# CS494 Natural User Interaction

***Group Members:***

Tosan Egbesemirone

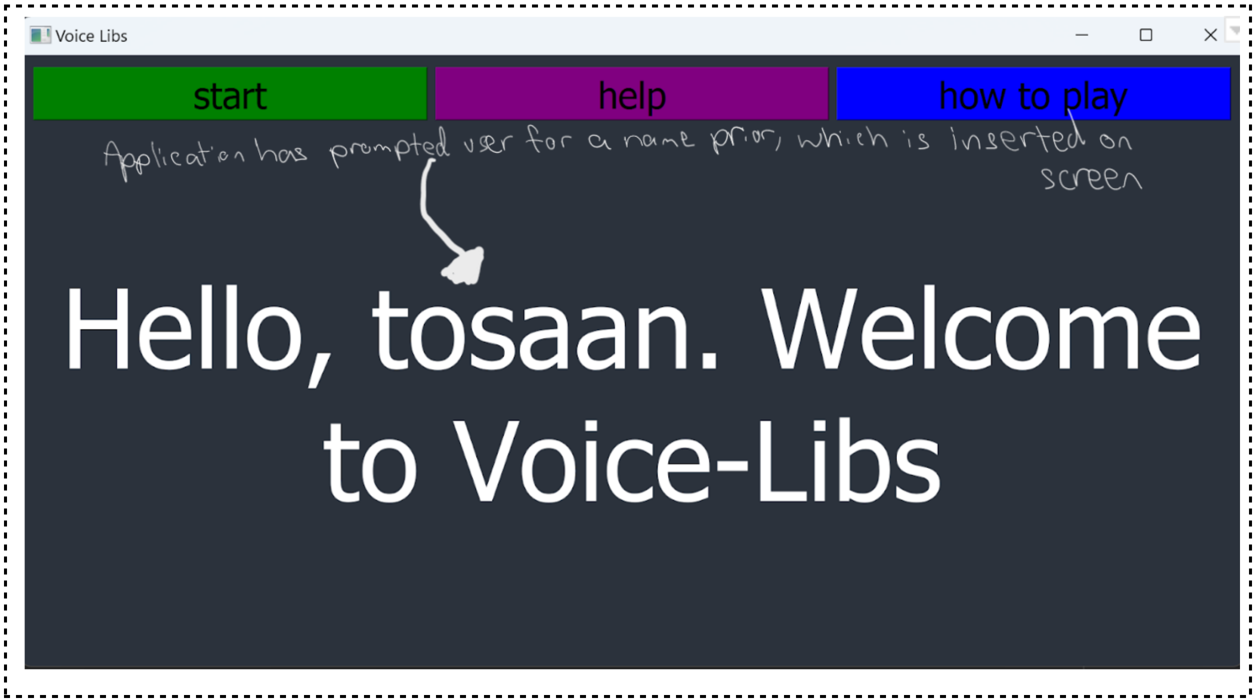
Haresh Jhaveri

Megan Mehta

Rag Sharma

***Section:*** Undergraduate/Graduate

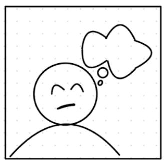
## VoiceLibs


Voice Libs
start
help
how to play
Application has prompted user for a name prior, which is inserted on screen
Hello, tosaan. Welcome to Voice-Libs

# 1 Iterative Design and Development Process

## 1.1 Initial Design

The original project idea was to design a choose-your-own-adventure audio game that would be operated and executed through phrasal commands. We thought using a speech NUI would work well since we wanted to give users audio feedback. The speech commands we included in the first prototype were Finish Story, Help, and Read Story. We chose these commands since the primary goal of the audio game is to create a story - Finish Story saves the final story creation offline. Help was chosen as a way for users to understand more about the game and to also know the possible command options. Read Story was chosen since in the process of going through the game, users are creating each line of their story one at a time. Read Story can give users a preview of what they have created so far without ending the game. We used a persona to model what our intended user journey would be like. Our vision for the game is to give users the most creative freedom without having to use anything other than their voice. Because Madlibs traditionally required you to write on paper to play the game, the utility of speech in the user experience creates a more natural flow of the game. Speech navigation also creates accessibility for users that lack the hand dexterity necessary to write with a pen or pencil. The culmination of these inclusions aims to create an application that can be navigated completely using audible words and phrases if the user decides, with the option of buttons to start, ask for help, and exit the application. Here is a storyboard for the flow state of the game:



Bill has a lot of story ideas in his head, but sometimes he needs help just putting it all together.

Bill discovers VoiceLibs - an interactive audio game that allows users to create their own stories.

please give a noun
once upon a time. a blank lived

With the help of VoiceLibs, Bill can inject different words (like nouns and adjectives) into a template and make it his own story.

please give a silly word
in the land of blank

After going through a few iterations...

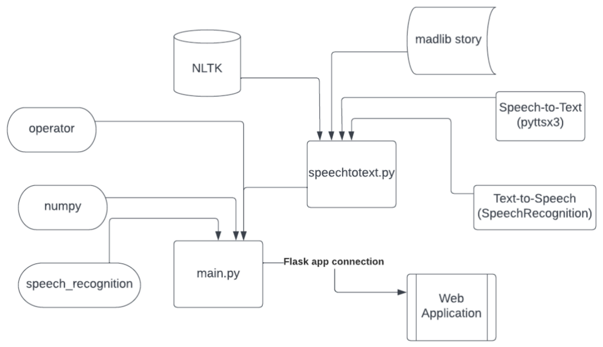
Once upon a time,
a prince lived in
the land of tomato's
finish story

Bill decides to save his final story!

Now Bill is happy that he was finally able to create his own short story - with the help of VoiceLibs!

## 1.2 Developing the First Prototype

When developing the first prototype, we kept track of potential features and what group members were working on in a Google Doc and would update each other via GroupMe if we had any successes/failures in building. We used GitHub for version control. Our initial prototype design process intended on utilizing Java and Maven as the language to support application development. We believed we needed an OOP based language to execute this project, which is why we opted for Java. We also realized that since we planned on building an application that primarily utilized speech and audio to traverse the play state, IBM Watson would be a good starting API to integrate for text-to-speech functionality. Our team immediately faced issues with integrating java into our devices, as half of us developed on MacOS, and the other half on WindowsOS. This led us to changing languages to python, and utilizing Pyttsx3, a python text-to-speech API, as well as google-speech to transcribe audio. Since we developed our first prototype without user feedback, we assumed that static commands like "Help", "Read Story", and "Finish Story" would be suitable for navigating this first version of Voicelibs. For processing if a word was an acceptable answer for certain sections of the story, we used wordnet through NLTK API to pull from an online dictionary. Although this is primarily an application that uses audio, we needed some visual aid to accompany the play state. We initially planned on using python Flask to accomplish this. See below for the first version of our system architecture:



**Member Contributions:**

Tosan: Core application development, Game state algorithm design, Github Version Control (Converting git repo from Java to Python), Bug tracking and error mitigation.

Rag: Api Integration, Word Analyzer algorithm development, Speech Ambient Noise integration.

Megan: Figma prototyping, Storyboard, User Testing, GUI development (HTML/CSS/JS and PyQt5)

Haresh: Error checking and Help Page development, research.

## 1.3 Developing the Second Prototype

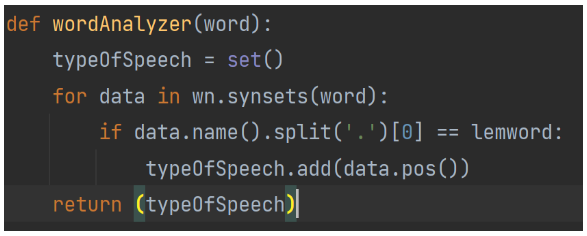
We built the second prototype with the context of having user feedback from our prior user studies. Users were mainly college students, a mix of CS and non-CS majors. Majority of our users were males, ages ranging from 22-26 years old. They were also provided with an introduction to Voicelibs, and were asked how familiarity with voice applications prior to their user test. The user tasks were: make your own story, save your story offline, learn more about what to do, and listen to your story. The procedure for our user testing was to have the users start with the application, then our team would record the session (either by audio recording the session or taking notes) and observe how users interacted with the application. We designed the study so that only one developer needed to be present at the session. Immediately, we noted a recurring critical incident we named “Infinite Listen Loop,” where the application’s inability to record a user’s answers in noisy environments frustrated some participants, and froze the playstate. Another important critical incident was the initial name prompt accepting longer answers. The application would ask, “Hi, what is your name?” If the user responds, “Hello my name is Adam”, then the application’s response would be “Hello Hello my name is Adam, welcome to Voice Libs…” Throughout the playstate, a critical error would reveal itself when a user submitted a word in the past or future tense, or as the plural version of the word. We addressed these major critical errors by integrating two APIs, Google Dialogflow and NLTK Lemmetizer, and adding dynamic ambient noise adjuster to the Pyttsx3 function we developed.

### 1.3.1 Change 1

The first major change was the integration of Google Dialogflow. Although our initial design made inclusion for Google Dialogflow, our first prototype did not have any of the API’s functionality. Google Dialogflow is able to parse and interpret different sentences to assign parameters to key words recognized in a user's response. This is important because it solves the critical error where a user’s full response to the name prompt would be imported into the application. DialogFlow allowed us to extract the user’s name for the gameplay. We were also able to alter our static commands like “Help”, “Read Story,” and “Finish Story” and make them dynamic. This means that users can more naturally communicate with the application to navigate and traverse the play state. Instead of saying “Finish Story,” a user can say “Can you finish my story for me?” or “Complete my current game” or “I would like to finish my story now.” Dialogflow would recognize these variates, and return the correct command to the application to prompt actions in the application. These design concepts were implemented while enhancing the “Read Story” and “Help” commands, as well as while integrating the “Repeat” and “Quit” functionality. DialogFlow also allows us to account for when a user doesn’t respond with a one word response while entering words to fill the story. We can now accurately recognize answers like “hmm, add time” or “can i say pretty?” by identifying the keyword in that response, and returning the keyword to the application. This change had the third priority on our list of changes.

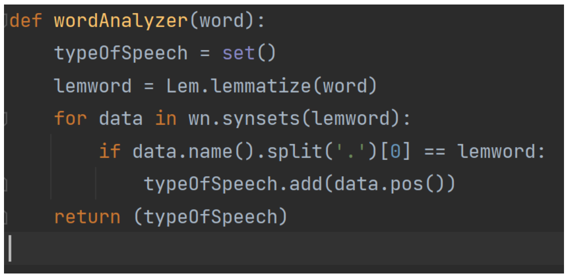
### 1.3.2 Change 2

The user studies revealed that users faced issues when saying past tense words or plural words. Below is a screenshot of our initial word analyzer, which returns what type of word a user’s response would be in the english lexicon:



```
def wordAnalyzer(word):
    typeOfSpeech = set()
    for data in wn.synsets(word):
        if data.name().split('.')[0] == lemword:
            typeOfSpeech.add(data.pos())
    return (typeOfSpeech)
```

The issue with our original design was that our word analyzer was unable to evaluate past tense or plural words. The application would not recognize “apples” or “ran”, which led to the user's answers being rejected. See below for the new version of word analyzer that was developed for the second prototype:

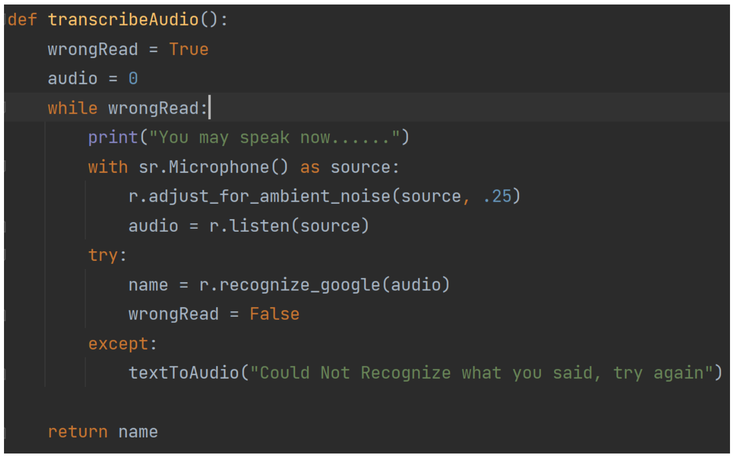


```
def wordAnalyzer(word):
    typeOfSpeech = set()
    lemword = Lem.lemmatize(word)
    for data in wn.synsets(lemword):
        if data.name().split('.')[0] == lemword:
            typeOfSpeech.add(data.pos())
    return (typeOfSpeech)
```

By integrating one line, we were able to now analyze these plural, past tense, and future tense words. Lemmetizer would derive a user’s response into its base root. For example, if a user said "apples" the Lemmetizer would derive it to "apple," then analyze if that word type is acceptable. If "apple" showed up in the wordnet dictionary, "apples" would be returned to the application, to replace the appropriate blank. This creates a more natural user experience, as it increases the amount of acceptable words a user can respond with while playing the game. This change had third priority in terms of importance to application, as the game still ran with less functionality.

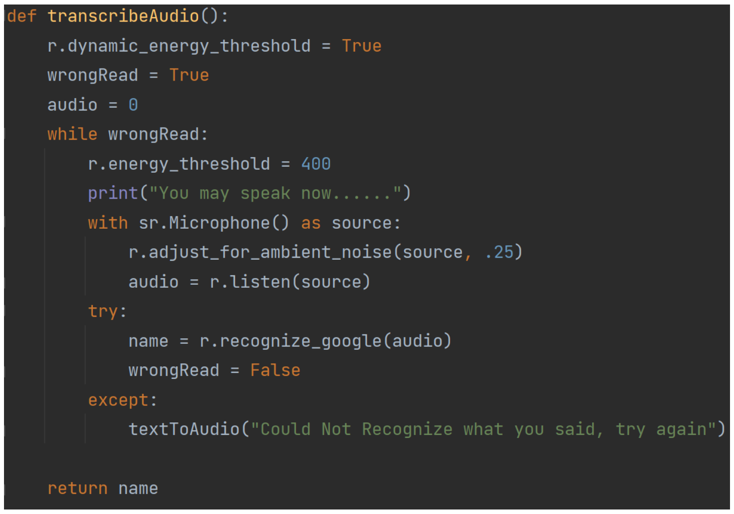
### 1.3.3 Change 3

One of the most consequential critical incidents was the "Infinite Listen Loop" which was capable of freezing the game state for the user, resulting in an application failure. Please look at the diagram for our function transcribeAudio():



```python
def transcribeAudio():
    wrongRead = True
    audio = 0
    while wrongRead:
        print("You may speak now......")
        with sr.Microphone() as source:
            r.adjust_for_ambient_noise(source, .25)
            audio = r.listen(source)
        try:
            name = r.recognize_google(audio)
            wrongRead = False
        except:
            textToAudio("Could Not Recognize what you said, try again")

    return name
```

The issue was the r.adjust_for_ambient_noise(source, .25) would adjust the energy threshold, which is how loud a user has to speak to be heard by the application, and would then make the threshold a static value, meaning it wouldn't readjust for the next time the application attempted to listen to the user. See the screenshot below:



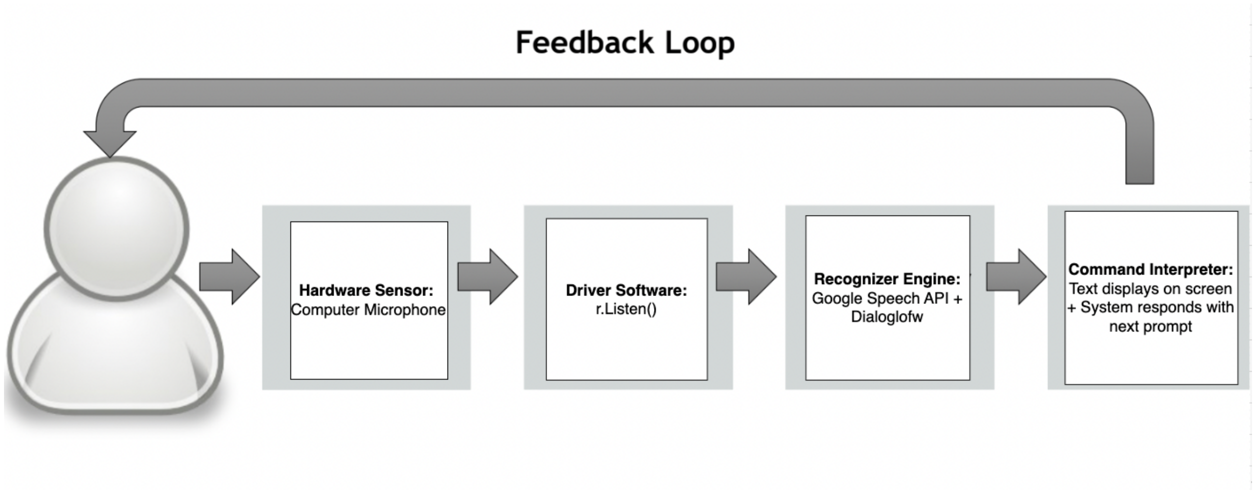
```
def transcribeAudio():
    r.dynamic_energy_threshold = True
    wrongRead = True
    audio = 0
    while wrongRead:
        r.energy_threshold = 400
        print("You may speak now......")
        with sr.Microphone() as source:
            r.adjust_for_ambient_noise(source, .25)
            audio = r.listen(source)
        try:
            name = r.recognize_google(audio)
            wrongRead = False
        except:
            textToAudio("Could Not Recognize what you said, try again")

    return name
```

We added a few new lines of code. For one, we set r.dynamic_energy_threshold to True which would make the function transcribeAudio(): adjust the energy threshold each time the application needed to transcribe the user's response. We also added a line to reset the energy threshold to 400 each time the application can't recognize the user's response. This creates a system that can analyze a user's answers more effectively in multiple environments. This was the highest priority change, as this critical incident was detrimental to the application and user interaction.

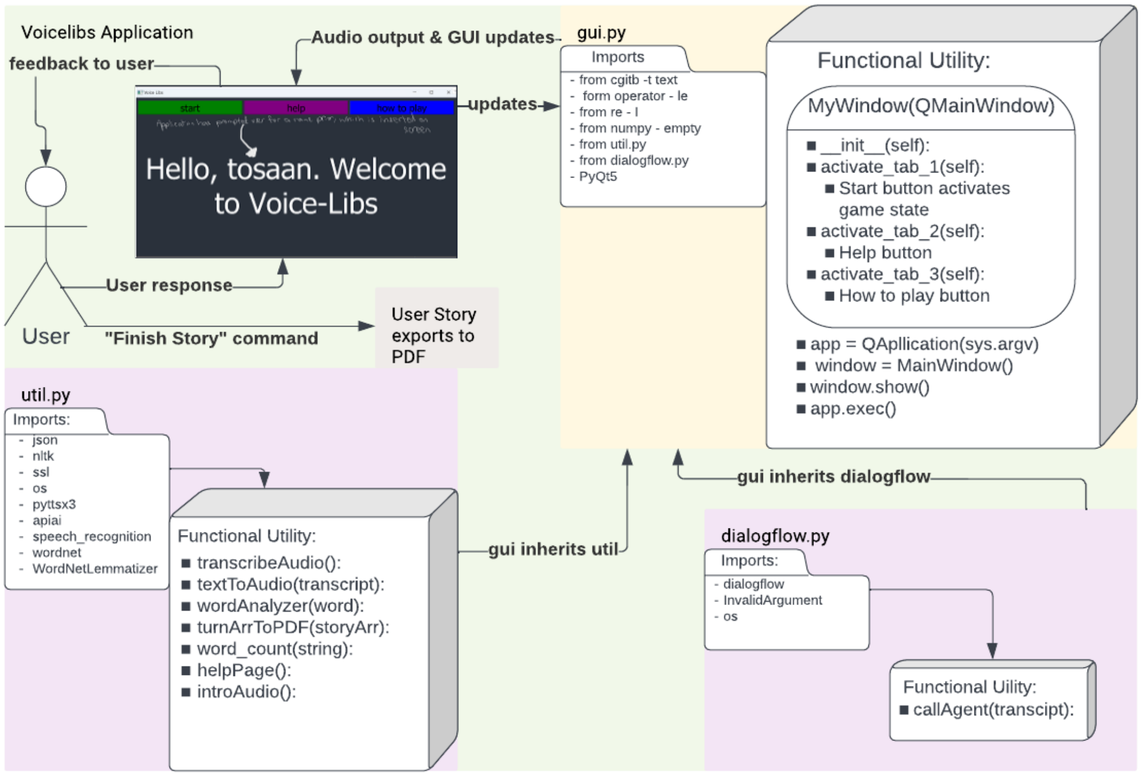
## 2 Final Architecture

### 2.1 System Architecture

Below is our feedback loop depicting the flow of the game play and audio recording/delivery:


Feedback Loop
Hardware Sensor: Computer Microphone
Driver Software: r.Listen()
Recognizer Engine: Google Speech API + Dialoglofw
Command Interpreter: Text displays on screen + System responds with next prompt

## 2.2 Code Modules:



Above is the code module to represent python file interactions, and GUI to user interaction.

## 2.3 Third-Party Tools

- PyQt5: enables Python to be used as an alternative development language which is used to produce the GUI of the program. https://pypi.org/project/PyQt5/
- Google Cloud Dialogflow: Api used to parse through the user's speech. This took the longest to integrate and implement, but the quickest time to develop. https://cloud.google.com/dialogflow/docs
- SpeechRecognition 3.8.1: speech_recognition is derived from this API and allows us to record audio and translate it into text with a builtin Google's speech to text function. It helped us resolve issues of detecting user voice in different environments by dynamically adjusting the ambient noise level. https://pypi.org/project/SpeechRecognition/
- Google ApiCore Exceptions: Used to catch an InvalidArgument exception. https://googleapis.dev/python/google-api-core/latest/exceptions.html

- Nltk: Leading platform to build python programs to work with human language data that provides various easy to use interfaces. Used to get the Wordnet and Lemmetizer interfaces. https://www.nltk.org/
- Wordnet: Utilized as the dictionary for the words to get word type and lemmatizer https://www.nltk.org/howto/wordnet.html
- Lemmatizer: Used to get the word's lemma/dictionary form for word analyzer to understand its word type. https://www.nltk.org/_modules/nltk/stem/wordnet.html
- Pyttsx3: Converts text to speech which is used to speak to the user. https://pyttsx3.readthedocs.io/en/latest/
- fpdf: Utilized to convert the story into a pdf file. http://www.fpdf.org/en/doc/index.php
- json: Used for work with json files. https://docs.python.org/3/library/json.html
- apiai: Similar to speech_recognition, takes the user's speech and converts it to text. https://apiai.readthedocs.io/en/latest/
- cgitb text: Utilized in exception handling. https://docs.python.org/3/library/cgitb.html
- os: Utilized for operating system dependent functionality like opening and writing into files or reading from files. https://docs.python.org/3/library/os.html
- re I: Utilized for special python operations that specify the rules of the set of possible strings that you want to match. https://docs.python.org/3/library/re.html
- dialogflow: This imports the functions from dialogflow.py into the GUI application.
- util: Imports the functions and imports from util.py into the GUI application.
- numpy empty: Returns a new array of given shape and type with uninitialized values. https://numpy.org/doc/stable/reference/generated/numpy.empty.html
- ssl: Wrapper for socket objects. https://docs.python.org/3/library/ssl.html
- asyncio windows events: Used for NULL. https://github.com/python/cpython/blob/main/Lib/asyncio/windows_events.py
- operator le: Standard operators as functions. https://docs.python.org/3/library/operator.html

### 2.4 Source Code

Github Link (We have sent an invite to the repo to your UIC email. If you can't join, we also have the zip files here: Google Drive Link)

## 3 Future Work

Design:

For a future application we would like to incorporate a typewriter effect in the GUI. As the application speaks, text should appear as if it was being typed on screen at the same time as each word is being spoken. We also want to incorporate a smarter word analyzer to help users gain access to a larger vocabulary of words to use. We would also like to modify the Help and How To pages to be more robust and user friendly.

Development:

To make the word analyzer smarter, our plan is to create an algorithm to determine a word's synonyms. Sometimes a word such as “slim”, while recognized, does not have any word type according to the NLTK dictionary. However, we can take a set of synonyms and run a word analyzer helper to determine if any of the synonyms match our required word type. If found we can then pass the original word as a valid word to use.

To implement the typewriter we need to change the way text is currently being displayed. Currently, it only shows what the sentence the speaker is saying. To add the typing effect we will have to send each character individually to the GUI with some delay in between each. The delay should be very short and match the speed of the story teller.

Our user interface would be cleaned up with colors to match a theme. This would help create a complete look as well as an updated font and button styles to make it easier for the user to navigate and look at.

Evaluation:

After developing the application with these new features we have to test run our application again. We would cross-check to see if words considered no word types, like “slim”, can be recognized now and test to see if our typewriter is matched to the speed of the speaker. We then would do a similar user study to get a better understanding of what others think of the application. The study would keep the tasks similar except have the user maneuver and use the GUI. The review questions would still include their opinions on the application and we will be able to go through more critical incidents to improve our program for an official release.

## 4 Video

https://youtu.be/uSoYsvM18BU