# Introduction

# Problem background

-Elderly more disposable income but less access to the internet due to problems with hands etc.

-With time, more and more old people will be computer literate and will thus need to be able to access computers or the internet in particular.

-Common uses of the internet for the elderly: Shopping, email, banking, Skype etc.

-Larger problem: Finding a set of techniques that can be used to improve the usability of computers for the elderly.

# Investigation purpose and scope

-Use the web browsing as a means of determining a set of methods that can be used to improve the usability of the internet for the elderly. Beyond this, such techniques can be used to improve the general usability of computers for the elderly.

Main means of improving computer usage is the use of speech recognition. Issue with regard to speech recognition is: What sort of visual annotations and voice commands are most suited to the elderly

The primary area of investigation was the issue of whether or not numerical referencing (assigning numbers to links on web pages) would perform better or be less confusing to elderly uses over spoken link names (referring to either their name or a key word within a link name).

-idea that has a simple set easy to pronounce words will perform better.

-idea that numbers are less conducing than having to read out entire link name or specific words

# Requirements

## Fundamental requirements

Iteration 1

....

Iteration 2

...

Iteration 3

...

-Determine whether or not spoken link names or spoken links perform better

-Determine which is preferred by the elderly users

-Determine whether visual feedback or voice feedback is more preferable for the elderly (iteration 1, minor)

## Secondary requirements

-Determine the feasibility of having voice processed on the cloud instead of on local machines

# Success criteria

-get an indication of which voice referencing technique performs better for the elderly

-Get results that give an indication of the preferred means of controlling the web pages by voice

-Does visual feedback give some sort of improved performance or improve user experience?

# Constraints

-Cost: Constrained to use freeware tools.

-Time: Limited time to conduct tests and develop software. Lots of time required to train the elderly...

-Test subjects: The elderly are not always keen to currently conduct experiments on computers

-Health: Finding test subjects who have sufficient hearing, eyesight and intellectual consistency to conduct the test.

-Skill: Most elderly people are not computer literate

# Assumptions

-Assume that all test subjects are not computer literate. This meant that earlier tests had to be very sequential in nature.

-More complicated tests, complete guidance

# Investigation procedure

-construct basic test to derive basic preference

-conduct more specific tests to confirm previous results and elicit more specific user preference

-Conduct a test on a facsimile website which incorporates multiple features from previous tests in an attempt to gain a more qualitative assessment of user experience on this more complicated site.

# Sensitivity to test subjects

-Repetition limited to three times per mistake

-If individuals discomforted at any stage of the test, they were permitted to stop (some had heart problems)

# Ethics

# Tools

Free tools used.

-Initially, Eclipse was used to write the JavaScript and run the client code locally.

-Eclipse also ran very slowly on our machines

-unpredictably and problems that arose meant that an alternative had to be sought out. The loosely typed language of JavaScript meant the usual advantages of IDE’s are not really experienced...

-Chose to use Note++ to do syntax checking

-ran IIS server

-Version control: Used Git and Github to perform merges and backups

-Toggl used to keep track of the time spent on the project

# Group work and division

-Git

-from when Git was used (initial merge)

-Division

# Problems encountered

-Unpredictable behaviour with IIS

-Going through proxy server, vocabulary populated with unknown words

-Internet connectivity and the problem with testing. One of the tests had to be aborted

-Finding test subjects who were willing

# Time breakdown

-History with Toggl

-Due dates for different iterations

-Meeting deadlines?

-Meeting with supervisor?

# Results

## First iteration

## Second iteration

## Third iteration

## Interpreting the results

# Implementation

## Implementation overview

The developed code essentially evolved through the three iterations of the project. For this reason, the implementation will be discussed in light of the most complex iteration, Iteration 3. Most of the functionality remaining in Iteration 3 could be used in earlier iterations as well, provided that the appropriate parameters are set. However, since no more changes were required in earlier iterations of the project as iterations progressed, the *JS* code used for each iteration was separated for each iteration.

An overview of the flow of the web application is given in Figure XXX in Appendix XX. Each web-page includes various segments of javascript. This code serves to provide an interface to the flash component used to stream audio to and from the server and to process the results from the server. The overview of the processes involved will now be discussed in brief.

Once the javascript has been loaded, the necessary flash component is set up (see section XXX). The page is then processed according to the mode set (see section XXX) and the vocabulary required for a certain page is determined. This vocabulary then gets initialised onto the server so that the server knows what words to expect. From here, the program waits for the voice recognition to be activated through the flash component. When this occurs, that is when the *Ctrl* button is depressed, the user’s voice input from the microphone is then streamed to the server through the flash component until a stop recognition command is given (until *Ctrl* button is released). The program will then wait for the server to return a result (either a known word or the words, “recognition error”). The result is then processed and classified according to the type of command being issued.

If the result could potentially lead to navigation, a confirmation of whether the command was recognised correctly is required (see section XXX) before navigation is carried out. Confirmation, in the programs context, means that the user must either say, “Yes” or “No”. If confirmation mode (*confrimationMode* global) is set to true and the user confirms positively, a link is followed or the command is performed, otherwise, the code will refresh the pages.

If the result pertains to link navigation, the link is highlighted and the confirmation mode is changed, otherwise, if the command is not associated with any link in particular, the user has to rely purely on the verbal feedback of the command for indication of result interpretation. Once confirmation mode has been set, the vocabulary on the server is set to only, “Yes” and “No”. The user will then need to depress and speak in to the microphone to confirm or decline the speech recognition’s interpretation of what is said. If what is said this time is an affirmation of the command (“yes”), then the highlighted link is followed or specific command is carried out. Otherwise, if no, the highlighting of the link is undone, and the vocabulary is set up again for the page.

On the other hand, if the result is of a type that does not require confirmation (namely, “up” to scroll up or “down” to scroll down) then the command is simply executed.

## Using the speechAPI

All voice recognition is processed using an online API from SpeechAPI.com.

SpeechAPI [ref] provides a flash object, *SpeechAPI.swf*, which can be embedded onto an HTML page using a *SWFObject.* This flash object is then used to stream audio between the client and server. When the server sends a callback back to the client, the *SPeechAPI.js* code interfaces with the flash object to allow processing to be conducted on the result call-back.

Once the flash has been successfully loaded, the *onLoaded (see Figure XXX)* method is called. Importantly, when this method is called the server needs to receive the vocabulary it needs to enable recognition for a particular web page. So depending on what the flow of the program has been set for (numerical voice referencing or spoken link names), the vocabulary is either set to include certain numbers (as required by a particular page) or names (as indicated by various link ids).

When the flash component receives the command to start recognition it starts streaming audio to the server (see section xxxx). The server will try and match in given input to one of the specified vocabularies. A string result is then returned indicating a closest match result or the string, “recognition error” if no result was found.

The API also lets you set it up so that it recognises alternative words for the same thing (so one could potentially say “go up” or “scroll up” etc.). This would give the application a more natural interface. Beyond this, the API also lets you construct grammar segments in such a way that parts of a sentence are not necessarily necessary. If thus functionality was used, links could be followed in this way instead using highlighting. However, the scope was limited to highlighting at this stage so that a basic comparison between numerical referencing and spoken link name referencing could be conducted.

## Flow Control

### Controlling flow and notifications

The main processing of results from the server (including setting up connections) is conducted in *speechprocessor.js*. Since two primary streams of tests were undertaken, two main courses of the program are determined by the state of the Boolean variable, *numericalReferenced*. If this variable is true, the program will divert to a stream which performs the required functions for numerical functions. Otherwise, the code diverts on a course that is implicitly, one of spoken name referencing.

Another Boolean variable which helps determine the course of the code is the variable, *confrimationMode*. This variable is set to true every time a result is returned from the server indicating that a link should be followed or the result is interpreted as a certain command that requires confirmation. The process introduced with the confirmation mode can be seen in section XXX.

### Confirmation mode

Confirmation was a feature only introduced at Iteration 3 of the program. It was introduced to compensate for misinterpreted user commands. From the results of the earlier iterations it was clearly evident that results would often be returned as something they are not. If a result is interpreted by the speech engine as being significant in nature, that is it will result in some form of navigation, a confirmation is required of the user (so for commands “Home”, “Backwards”,””Forwards” and all link selection).Although users preferred not to have to perform a confirmation step (See Section XXX), it is at this stage necessary.

## Implementation details

### Initialising and loading

The script will first call the *initialise* function which will set up the flash, establish a connection to the server and embed the flash. From here the program will wait until the *onLoaded* call-back is called by the provided SpeechAPI.js code. The processes involved in this call-back are illustrated in Figure XXX. In here, the vocabulary is calculated using *determineTotalVocabulary* (Figure XXX) and then *determineSpecificVocab* (Figure XXX). *DetermineSpecificVocab* then calls specific functions : d*etermineSpecificConfirmationString*, determineSpecificLinkNamesString and determineSpecificNumericalString depending on whether or not confirmation mode is enabled (through the confirmation mode variable, see section XXX) or if numerical referencing is being used (See section XXX). These functions will separate a string which contains all the commands (“home” etc.) strings and all the link referencing vocabularies (“one”, “two” etc. or “dog”, “cat” etc.) from a single string which was constructed to contain both the commands and link vocabularies (Please note: the vocabulary process for all types of referencing revolved around these sort of concacentated strings because this system was used initially on numerical referencing. Having all the vacab placed in one string meant that If you wanted to add additional commands or numbers you could all do that through a single string. The other streams (namely spoken link names and confimaion mode) also revolved around this sort of concatenated string so that all the processing for all modes was similar in logic. This makes it easier to follow each flow of vocabulary processing because all undergo the same sort of processing.)

The flash component then has it’s designated vocabulary loaded onto it (which in turn loads on the server).

### Assign referable names to links

For numerical referencing, the neededNumbers array (see section XXX) stores a list of all the needed numbers. These numbers can the be aligned to the link URLs as mentioned in section XXX. To extract the actual words for spoken link names, the function *assignLinkWords* is used to extract all the link id’s (which have previously been assigned the appropriate names) which are then assigned to the array. This array’s (*neededNamesArray)* information is later used in the same way as the *neededNumbersArray*  to search for the correct index of a returned result and consequently navigate to a link (see section XXX).

### Processing the result

When the *onResult* call-back is called, the *processResult* function is called (See Figure XXX). This function performs a series of checks that determine how to treat the result. The function hieratically performs the indicated checks. The functions *tryProcessConfirmatio*n and *tryProcessResyltAsCommand* simply conduct tests to check what type of confirmation or commands have been issued and perform the appropriate actions if a command is found. For *tryProcessResultAsLinkName* and *tryProcessResultAsNumberedLink* arrays containing all the words derived from the ids and all the numbers (*neededNumbersArray*  and *neededNameArray)* are traversed to try and find the result. If a result is found, the array index is recorded. Every link URL is then assigned to an array so that each one aligns with its corresponding link name or number in the *neededNumersArray* or *neededNameArray*. The index from the traversal is then used to access and extract the appropriate URL from the links array. Once the right URL is had, the link is followed.

### Different settings for the script

# Variation from plans

# Critical analysis

## Tradeoffs

Problems

## Recommendations

-finding the key words within link names dynamically. Can this be done to an extent that will provide adequate clarity sufficient distance between words?

-What sort of words is commonly misinterpreted?

-trained speech engine to deal with elderly voice requirements

-Hand held activation?

-Future iterations:

-Automatic button pressing

-Numbering starts again for different sections (confirm whether people like it or not?)

-Colour calibration for links colour and link highlighting, setting confirmation settings

-Do all of this on another website? (more complicated?)

-Possibly extend from mock up to real website to test feasibility of methods in real applications (real websites)

-parts of sentences looking up alternative words for things

# Conclusion

# Bibliography

Appendix:

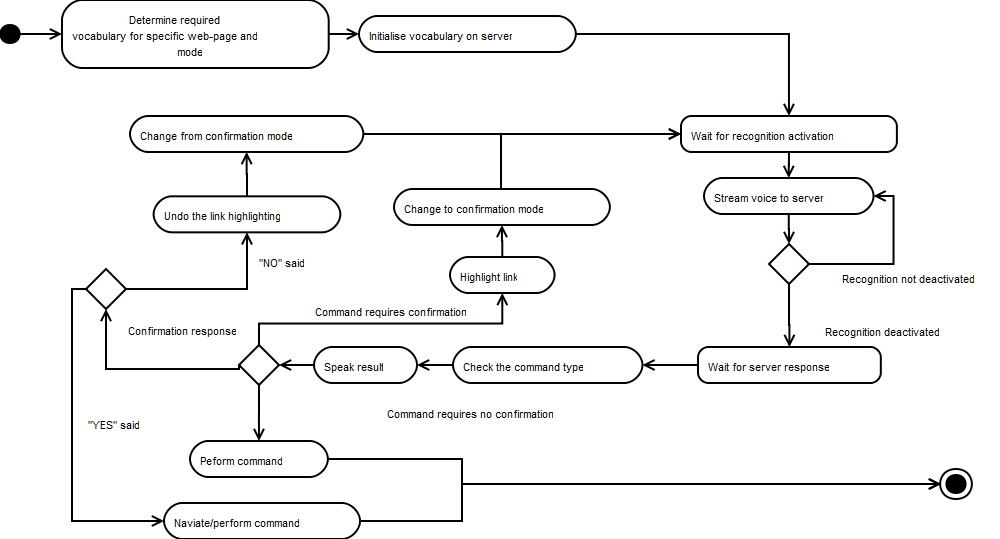


Figure A: Total overview of voice navigation

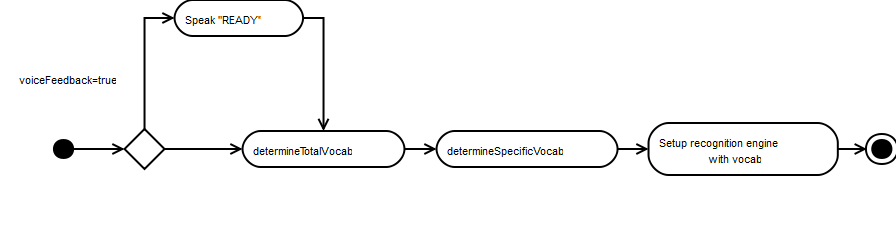


Figure : OnLoaded Overview

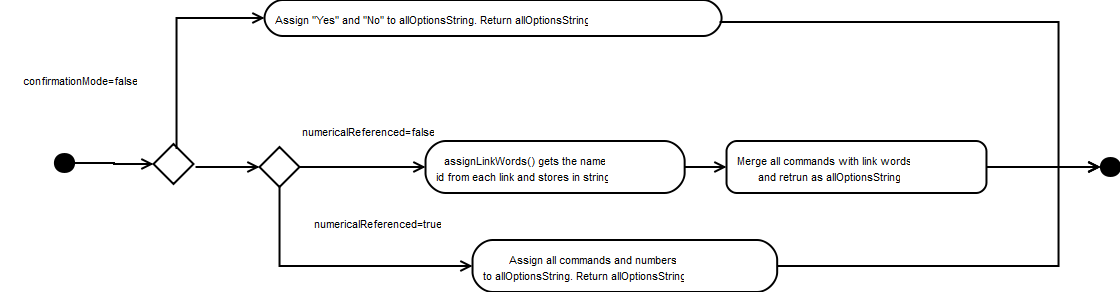


Figure : Determine total vocab overview

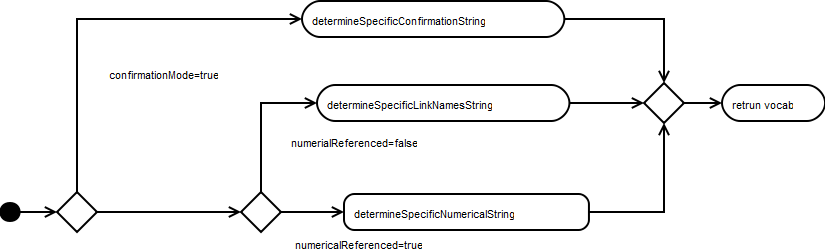


Figure :Determine specific vocab

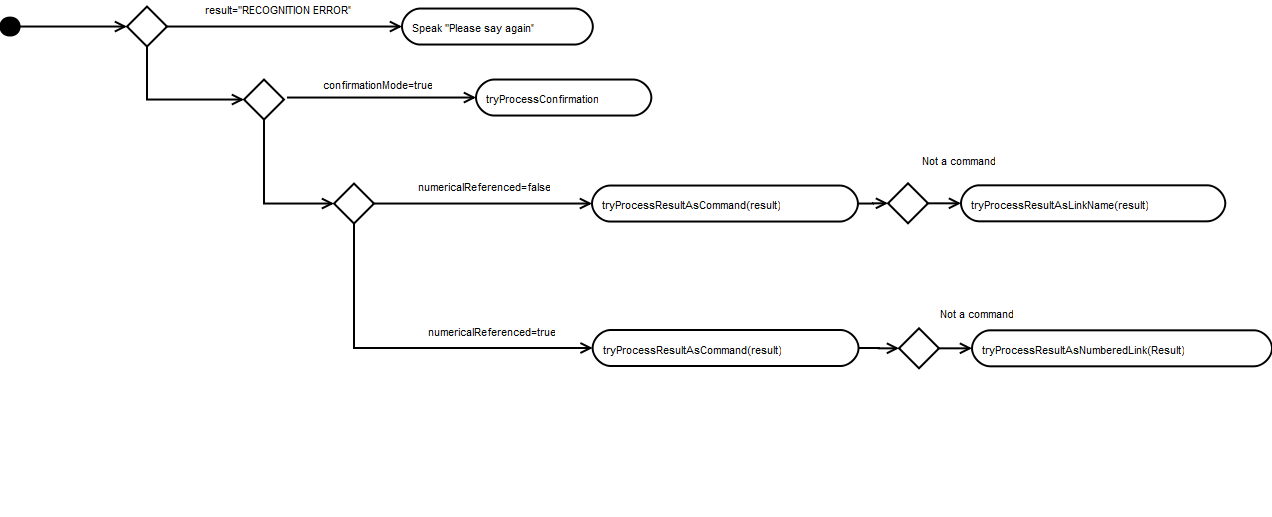


Figure : Result processing