

401I - Final Year Project

Interim Report

Voice Recognition RPG

Baron Khan (bak14)

Contents

1	Introduction	1
2	Background	3
2.1	Voice Recognition in Games	3
2.1.1	In Verbis Virtus	3
2.1.2	Skyrim Kinect	3
2.1.3	Star Trek Bridge Crew	4
2.1.4	Classic Zork on Alexa	5
2.2	Limitations of Zork	5
2.3	Voice Recognition Implementations	6
2.3.1	Tazti - Speech Recognition for PC Games	6
2.3.2	Houndify	7
2.3.3	Watson Conversation Engine	8
2.4	Natural Language Processing	9
2.4.1	Stages of Natural Language Processing	9
2.4.2	Part-of-Speech Tagging	10
2.4.3	Synonyms and Hypernyms	10
2.4.4	Exploring Hypernym Trees	10
2.4.5	Semantic Similarity	10
2.4.6	RPG Sentence Structure	10
2.4.7	Object Properties and Relations	10

1 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 speaks 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. The hardware running the game requires a microphone that always listens to the user (or detects a 'hot-word' phrase in order to begin accepting voice input), whether this is built into the hardware architecture, or external hardware connected to the main system. The audio input requires some pre-processing (e.g. noise-filtering, de-reverberation, echo cancellation, etc.) before it can be intelligible and used in the game.

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 use, 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].

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 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 corresponding 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 prevents them from using traditional controllers or a keyboard and mouse. Very few popular games support voice recognition as a form of control, whether it's consoles, PC or mobile.

A simple mobile 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 with 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 phones to allow for hand-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.

2 Background

2.1 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.

2.1.1 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.

2.1.2 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 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 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.

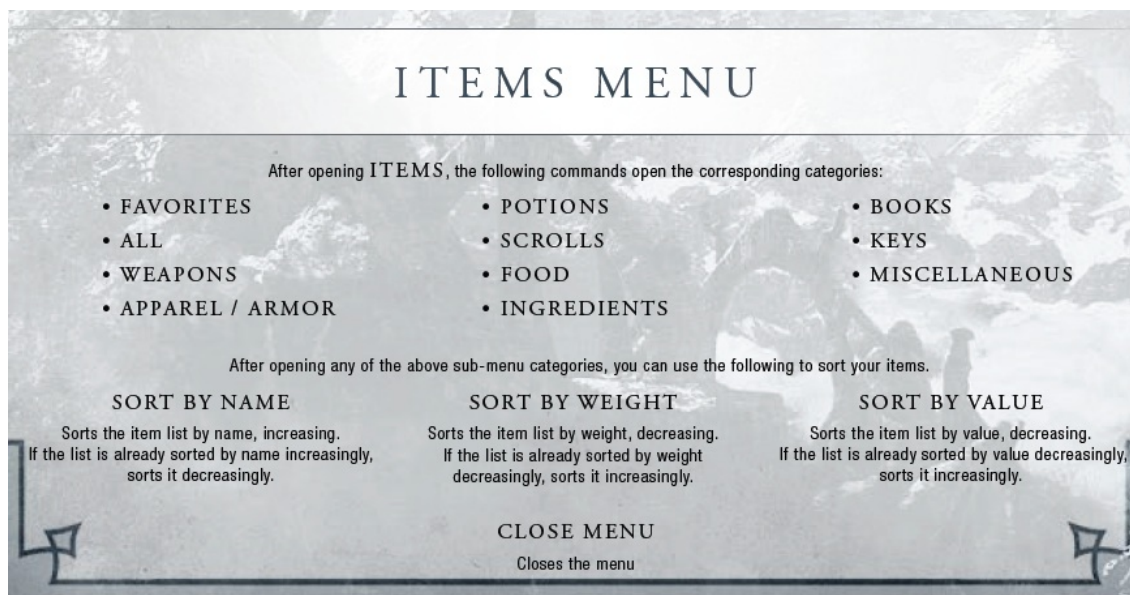


Figure 1: Voice commands that are available within the items menu in Skyrim.

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

2.1.3 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 [?]. 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 [?]. 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" [?]. 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; not only does it require a constant internet connection, but users are also charged for each API call that they make.

2.1.4 Classic Zork on Alexa

Zork is a trilogy of classic text-based role-playing games that supports 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") [?]. 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 [?]. 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 (described in the next section).

2.2 Limitations of Zork

The exploration mechanics of the proposed design of 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" [?].

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.

2.3 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:

2.3.1 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 [?]. 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.

2.3.2 Houndify

Houndify is a paid cloud service developed by SoundHound that allows developers to add voice recognition and control to any application they wish [?]. 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 [?].

Below is an example of a possible expression for attack a troll enemy:

"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")]

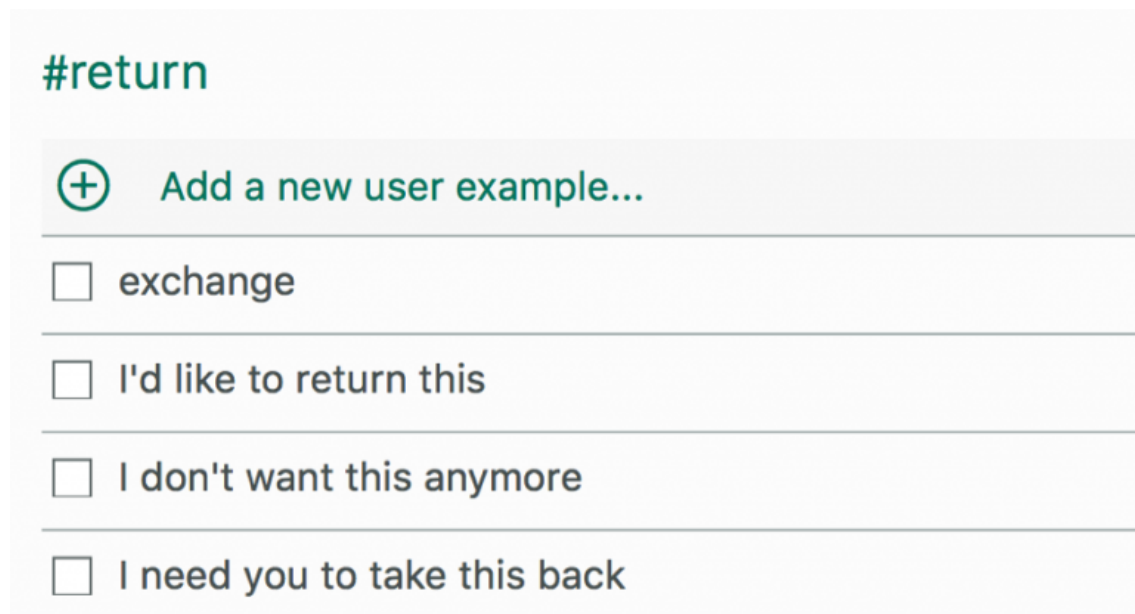
Even though this expression allows for more variation in what the user can say, it still doesn't cover all the possible variations of specifying an attack, and this long expression is only for one intent; many more expressions would have to be written for other intents, and this can become tedious for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

2.3.3 Watson Conversation Engine

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

'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 for the intent 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.



The screenshot shows a user interface for defining a conversation intent. At the top, the intent name is '#return' in a teal font. Below it is a button with a teal plus icon and the text 'Add a new user example...'. Underneath the button is a list of four example phrases, each preceded by an unchecked checkbox:

- ☐ exchange
- ☐ I'd like to return this
- ☐ I don't want this anymore
- ☐ I need you to take this back

Figure 2: An example of an intent to return something, along with examples of phrases for the intent [?].

'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.

@returnItems			
<div> <div>+</div> <div>Add a new value</div> </div>			
<input type="checkbox"/> book	text	tome	
<input type="checkbox"/> parrot	bird	macaw	Norwegian Blue
<input type="checkbox"/> video cassette	movie	tape	

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 [?].

Once the classifier is trained, it will be able to use the examples of intents provided to classify whether new examples 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 [?].

This API is very flexible and matches the requirements of this project; only a few examples 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 the cloud server) for 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.

2.4 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.

2.4.1 Stages of Natural Language Processing

NLP can be broken down into six stages [?]. 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 re-joined if needed) to uncover more information. These include words such as: clitic 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.) [?].

The third stage is syntactic analysis.

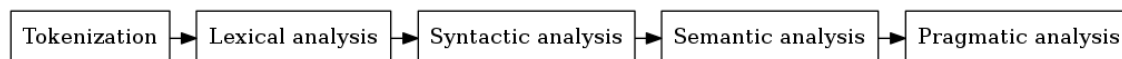


Figure 4: The general stages of processing a text string for its meaning using NLP.

2.4.2 Part-of-Speech Tagging

2.4.3 Synonyms and Hypernyms

2.4.4 Exploring Hypernym Trees

2.4.5 Semantic Similarity

2.4.6 RPG Sentence Structure

2.4.7 Object Properties and Relations

TODO: add brackets around title and author in bib file.

References

- [1] M. 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] I. Games, “In verbis virtus.” [Online]. Available: <http://www.indomitusgames.com/in-verbis-virtus>
- [3] B. S. 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/>