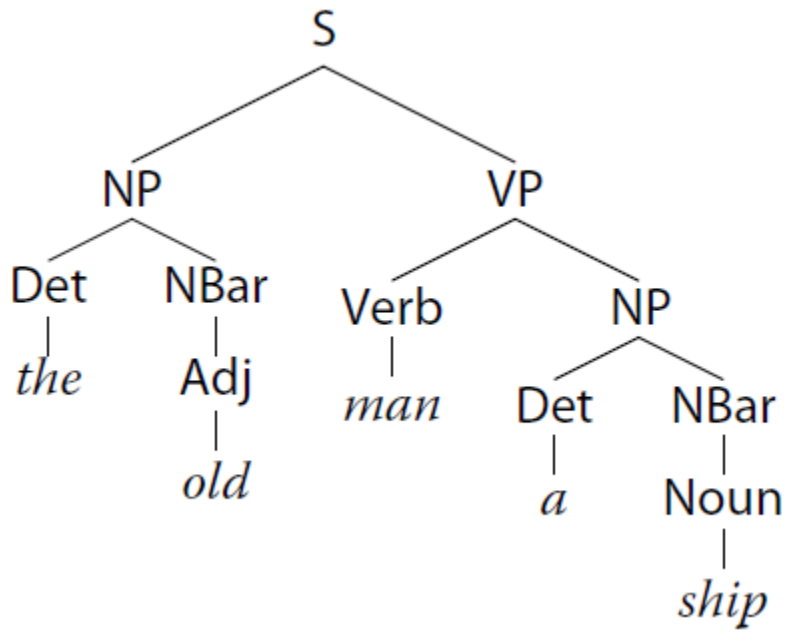


Figure 5: An example of a syntax tree for the sentence. "the old man a ship". Diagram taken from the Handbook of Natural Language Processing [23].

In the example above for the sentence, "the old man a ship", the word, "man" could be mistaken to be a noun, but in this case, the sentence is implying that the elderly control a ship, and therefore, "man" is actually a verb here. This meaning can be more easily inferred when the sentence is broken down using a syntax tree, as above.

The final two stages are very similar and is difficult to separate them into separate stages. Both stages involve determining the meaning of the sentence. The fourth stage is often concerned with semantic analysis, whereas the last stage is more concerned with discourse analysis [22].

3.4.2 Part-of-Speech Tagging

A technique often used in NLP is to try to label each word with its correct part of speech, known as part-of-speech (POS) tagging. POS-tagging systems generally use a tagset containing POS tags to assign a tag to each word. The most popular tagset is the Penn Treebank tagset. This consists of 48 tags, of which 12 are for punctuation and other symbols [24]. See the Appendix for the full Penn Treebank tagset.

There are two main challenges regarding POS tagging. The first challenge is that words can sometime be ambiguous. That is, the word can be assigned a number of possible POS tags depending on its context. For instance, the word, "can" can either be a verb ("I can..."), a noun ("a can of tuna"), or another verb with a completely different meaning to the previous ("can that noise!") [25]. The many meanings that are attributed to a word are known as its word senses.

The second challenge are words that are unknown to the tagger and cannot be tagged. A default tag is required for words which are unknown, but that could interfere with the rest of the POS-tagging process [24].

The proposed design for this project will use an open-source POS tagger in order to identify specific words in the player's input, namely verbs and nouns.

3.4.3 Slot Filling

According to the authors of *Speech and Language Processing* [26], three tasks need to be done to understand a user's utterance to a chatbot. The first is *domain classification*, to determine the topic that the user is talking about (e.g. weather, sport, etc), although this is unnecessary for single-domain applications such as the environment of a video game. The second task is *intent determination* to determine the goal of the user, and finally, the last task is to do *slot filling*.

Slot filling involves determining the specific details of an intent by defining a grammar that the intent must adhere to. An example for arithmetic operations would be, $< DIGIT1 > < OPERATION > < DIGIT2 >$. The input would be parsed by a context-free grammar parsing algorithm to extract each bit of information in order to fill each slot in the grammar.

A disadvantage to slot-filling using context-free grammar parsing is that not all valid expressions for an intent would match with the slots. A solution to this would be to define multiple grammars for different structures of inputs.

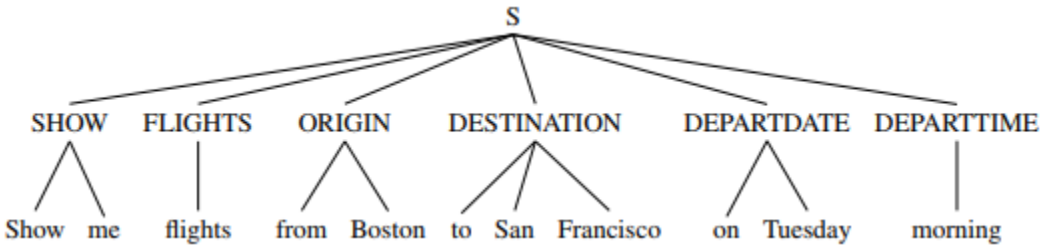


Figure 6: An example of a slot-filling grammar for flights, using slot names as the internal parse tree nodes. Diagram taken from Chapter 29 of the book, *Speech and Language Processing* [26].

3.4.4 Synonyms and Hypernyms

Synonyms are words or phrases which have a similar meaning to another word [27]. For example, the word "dog" has seven word senses (different meanings), and for one of the word senses with a meaning of, "a member of the genus Canis", the synonyms would be, "domestic dog" and "Canis familiris" [28].

A hypernym, on the other hand, is a word or phrase which generalises words for which it is a hypernym of. For instance, the word, "instrument" is a hypernym of "guitar", because a guitar is a type of instrument. Similarly, the word "animal" is a hypernym of "cat" (or, at least, the sense which refers to the domestic animal, as "cat" can have several usages).

The synonyms and hypernyms of words and phrases can be represented as a hypertree structure, where the parent nodes are hypernyms of the child nodes, while each sibling node would represent a different word sense for a word, with each sibling containing the synonyms for that sense. Note that a hyponym is the inverse relationship between a word and its hypernyms.

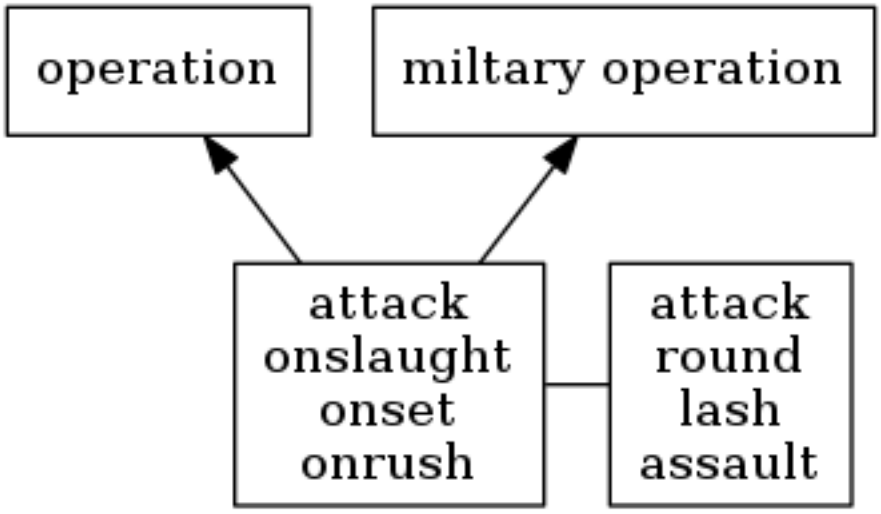


Figure 7: A partial hypernym tree showing the synonyms and hypernyms for one of the senses of the noun, "attack". The "attack" verb sense (right) is a sibling node.

Synonyms and hypernyms could be used to determine the semantic similarity between words and infer whether they match a certain intent of the user.

3.4.5 Semantic Similarity

There are many ways of calculating the semantic similarity between two words or phrases. Many of these methods use the synonyms and hypernym trees of the words to infer similarity, and assign a 'similarity score' to them. This score could be used to determine whether, for instance, a verb has semantic similarities to one of the intents in a game, and therefore can be mapped to that intent.

The following are some examples of measures for semantic similarity [29]. The first is the Wu and Palmer (WUP) measure, which takes the depth in the hypernym tree of the Least Common Subsummer (LCS) (that is, the most general hypernym that both words have in common) and compares it with the sum of the depths of each word. For example, the WUP score for the semantic similarity between "juice" and "water" is calculated as follows [30]:

```
T1 = HypernymTree( juice ) =
    [Sense #1] *ROOT* < entity < physical_entity < matter
        < substance < food < foodstuff < juice

T2 = HypernymTree( water ) =
    [Sense #3] *ROOT* < entity < physical_entity < matter
        < substance < food < water
```

```
Lowest Common Subsumer(s) = argmax(depth(subsumer(T1,T2)))
    = { subsumer(T1[1], T2[3]) } = { food }

DepthLCS = depth( food ) = 6
Depth1 = min(depth( {tree in T1 | tree contains LCS } )) = 8
Depth2 = min(depth( {tree in T2 | tree contains LCS } )) = 7
Score = 2 * DepthLCS / ( Depth1 + Depth2 ) = 2 * 6 / (8 + 7)
Score = 0.8
```

The WUP score of 0.8 out of 1.0 indicates that juice and water are semantically very similar [31], whereas the words "breathing" and "fire" only have a WUP score of 0.47, suggesting they are not semantically similar.

Another measure of semantic similarity is Leacock and Chodorow's method, which relies on the shortest distance between the two words in the hypernym tree, divided by the maximum depth of the whole tree; the shorter the distance, the more similar the words are [29]. There are many more different measures and it is possible to combine them to produce an overall score for semantic similarity.

3.4.6 Generating Sentences

One feature for the proposed design of the RPG is to be able to generate text-based descriptions of the current situation that the player is in based on objects around them. For instance, a description could be, "You are in a room lit by a candle on a table. There is a piece of broken glass on the floor and a door in front of you." One could hard-code this sentence for each room the player visits, but if there are a lot of rooms in the game, this could become time-consuming. We would like to generate a sentence of the current situation based on sparse pieces of information (such as the items in the room, plus their locations, in a vector representation). This is the opposite of the intent inference described above, where the meaning of a sentence/phrase is extracted; we would like to be able to go in the opposite direction.

This is a relatively new field in NLP as it is particularly difficult. One method of sentence generation is suggested by Iyyer et al [32], where recursive autoencoders (a type of neural network) are used to generate a decomposition model of the sentence that can be used to reconstruct the sentence again. However, they found that proper nouns were not reconstructed correctly, and the model would need to be built in the first place to reconstruct the sentence.

Another group of researchers proposed a method of sentence generation which involves taking a sum of word embedding vectors and modelling it as a mixed integer

programming problem [33]. However, this also requires the original sentence to be de-constructed in the first place in order to know what the vector should contain.

One possible solution would be to only hard-code the general sentence structure of the description of rooms, and then insert the appropriate adjectives and nouns of the items, which is given as input in the form of a vector of strings.

3.5 WordNet

Princeton University created a large lexical database for the English language [34]. Words are grouped into synonyms (known as synsets), and are connected to their respective hypernyms (more general words) and hyponyms (more specific words), creating the large hypernym tree mentioned previously. The database is open-source and is used extensively in this project, using various APIs that interact with the WordNet database (which will be initially downloaded by the game for offline use).

3.6 Object Properties

One issue to note is when a user attempts to specify an item based on a description of it (e.g. the user says, "attack the enemy with a sharp object"); we would like to be able to select a item that matches the description (e.g. a sword is sharp, so use that if the user possesses one). One approach to this is to assign to the items a set of properties such as whether it is sharp or blunt, whether it can be thrown, and so on.

A similar technique was used by the developers of Scribblenauts for the Nintendo DS. Using the company's ObjectNaut engine, the game utilises a large database of a hierarchy of objects [35]. This allows them to design sprites for different objects (such as a cyborg, robot or android), but they have the same interaction properties. However, it still required the developers to go through encyclopaedias word-by-word. Using the semantic similarity scoring mechanism, this would not need to be done, as we can infer if two objects would have the same properties instead of hard-coding each one.

By assigning items different properties, we don't have to code the actions the user can perform for each variation of items (e.g. sword, katana, knife, etc.). Instead, we can have general objects which have certain properties, and then customise the actions for each general object. For example, there could be a general object with a 'sharpness' property, that would be able to, for example, cut something, and all swords and knifes would be of this type of general object.

4 Design

4.1 Target Platform

According to Ovum's Mobile Games Market forecast for 2017-2022, mobile games continue to increase their market share as they grow in popularity, and in two year's time almost half of the total video game revenue generated will come from mobile games. Therefore, it makes sense to design a system for a mobile platform such as iOS or Android.

The chosen platform to develop the voice recognition system and game will be Android mobile devices, as the availability of the open-source Google Speech-to-Text API means that less effort is required to get the text input from the user's utterance.

4.2 Java Programming Language

The language used to develop the application is Java, which is an object-oriented programming (OOP) language. The use of an OOP language makes it easier to create objects in a video game using the inheritance hierarchy. For example, there would be an abstract class for physical objects called `PhysicalObject`, and another class such as `Glass` would be a derived class of a PhysicalObject, and so on.

A disadvantage to using Java is that it does not support multiple inheritance, unlike other languages with OOP constructs such as C++, due to methods such as `super()` having the same signature for all parent classes. For instance, a Glass class cannot inherit properties from both a PhysicalObject and a BreakableObject (i.e. they are both direct parents of the Glass class). Instead, this must be overcome by making the inheritance a single chain (e.g. Glass inherits from BreakableObject which inherits from Physical Object). While this can be considered a hindrance in some circumstances, it is not a major issue for the proposed design of the game.

While Java applications - and therefore Android - support the invocation of native code such as C++ using the Java Native Interface (JNI) for improved performance, this will not be used due to time constraints for the project (as performance is not a major objective for this project).

4.3 System Overview

The following is a brief description of the voice recognition system, including the main components and the general flow of data.

The voice recognition system takes as input the raw microphone input contain-

ing the player's utterance, and outputs the execution of a developer-defined action based on the player's utterance. Figure 8 shows a simple flow chart outlining the main stages of the voice recognition system form input to output.

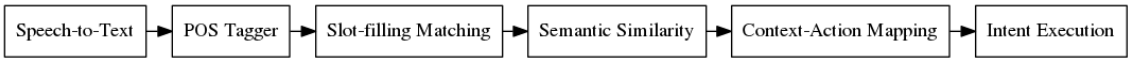


Figure 8: A flow chart showing the general stages of the voice recognition system. Graph generated using Graphviz.

The audio from the microphone input is processed into a text string using a Speech-to-Text (STT) engine.

4.4 System UML Class Diagram

Figure 9 shows a simplified UML class diagram focusing on the classes used for the voice recognition system, including the inheritance and associations of the classes. Although the `GameState` class is related to the game system, it helps to illustrate the overall system design. Only the important member fields and methods are shown for each class.

See the Appendix for the full UML class diagram of the Android application.

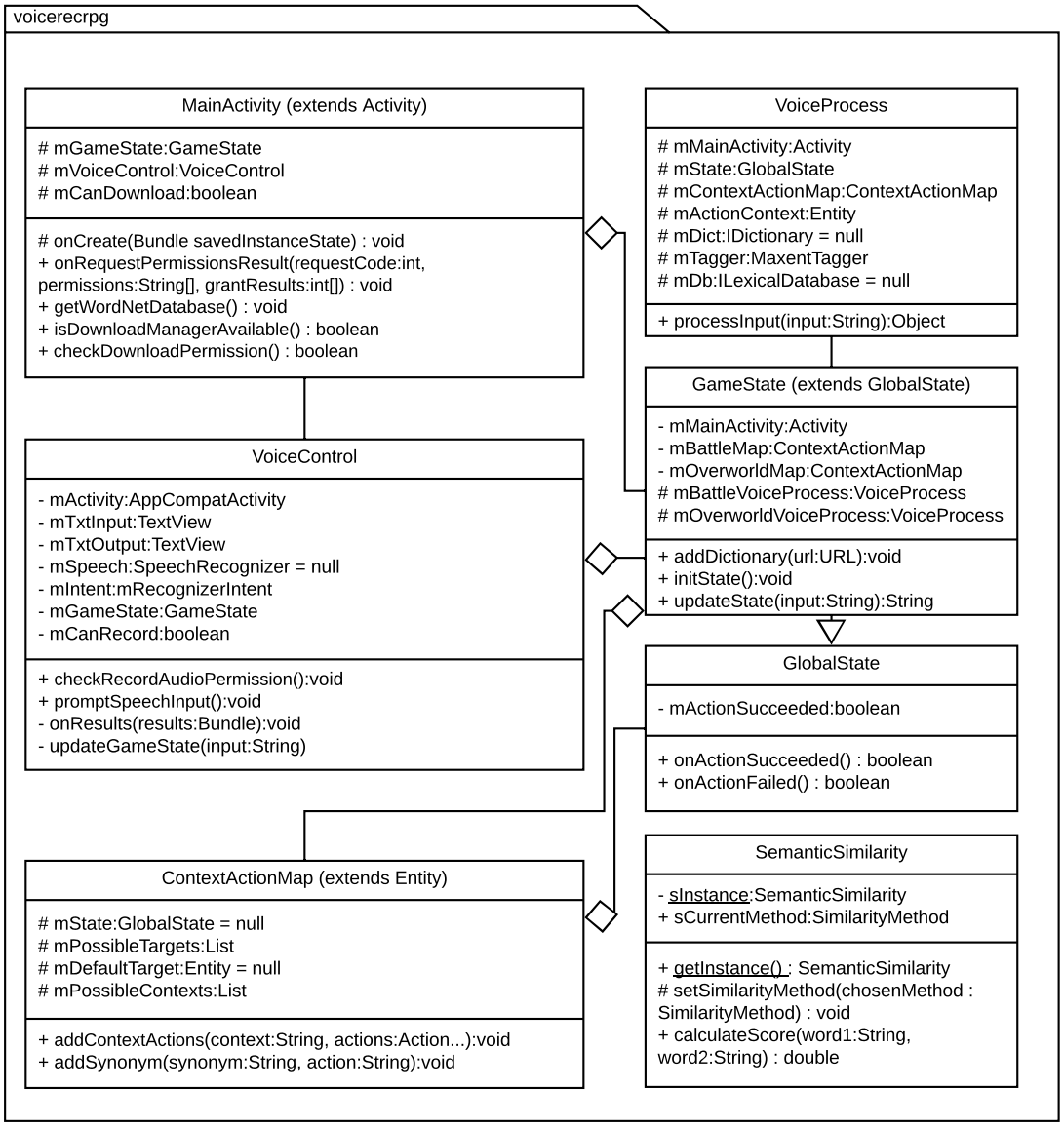


Figure 9: A simplified UML class diagram outlining the design of the voice recognition system. Diagram created using the online LucidChart tool. See the Appendix for the full class diagram for the entire voice recognition system in the Android application.

4.4.1 `MainActivity` Class

The `MainActivity` class is the skeleton class which contains all the systems. It acts as the interface between the I/O and the rest of the application. It contains instances of the `VoiceControl` class, which handles the I/O, and the `GameState`,

which contains the game environment.

When the application starts up, an instance of `MainActivity` is created, and searches the public directory of the phone's external storage (not necessarily an external device like an SD card, but whatever the OS has has defined the external storage to be). It searches for a WordNet database that is downloaded beforehand, and if it is not found, it will download/re-download the database again. It will also copy a model used for POS tagging to the external storage from the APK archive.

4.4.2 `VoiceControl` Class

As mentioned, the `VoiceControl` class is used to handle the input and output of the application, such as the microphone input and the text display. It contains an instance of the SpeechRecognizer class, which is provided by Android and uses Google's Speech-to-Text API, aa well as two instances of the TextView class: one which is used to display the text output of the game, and the other to display the Speech-to-Text result at the top of the screen.

When the `SpeechRecognizer` instance performs a Speech-to-Text translation, the final result is forwarded to the `GameState` instance where it is processed by the voice recognition system using NLP, and used to update the state of the current game session (`GameState`).

4.4.3 `VoiceProcess` Class

The `VoiceProcess` class is responsible for processing the string input containing the player's utterance that is received from teh `VoiceControl` class. It contains the bulk of the NLP processing described later.

4.4.4 `GlobalState` Class

The `GlobalState` class is a generic abstract class that can be derived from in order to pass around environment objects between method calls easily. For example, if a method wishes to access an instance of an `Inventory` class (to access the game's items), then a reference to the inventory can be held in a class derived from `GlobalState` (e.g. `GameState`).

4.4.5 `ContextActionMap` Class

The `ContextActionMap` class contains the mapping of contexts to actions. It is an abstract class that is overloaded by the developer to add mappings of different ways of performing actions based on the given context (e.g. "attack with a sword" versus "attack with a knife").

4.4.6 `Semantic Similarity` Class

5 Appendix

5.1 Skyrim Kinect Command List

Below is a list of all 200 commands that can be used in the video game, The Elder Scrolls V: Skyrim, when a Microsoft Kinect peripheral is connected.

The Elder Scrolls V

SKYRIM

KINECT VOICE COMMANDS

To further enhance your adventures, we've added more than 200 Voice Commands available to players using Kinect. Learn how to harness the power of the Thu'um and other Voice Commands using the capitalized words in the guide below.

When Kinect is Disabled
: Shout/Powers

When Kinect is enabled:
: Enabled dragon language shouts
+ : Shout/Powers

MAIN GAMEPLAY

Only usable during main gameplay.

QUICK ITEMS Opens Inventory Menu	QUICK MAGIC Opens Magic Menu	QUICK SKILLS Opens Skills Menu	QUICK MAP Opens Map Menu
QUICK SAVE Quick-saves	QUICK LOAD Quick-loads	NEW SAVE Saves into a New Save Slot	PAUSE GAME Pauses the game

SHOUTS

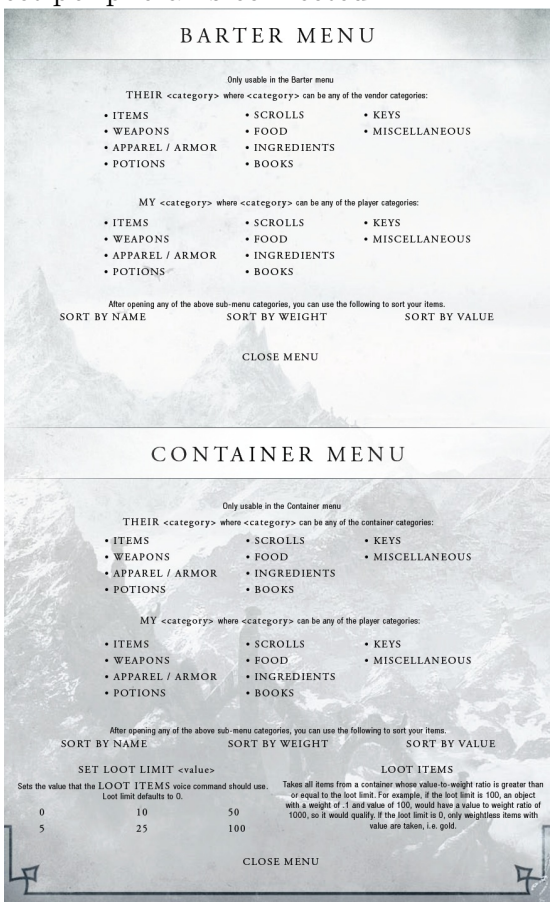
Only usable during main gameplay. There are two things to remember for Kinect shouts.

First, you need to have the shout unlocked to the level that you want to use before the Kinect will allow you to do it, i.e., if you haven't unlocked Fire Breath, saying the shout in English or Dragon Language will do nothing.

Second, you can say the Shout in English or Dragon Language.

1. **English Language.** Available without holding . You can say the name of the shout, e.g UNRELENTING FORCE, which will trigger the highest known level of the shout.
2. **Dragon language.** Only available when holding . You can say one, two, or three words in the dragon language of the shout which will trigger the corresponding level of shout.

UNRELENTING FORCE or FUS RO DAH	DISMAY or FAAS RU MAAR	BECOME ETHEREAL or FEIM ZII GRON	WHIRLWIND SPRINT or WULD NAH KEST
ELEMENTAL FURY or SU GRAH DUN	CLEAR SKIES or LOK VAH KOOR	FIRE BREATH or YOL TOOR SHUL	DRAGONREND or JOOR ZAH FRUL
CALL OF VALOR or HUN KAAL ZOOR	CALL DRAGON or OD AH VIING *(No 1st or 2nd level variations of this shout)*	ANIMAL ALLEGIANCE or RAAN MIR TAH	AURA WHISPER or LAAS YAH NIR
DISARM or ZUN HAAL VIIK	MARKED FOR DEATH or KRII LUN AUS	FROST BREATH or FO KRAH DIIN	STORM CALL or STRUN BAH QO
KYNE'S PEACE or KAAN DREM OV	SLOW TIME or TIID KLO UL	THROW VOICE or ZUL MEY GUT	ICE FORM or IIZ SLEN NUS



BARTER MENU

Only usable in the Barter menu

THEIR <category> where <category> can be any of the vendor categories:

- ITEMS
- WEAPONS
- APPAREL / ARMOR
- POTIONS
- SCROLLS
- FOOD
- INGREDIENTS
- BOOKS
- KEYS
- MISCELLANEOUS

MY <category> where <category> can be any of the player categories:

- ITEMS
- WEAPONS
- APPAREL / ARMOR
- POTIONS
- SCROLLS
- FOOD
- INGREDIENTS
- BOOKS
- KEYS
- MISCELLANEOUS

After opening any of the above sub-menu categories, you can use the following to sort your items.

SORT BY NAME SORT BY WEIGHT SORT BY VALUE

CLOSE MENU

CONTAINER MENU

Only usable in the Container menu

THEIR <category> where <category> can be any of the container categories:

- ITEMS
- WEAPONS
- APPAREL / ARMOR
- POTIONS
- SCROLLS
- FOOD
- INGREDIENTS
- BOOKS
- KEYS
- MISCELLANEOUS

MY <category> where <category> can be any of the player categories:

- ITEMS
- WEAPONS
- APPAREL / ARMOR
- POTIONS
- SCROLLS
- FOOD
- INGREDIENTS
- BOOKS
- KEYS
- MISCELLANEOUS

After opening any of the above sub-menu categories, you can use the following to sort your items.

SORT BY NAME SORT BY WEIGHT SORT BY VALUE

SET LOOT LIMIT <value>

Sets the value that the LOOT ITEMS voice command should use. Loot limit defaults to 0.

0	10	50
5	25	100

LOOT ITEMS

Takes all items from a container whose value-to-weight ratio is greater than or equal to the loot limit. For example, if the loot limit is 100, an object with a weight of .1 and value of 100, would have a value to weight ratio of 1000, so it would qualify. If the loot limit is 0, only weightless items with value are taken, i.e. gold.

CLOSE MENU

FAVORITES MENU & HOTKEY EQUIPPING

Only usable in the Favorites menu.

ASSIGN <hotkey command> sets the selected item to the spoken hotkey. <hotkey command> can be any of the following:

- HEALTH POTION
- MAGICKA POTION
- STAMINA POTION
- POISON
- SWORD
- MACE
- AXE
- DAGGER
- BOW
- SHIELD
- DUAL WIELD LEFT
- DUAL WIELD RIGHT
- SOUL TRAP
- GREATSWORD
- BATTLEAXE
- WARHAMMER
- FIRE SPELL
- FROST SPELL
- LIGHTNING SPELL
- WARD SPELL
- RITUAL SPELL
- BOUND WEAPON
- SUMMON SPELL
- ARMOR SPELL
- CALM SPELL
- FRENZY SPELL
- HEALING SPELL
- LIGHT

Only usable during main gameplay.

EQUIP <hotkey command>
Equips item in the default hand.

EQUIP LEFT <hotkey command>
Equips item in the Left hand, if possible.

EQUIP RIGHT <hotkey command>
Equips item in the Right hand, if possible.

EQUIP DUAL <hotkey command>
Equips the item in both hands, if possible.

EQUIP SWORD AND SHIELD/ EQUIP MACE AND SHIELD/ EQUIP AXE AND SHIELD/ EQUIP DAGGER AND SHIELD
Equips the item assigned to the Sword / Mace / Axe / Dagger hotkey in the right hand, and the item assigned to the Shield hotkey in the left hand.

EQUIP DUAL WEAPONS
Equips the item assigned to the DualWieldLeft hotkey in the left hand, and the item assigned to the DualWieldRight hotkey in the left hand.

FOLLOWER COMMANDS

ALLY <action command>
Tells the ally to activate the thing you're looking at, as if you entered Command Mode and selected it.

FOLLOW / FOLLOW ME
Ally follows you

WAIT / WAIT HERE
Ally stays where they are

TRADE / ITEMS
Opens the Teammate Menu

OPEN
Ally opens the container/door you're looking at

STAND
Ally stands at the spot you're looking at

RETRIEVE
Ally picks up the item you're looking at

ATTACK
Ally attacks the NPC you're looking at

INTERACT / USE
Does a generic Interact command with the ally. Same as if you entered Command mode and pressed A. What the ally does is dependent on what you're looking at.

ITEMS MENU

After opening ITEMS, the following commands open the corresponding categories:

- FAVORITES
- ALL
- WEAPONS
- APPAREL / ARMOR
- POTIONS
- SCROLLS
- FOOD
- INGREDIENTS
- BOOKS
- KEYS
- MISCELLANEOUS

After opening any of the above sub-menu categories, you can use the following to sort your items.

SORT BY NAME
Sorts the item list by name, increasing. If the list is already sorted by name increasingly, sorts it decreasingly.

SORT BY WEIGHT
Sorts the item list by weight, decreasing. If the list is already sorted by weight decreasingly, sorts it increasingly.

SORT BY VALUE
Sorts the item list by value, decreasing. If the list is already sorted by value decreasingly, sorts it increasingly.

CLOSE MENU
Closes the menu

MAGIC MENU

After opening MAGIC, the following opens the menu to the corresponding category:

- FAVORITES
- ALL
- ALTERATION
- ILLUSION
- DESTRUCTION
- CONJURATION
- RESTORATION
- SHOUTS
- POWERS
- ACTIVE EFFECTS / EFFECTS

CHARACTER MENU

After opening CHARACTER, the following opens the menu to the corresponding category:

- ITEMS
- MAGIC
- SKILLS
- MAP

MAP MENU

After opening MAP, the following centers the camera on the corresponding locations:

- WINDHELM
- FALKREATH
- DAWNSTAR
- MORTHAL
- SOLITUDE
- MARKARTH
- RIFTEN
- WHITERUN
- WINTERHOLD

PLAYER / WHERE AM I
Centers camera on you.

WAYPOINT
Centers camera on marker set by you, if applicable.

QUEST MARKER / QUEST
Centers camera on active quest target. Saying it multiple times will cycle through targets.

SKILLS MENU

After opening SKILLS, use the following commands to center the camera on that skill's perk tree:

- ONE HANDED
- TWO HANDED
- ARCHERY
- BLOCK
- SMITHING
- HEAVY ARMOR
- LIGHT ARMOR
- PICKPOCKET
- LOCKPICKING
- SNEAK
- ALCHEMY
- SPEECH
- ALTERATION
- CONJURATION
- DESTRUCTION
- ILLUSION
- RESTORATION
- ENCHANTING

5.2 Penn Treebank Tagset

Below is a table of the 32 tags from the Penn Treebank tagset, used for POS tagging. The 12 remaining tags for punctuation and other symbols have been left out [36].

CC	Coordinating conjunction	TO	to
CD	Cardinal number	UH	Interjection
DT	Determiner	VB	Verb, base form
EX	Existential there	VBD	Verb, past tense
FW	Foreign word	VBG	Verb, gerund or present participle
IN	Preposition or subordinating conjunction	VBN	Verb, past participle
PRP$	Possessive pronoun	NNS	Noun, plural
RB	Adverb	NNP	Proper noun, singular
RBR	Adverb, comparative	NNPS	Proper noun, plural
RBS	Adverb, superlative	PDT	Predeterminer
RP	Particle	POS	Possessive ending
SYM	Symbol	PRP	Personal pronoun
JJ	Adjective	VBP	Verb, non-3rd person singular present
JJR	Adjective, comparative	VBZ	Verb, 3rd person singular present
JJS	Adjective, superlative	WDT	Wh-determiner
LS	List item marker	WP	Wh-pronoun
MD	Modal	WP$	Possessive wh-pronoun
NN	Noun, singular or mass	WRB	Wh-adverb

5.3 System UML Class Diagram

Below is the UML class diagram for the voice recognition system used in the Android app. This was automatically generated using the third-party *simpleUML* plugin for Android Studio.

References

[1] Marco Tabini, "Inside Siri's brain: The challenges of extending Apple's virtual assistant," 2013, April 8. [Online]. Available: https://www.macworld.com/article/2033073/inside-siris-brain-the-challenges-of-extending-apples-virtual-assistant.html

[2] Indomitus Games, "In Verbis Virtus." [Online]. Available: http://www.indomitusgames.com/in-verbis-virtus

[3] Bethesda Softworks LLC, "The Elders Scrolls Official Site," 2018, January 1. [Online]. Available: https://elderscrolls.bethesda.net/en/skyrim

[4] NowGamer, "Skyrim Kinect: Full List Of 200 Voice Commands," 2012, May 1. [Online]. Available: https://www.nowgamer.com/skyrim-kinect-full-list-of-200-voice-commands/

[5] Ubisoft, "Star Trek Bridge Crew," 2017. [Online]. Available: https://www.ubisoft.com/en-gb/game/star-trek-bridge-crew/

[6] Chris Watters, "Star Trek: Bridge Crew Integrates IBM Watson For Voice Commands," 2017, June 22. [Online]. Available: https://news.ubisoft.com/article/star-trek-bridge-crew-integrates-ibm-watson-voice-commands

[7] developerWorks, "Star Trek: Bridge Crew now live with IBM Watson Speech Technology," 2017, June 22. [Online]. Available: https://developer.ibm.com/tv/dwnewsblast-watson-star-trek/

[8] Rob Lammle, "Eaten by a Grue: A Brief History of Zork," June 2014. [Online]. Available: http://mentalfloss.com/article/29885/eaten-grue-brief-history-zork

[9] Jacob Foster and Matt Thompson and Myles Loffler, "Classic Zork on Alexa," Dec 2016. [Online]. Available: https://www.hackster.io/devops-dungeoneers/classic-zork-494ff1

[10] Dave Lebling and Marc Blank, *Zork Trilogy Instruction Manual.* InfoCom, 1984. [Online]. Available: http://infodoc.plover.net/manuals/zork1.pdf

[11] VoiceTechGroup, "tazti — Speech Recognition for PC Games," 2018, January 1. [Online]. Available: https://www.tazti.com/speech-recognition-software-for-pc-games.html

[12] Epic Wiki, "Speech Recognition Plugin," 2018, March 17. [Online]. Available: https://wiki.unrealengine.com/Speech_Recognition_Plugin

[13] CMU Sphinx, "PocketSphinx," 2015. [Online]. Available: https://github.com/cmusphinx/pocketsphinx

[14] SoundHound, "Houndify," 2017. [Online]. Available: https://www.houndify.com/

[15] ——, "The ClientMatch JSON Format," 2017. [Online]. Available: https://docs.houndify.com/reference/ClientMatch

[16] IBM, "Conversation - IBM Cloud," 2017, December 12. [Online]. Available: https://console.bluemix.net/catalog/services/conversation

[17] "IBM Cloud Docs — Conversation — About," 2018, January 5. [Online]. Available: https://console.bluemix.net/docs/services/conversation/index.html

[18] Zach Walchuk, "Build a chatbot in ten minutes with Watson," Dec 2016. [Online]. Available: https://www.ibm.com/blogs/watson/2016/12/build-chatbot/

[19] IBM, "Conversation Demo," 2017, December 18. [Online]. Available: https://conversation-demo.ng.bluemix.net/

[20] "Dialogflow." [Online]. Available: https://dialogflow.com/

[21] Nitin Indurkha and Fred Damerau, *Classical Approaches to Natural Language Processing*, 2 ed., ser. Handbook of Natrual Language Processing. United States: Chapman and Hall, 2010, pp. 4–6.

[22] Craig Trim, "The Art of Tokenization," Jan 2013. [Online]. Available: https://ibm.com/developerworks/community/blogs/nlp/entry/tokenization

[23] Nitin Indurkha and Fred Damerau, *Classical Approaches to Natural Language Processing*, 2 ed., ser. Handbook of Natrual Language Processing. United States: Chapman and Hall, 2010, p. 61.

[24] ——, *Part-of-Speech Tagging*, 2 ed., ser. Handbook of Natrual Language Processing. United States: Chapman and Hall, 2010, pp. 205–206.

[25] Dictionary.com2018, "The Dictionary of American Slang," 2018, January 18. [Online]. Available: http://www.dictionary.com/browse/can

[26] D. Jurasky and J. Martin, *Dialog Systems and Chatbots*, ser. Speech and Language Processing, 2017. [Online]. Available: https://web.stanford.edu/ jurafsky/slp3/29.pdf

[27] WikiDiff, "Synonym vs Hypernym - What's the Difference?" [Online]. Available: https://wikidiff.com/synonym/hypernym

[28] Princeton University, "Dog — WordNet Search - 3.1," 2018, January 18. [Online]. Available: http://wordnetweb.princeton.edu/

[29] Bridget McInnes and Ted Pedersen, "Evaluating measures of semantic similarity and relatedness to disambiguate terms in biomedical text," *Journal of Biomedical Informatics*, 2013, Sep 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1532046413001238

[30] Sagar Gole, "Words similarity/relatedness using WuPalmer Algorithm - Sagar Gole's Blog," Jun 2015. [Online]. Available: http://blog.thedigitalgroup.com/sagarg/2015/06/10/words-similarityrelatedness-using-wupalmer-algorithm/

[31] Hideki Shima, "Juice, Water — WS4J Demo," 2018, January 18. [Online]. Available: http://ws4jdemo.appspot.com

[32] Mohit Iyyer and Jordan Boyd-Graber and Hal Daume III, "Generating Sentences from Semantic Vector Space Representations," 2014.

[33] L. White and R. Togneri and W. Liu and M. Bennamoun, "Modelling Sentence Generation from Sum of Word Embedding Vectors as a Mixed Integer Programming Problem," in *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, 2016, pp. 770–777, ID: 1.

[34] Princeton University, "About WordNet," 2013, January 9 2013. [Online]. Available: http://wordnet.princeton.edu

[35] Mark Bozon, "World Debut: Scribblenauts," 2008, December 5. [Online]. Available: http://uk.ign.com/articles/2008/12/05/world-debut-scribblenauts

[36] University of Pennsylvania, "Penn Treebank P.O.S. Tags," 2018. [Online]. Available: https://ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html