4-11-2023

Computer Based Organization for the UAH Moonbuggy Workspace

Riley William Cavitt

Computer Based Organization for the UAH Moonbuggy Workspace

by

Riley William Cavitt

An Honors Capstone

submitted in partial fulfillment of the requirements

for the Honors Diploma

to

The Honors College

of

The University of Alabama in Huntsville

4/23/2023

Honors Capstone Director: Mr.David Fikes

Lecturer of Mechanical and Aerospace Engineering

Riley W Cavitt	4/11/2023
Student	Date
	4/13/2023
Director	Date
	4/17/2023
Department Chair	Date
William Wilkerson Digitally signed by William Wilkerson Date: 2023.04.23 14:46:58 -05'00'	
Honors College Dean	Date

HONORS COLLEGE
THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Honors College
Frank Franz Hall
+1 (256) 824-6450 (voice)
+1 (256) 824-7339 (fax)
honors@uah.edu

Honors Thesis Copyright Permission

This form must be signed by the student and submitted as a bound part of the thesis.

In presenting this thesis in partial fulfillment of the requirements for Honors Diploma or Certificate from The University of Alabama in Huntsville, I agree that the Library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by my advisor or, in his/her absence, by the Chair of the Department, Director of the Program, or the Dean of the Honors College. It is also understood that due recognition shall be given to me and to The University of Alabama in Huntsville in any scholarly use which may be made of any material in this thesis.

Riley W Cavitt

Student Name (printed)

Riley W Cavitt

Student Signature

4/11/2023

Date

Table of Contents

Abstract

For the NASA Human Exploration Rover Challenge (HERC), The UAH Moonbuggy senior design course uses a small room to work on and store the various tools and parts used with the Moonbuggy. This results in a significant amount of clutter and time spent looking for parts and tools. In an attempt to reduce this issue, an easy-to-use program will be developed to keep track of a data file including the names and locations of various objects stored in the room. To build the device, a Raspberry pi will be used with a speaker and microphone to receive vocal commands and return an audible list. Python will be used to create a program that manipulates a log of where things are and informs the user of their whereabouts.

Introduction

Current Problem

The UAH Moonbuggy room is a medium sized room filled with parts, tools, and many different projects. For a student first seeing the room, it can be overwhelming just finding the simplest thing. Previous attempts to clean the room have yielded temporary results as boxes are frequently being left in the room for use in other projects. This creates a unsatisfactory work space where it is challenging to find things without first spending hours getting familiar with the room. It also encourages a practice of leaving projects and tools in locations other than their desired resting place. This slows down work and leads to an ever-growing mess of disorganization.

Presenting a Solution

The solution that this paper discusses is to use a Raspberry Pi to keep track of a database of information regarding the location of items and boxes around the room. This can then be used to update and change that list and, more importantly, tell the user exactly where the item is in the room along with the quantity of how many items there are and when it was catalogued into the database. This can be done with text commands, but to better help future generations of Moonbuggy students it has been designed with voice commands as an alternative input with text-to-speech audio as an output. Ideally, this would allow users to use the program hands-free and from across the room.

Previous Experience

Previous projects to use voice commands on the Raspberry Pi have been met with limited success due to recent events limiting the use of Google and Amazon online voice recognition and

the lack of updated information as to how to proceed. However, this paper outlines the steps necessary to putting and using voice input on a Raspberry Pi and how that can be used to aid in an organizational tool for clustered workspaces.

Main Body

Current Events

The biggest obstacle for this project was also its inspiration. In years prior, Google and Amazon gave free access to their online voice recognition software which inspired the idea of a Raspberry Pi based voice assistant. Unfortunately, a few months later, Google and Amazon rescinded the ability to connect to their online voice to speech translation and the ability to use speech recognition on Raspberry Pi was lost.

This led to a series of trials and errors to get Mozilla's DeepSpeech speech recognition software working on a Raspberry Pi. It is easy enough to put it on a traditional window computer using the python expansion Pip, but putting it on a Raspberry Pi is considerably harder because the current operating system is not listed as a valid operating system. This prevents pip from downloading DeepSpeech on up-to-date operating systems. These ineligible systems include the newest version of the Raspberry Pi OS and Windows 11. There is a work-around to this problem that does allow DeepSpeech and all the necessary modules to be installed on a Raspberry Pi. Which will be described in detail later in the paper.

Description of the device

The final product of the project is a Raspberry Pi 4, with a microphone, speaker, keyboard, mouse, and monitor display. When the main program is run it asks for an input for either text commands or voice commands. This will then either run the text-based program or the

speech-based version. They both function very similarly by asking for an input (0-9) which executes a function to alter a .CSV (Comma Separated Value) file consisting of the contents of the Moonbuggy room. Using command, 'END' or 'end' will cause the program to create a backup .CSV with the current date in the filename.

Physical attributes

The Raspberry Pi used is a Raspberry Pi 4 model B with 8Gb of ram. It is connected to a generic keyboard and mouse and uses a USB microphone and a USB speaker for the audio input and output. Any microphone or speaker should work. The operating system is loaded onto a 32Gb micro-SD card, however an 8Gb micro-SD card would suffice. Do note that this model of the Raspberry Pi uses a micro-HDMI port which can be converted to traditional HDMI. A picture of the device is shown below.

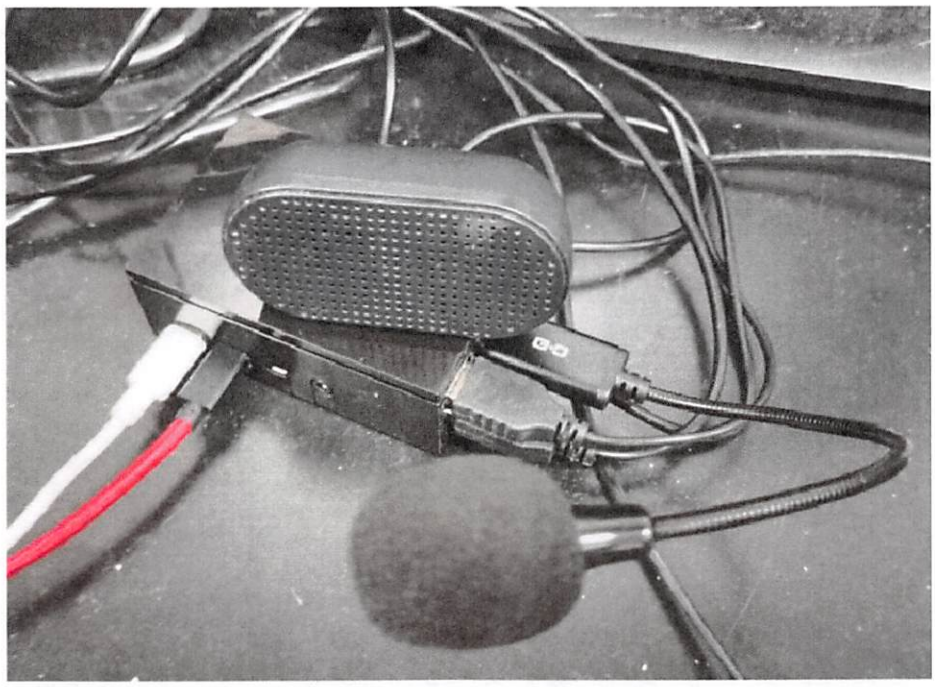

Figure 1: Picture of the Raspberry Pi

Overview of Main Code

The code works by running a main code which asks the user for either a 0, if they want to use text commands, or a 1, for voice commands. This then runs one of two separate programs, either a text-based code or a voice-based code, either of which corresponds to the input entered by the user. It will also ask if the user wants additional information, corresponding to inputting a 9 into the program. This will print off the information from the 'Readme.txt' file. If neither of the available options are chosen, the code will end.

The two versions function very similarly with the only difference being that the text version only uses "print" and "input" statements while the vocal version replaces the "input" and some "print" statements with a speech-to-text functions and text-to-speech functions.

Using either will print a list of commands for the user to select from. These include adding an item or box, removing an item or box, changing a quantity, displaying all items, and displaying additional information from the 'Readme.txt' file. Based on the input it will ask for necessary information, once provided it will modify the CSV file and return to the input selection. This will end when either the program is closed, or the user enters 'end 'into the program. Do note that the program lowercases the inputs, except for the section which is capitalized. This ensures that an item will not be skipped over if it's capitalized differently from the input.

Do note that to best describe where things are, the Moonbuggy room has been organized into sections. These sections are shown in a figure that helps understand where they are in relation to the room. A printout of this will be on display in the room to aid newcomers.

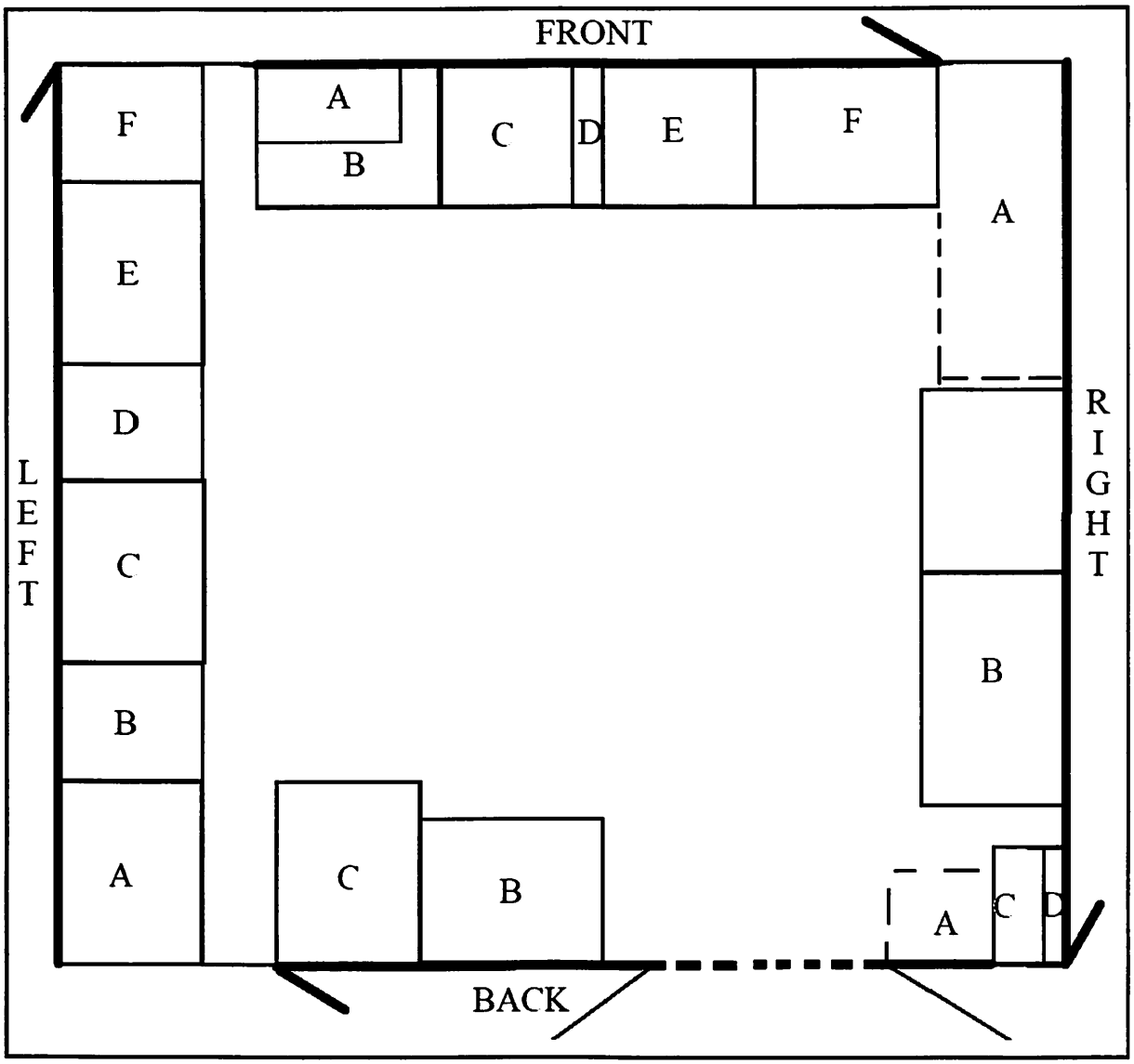

Figure 2: Sections of the Moonbuggy Room

Installing dependencies

The program does require several modules to be installed onto the Raspberry Pi and a specific operating system. This section specifies the commands used and the specifics. Do note that the command window may ask if it should install a large file; enter a 'y' and the download will proceed. This is intended to prevent accidental downloads of massive files, but the Micro-SD card will have ample space for all the necessities.

Raspberry Pi OS

The first step is to install the Raspberry Pi OS, this can be done by downloading the imager from www.raspberrypi.com/software/ running it will ask for the operating system to be installed, it is integral that the Raspberry Pi OS (Legacy) be installed. The Lite version may also work, but this project was done with the desktop environment.

Mozilla Deepspeech

Mozilla's Deepspeech is Mozilla Firefox's version of voice recognition. It has the added benefit of being able to work offline since it uses a downloaded model and scorer. It can be installed using the command:

'pip3 install deepspeech'

This will install it as a python module, if the Raspberry Pi has been updated or is not running the right operating system, it will return an error where it says that the GitHub file is not compatible with the operating system. This is because pip looks at a list of operating systems from the module download and compares it with the current operating system. If the OS is not on the list, it will not download it. The reason newer versions of the OS don't work is because the project has not been updated in years and by extension does not have newer operating systems on the list. This does include Windows 11 and as such, pip will not install DeepSpeech on a Windows 11 device.

For the module to work it also requires a voice model and a scorer. The model is a sample of the language for the program to compare to and the scorer is a file that shows the program what words go together and by extension what word is being said. This does make the

program much better at recognizing sentences, but makes it harder to recognize independent words. To get these, the following commands were used:

'wget https://github.com/mozilla/DeepSpeech/releases/download/v0.9.3/deepspeech-0.9.3-models.tflite'

'wget https://github.com/mozilla/DeepSpeech/releases/download/v0.9.3/deepspeech-0.9.3-models.scorer'.

Do make sure to get the .tflite file of the model for use on Raspberry Pi. There is also a .pbmm file which is used for larger computers. The difference is solely that the .pbmm is too large for the Raspberry Pi to work with and so it needs the .tflite file instead. These files are large and will take some time to download. Once downloaded, they must be in an easy to access location on the computer. This will be important when setting up the voice recognition.

Other Dependencies

For the Voice recognition to work it requires several other modules. These are used in the 'Getaudio.py' code to utilize DeepSpeech for current voice recognition.

The first module to install is pyaudio; to do this, run the following command into the command prompt. This will have an issue which can be fixed with the next command:

'pip3 install pyaudio'

'sudo apt-get install portaudio19-dev'.

Doing both will allow pyaudio to be used by the program, which allows the program to manipulate audio files.

Next the program needs three other modules, webrtcvad, halo, and scipy. These can be obtained by running the following commands,

'pip3 install webrtcvad'

'pip3 install halo'

'pip3 install scipy'

Espeak

Espeak is the program that allows the Raspberry Pi to output text-to-speech words. This can be installed with the command,

'sudo apt-get install espeak'

This will allow the command window to be used to spit out words. Espeak can be used at any time by running a command in the command window like such,

'espeak "Thing to be say" 2>/dev/null'

This will be used in a function to take the input and insert it into that command to create an audio output.

Information Regarding CSV File

What is a CSV

A CSV (Comma Separated Value) is essentially a text file where a comma separates the different columns. A better and more detailed description is to imagine creating a table in a spreadsheet. That spreadsheet can then be converted into a text file, but then all the columns would be messed up. Converting the spreadsheet to a CSV creates a column between every column. This lets programs, like Python, differentiate between different columns by reading the line.

MBDatabase.CSV

The CSV the program uses is called 'MBDatabase.CSV', the first row describes the 'columns' (Which are distinguished by a comma) they are as follows: Item, Quantity, Box, Section, Region, Day. This header does not determine what the code does, but is rather there to better read the file, either when using the display all or viewing it using a text viewer.

Creating Voice Functions

Text to Speech

The text to speech function ('TTS.py') is a very simple piece of code that takes a string of text as an input and places it into a larger string consisting of the commands for the command window to use and then sends it to the command window to be run. This is a modified version of the code used as an example in Dexter industries' article, *How To Make Your Raspberry Pi Speak.*[1] this article provides detail on how to set up Espeak on the Raspberry Pi and provides a example code.

Speech to Text

Setting up DeepSpeech is an arduous task. Luckily, Assembly AI has an article, *DeepSpeech for Dummies- A Tutorial and Overview* [2], which shows how to set up code either for voice recognition of an audio file or how to recognize speech dynamically. The latter was modified to function as a callable function that runs until it hears words, converts it to text, and then outputs a string of text. This can be seen in the Getaudio.py code as the main function. Doing it this way means that Getaudio.main() can be used as a voice alternative to the popular function, input.

Creating Text Functions

Find

The find command is possibly the most important function in the entire project as it finds a specific item and describes its location. It does this by opening the CSV file and reading through each line until it finds an item that matches the name of the item it's looking for. It'll then print the details to the screen. It will do this for every item that it finds with a similar name, just in case the same thing is stored in multiple locations.

Store

Store is a simple function that appends to the CSV. Appending a file is a way of simply adding a new line to the end of it. It does require all the information to be input so it can define the location of the item.

Moveitem

Moveitem is a function that uses the remove function (explained later) to remove an item from one box and uses store to put it into a new location. The quantity of the item is obtained in the remove command so it only requires the item, old box, new box, new section, and new region.

All

All is a similar function to find, where instead of only showing lines with the same name as the item it's looking for, it simply just shows all lines. This is handy to easily seeing the entire list of things in the Moonbuggy room although the entire list consists of almost 100 entries. Thus the need for other functions.

Remove

Remove is a function that takes an item and a box and removes it from the list. This does assume that each box name is unique and will have issues if there was a box and item with the same name located in two different parts of the room.

The function works by reading the CSV and writing a new CSV of the same name line by line, except it will skip any line that has the same item and box name as the inputs. This effectively removes the item from the list, but if there is an issue with the code, it may delete the data from the original CSV. Hence the need for backups. This is purely because the code can either read, write, or append, it cannot do multiple simultaneously.

An additional feature of the remove command is that it outputs the specifics of the item it removed. This allows it to assist in other functions by providing information on the item it previously removed.

Quantity

Quantity is a way of modifying an item's quantity. By removing the item and adding it back in using the same details except for an updated quantity.

Addbox

Addbox is a nifty way of adding 'boxes'. Boxes are intended to be unique and have an item name the same name as the box name. Additionally, the quantity is called 'box' to help distinguish it. Addbox is not a function, but rather a command that asks for less inputs to make cataloging easier. It then runs the store function to add the box. Every specific box should have one 'box' entity so the find function can show the box regardless of the items in it.

Removebox

Since boxes cannot be removed using the Remove function, a separate function had to be created where that exception is switched so that it only removes boxes. Like before it goes through the CSV and writes a new CSV without including the box that it's removing.

Info

In order to help new users, an Info function was created to read the Readme.txt file. This just loads it into python and

Backup

If the text or voice code gets a and 'end' command, it will take the CSV and create a new CSV with it with the only difference being that the date is added to the name. This will ideally prevent the Database from being completely lost in the event of an error.

Other

There are a few other functions used to aid in the code, however these are very straightforward and are just used to reduce repetition in the code. Unfortunately, this could not be done in some cases and some repetition is present in the code.

Voice Functions

It is worth nothing that the voice commands run a separate code of voice functions. These functions are nearly identical with the only difference being that they all use the Getaudio code for inputs and some use the TTS code for audio output. Display all does not use this since it would be rather unpleasant to sit through. Additionally the functions wait a second after the TTS code so the audio doesn't overlap with the audio unput.

Creating Main Code

MB_STORAGE_PROGRAM.py

The main program is essentially just an if statement that runs either the text code or voice code. It accepts a 1,0,9, or an end command. 0 is for text, 1 for voice, 9 for info, and end will run the backup command.

MBText.py

The text code lists 11 possible commands, numbered 0-9 and an 'end' command. Numbers 1-9 correspond to the functions shown above and ask for their respective information. Number 0 runs the info command and lastly, 'end' will create a backup and end the program.

MBVoice.py

The voice code works almost identically to the text code, but uses voice commands for the inputs and text-to-speech for the outputs. It also has its own functions that use the voice commands and text-to-speech whereas the text functions will solely use 'input' and 'print' commands.

Conclusion

There were many difficulties in putting Mozilla's DeepSpeech voice commands on a Raspberry Pi, but it is possible and having done so has allowed for a storage aid to be made. Alternatively, it has been designed to be functional without the need for audio inputs or outputs. The design has been capable of keeping track of items in the Moonbuggy room as well as the many functions listed in the report. Hopefully this project will be useful to others as not only a storage aid, but also a guide on implementing voice commands and text-to-speech on a Raspberry Pi.

References

[1] "How to Make Your Raspberry Pi Speak out Loud: Give It a Voice!" *Dexter Industries*, https://www.dexterindustries.com/howto/make-your-raspberry-pi-speak/.

[2] Tang, Yujian. "Deepspeech for Dummies - A Tutorial and Overview." *News, Tutorials, AI Research*, News, Tutorials, AI Research, 5 Nov. 2021, https://www.assemblyai.com/blog/deepspeech-for-dummies-a-tutorial-and-overview-part-1/.

Appendix A: MB_STORAGE_PROGRAM.py

```
#MB_STORAGE_PROGRAM
#Riley W Cavitt
#4/11/2023
#Uses an input to either run the text or voice code
#That code will then modify the MBDatabase.CSV
#Hopefully this will be a useful tool for Moonbuggy students

#IMPORTANT THINGS
import MBfunctions as MB
import MBVoice as MBV
import MBText as MBT
import time
#Initialize
x=input('Select Mode of operation:\n0 Text input\n1:voice input\n9: Info\n')
x=x.lower()
if x=='1' or x=='one' or x=='voice': #Runs vocal input
    MBV.Main()

elif x=='0' or x=='zero' or x=='text': #Runs txt input
    MBT.Main()

elif x=='9' or x=='nine' or x=='info':
    MB.Info()
else: #neither option was chosen
    print('invalid option')
time.sleep(1)
```

Appendix B: MBText.py

```
#MBText
#Riley W Cavitt
#4/10/2023
#Runs through a list of options and inputs to determine which function to use
#in order to modify the MBDatabase.CSV
#If end command is given, it backs up CSV to file with date on it.
import MBfunctions as MB
import time
def Main():
    #Main Text based MB storage program
    x=1
    print('Starting MB storage program')
    while x==1:
        print('What would you like me to do?\n ')
        print('1: Find an item\n')
        print('2: Store an item\n')
        print('3: Move an item\n')
        print('4: Move a box\n')
        print('5: Display all\n')
        print('6: Remove an item\n')
        print('7: Modify quantity\n')
        print('8: Add a box\n')
        print('9: Remove a box\n')
        print('0: Info\n')
        print('END: Ends program\n')
        y=input('input a number and hit Enter\n')
        y=y.lower()
        if y=='1' or y=='one' or y=='find an item': #Find an item
            #Will display all items with the same name and their locations
            item=input('\nWhat would you like to find?\n')
            MB.Find(item.lower())
            time.sleep(5)
        elif y=='2' or y=='two' or y=='store an item': #Store an item
            #Will store an item based on the details provided
            item=input('What would you like to store?\n')
            quantity=input('How many?\n')
            box=input('what box is the thing in?\n')
            section=input('what section is the item in?\n')
            region=input('what region of the room?\n')

MB.Store(item.lower(),quantity.lower(),box.lower(),section.upper(),region.lower())
        elif y=='3' or y=='three' or y=='move an item': #Move an item
            #Will relocate an item into a new box
            item=input('What item are you moving?')
```

```
        startbox=input('What box is it currently in?')
        endbox=input('What box is it going in?')
        section=input('what section?')
        region=input('what region?')

MB.Moveitem(item.lower(),startbox.lower(),endbox.lower(),section.lower(),region.lower
())
    elif y=='4' or y=='four' or y=='move a box': #Move a box
        #Move a box and its contents to a new location
        box=input('What box do you want to move?\n')
        newsection=input('What section?')
        newregion=input('What region?')
        MB.Movebox(box.lower(),newsection.lower(),newregion.lower())
    elif y=='5' or y=='five' or y=='display all': #Display all
        #Shows a list of every item and their details
        MB.All()
        #time.sleep(5) #optional time sleep, I don't like to use this
    elif y=='6' or y=='six' or y=='remove an item': #Remove an item
        #Removes an item from the list
        item=input('What item are you trying to remove?')
        box=input('What box did the item come from?')
        MB.Remove(item.lower(),box.lower())
    elif y=='7' or y=='seven' or y=='modify a quanitity': #Modify a quantity
        item=input('What item are you modifying')
        box=input('What box is it from?')
        quantity=input('What is the update quantity?')
        MB.quantity(item,box,quantity)
    elif y=='8' or y=='eight' or y=='add a box': #Add a box
        #Adds a an item as the same name of the box
        item=input('What box are you adding?\n')
        quantity='box'
        box=item
        section=input('what section is the item in?\n')
        region=input('what region of the room?\n')

MB.Store(item.lower(),quantity.lower(),box.lower(),section.upper(),region.lower())
    elif y=='9' or y=='nine' or y=='remove a box': #Remove a box
        box=input('What box are you removing?')
        MB.Removebox(str(box))
    elif y=='0' or y=='zero' or y=='info':
        MB.Info()
    elif y=='END' or y=='end' or y=='End': #Ends the program
        x=0 #Ends while loop
        MB.Backup()
    else: #Something was inputted that is not on the list
        MB.Invalid()
```

```
        time.sleep(1)

#Main() #Runs main function can be used to bypass main function
```

Appendix C: MBVoice.py

```
#MBVoice
#Riley W Cavitt
#4/10/2023
#modified version of MBText to use voice inputs and provide audio outputs

import MBVoiceFunctions as MB
import time
import TTS
import Getaudio as DS
def Vnput(ask):
    #One command to say line and then obtain audio input:
    TTS.main(str(ask))
    time.sleep(1)
    x=DS.main()
    return x

def Main():
    #Main Text based MB storage program
    x=1
    print('Starting MB storage program')
    while x==1:

        print('What would you like me to do?\n ')
        print('1: Locate an item\n')
        print('2: Store an item\n')
        print('3: Move an item\n')
        print('4: Move a box\n')
        print('5: Display all\n')
        print('6: Remove an item\n')
        print('7: Modify quantity\n')
        print('8: Add a box\n')
        print('9: Remove a box\n')
        print('0: Info\n')
        print('END: Ends program\n')
        #y=input('input a number and hit Enter\n')
        TTS.main('Choose a selection by entering a voice command')
        y=DS.main()

        if y=='1': #Find an item
            #Will display all items with the same name and their locations
            item=Vnput('\nWhat would you like to find?\n')
            MB.Find(item.lower())
            time.sleep(5)
        elif y=='2': #Store an item
```

```
            #Will store an item based on the details provided
            item=Vnput('What would you like to store?\n')
            quantity=Vnput('How many?\n')
            box=Vnput('what box is the thing in?\n')
            section=Vnput('what section is the item in?\n')
            region=Vnput('what region of the room?\n')

MB.Store(item.lower(),quantity.lower(),box.lower(),section.upper(),region.lower())
        elif y=='3': #Move an item
            #Will relocate an item into a new box
            item=Vnput('What item are you moving?')
            startbox=Vnput('What box is it currently in?')
            endbox=Vnput('What box is it going in?')
            section=Vnput('What section?')
            region=Vnput('What region?')

MB.Moveitem(item.lower(),startbox.lower(),endbox.lower(),section.lower(),region.lower
())
        elif y=='4': #Move a box
            #Will a box and its contents to a new location
            box=Vnput('What box do you want to move?\n')
            section=Vnput('What section?')
            region=('What region?')
            MB.Movebox(box.lower(),section.lower(),region.lower())
        elif y=='5': #Display all
            #Shows a list of every item and their details
            MB.All()
            time.sleep(5)
        elif y=='6': #Remove an item
            #Removes an item from the list
            item=Vnput('What item are you trying to remove?')
            box=Vnput('What box did the item come from?')
            MB.Remove(item.lower(),box.lower())
        elif y=='7': #Modify a quantity
            item=Vnput('What item are you modifying')
            box=Vnput('What box is it from?')
            quantity=Vnput('What is the update quantity?')
            MB.quantity(item,box,quantity)
        elif y=='8': #Add a box
            #Adds a an item as the same name of the box
            item=Vnput('What box are you adding?\n')
            quantity='box'
            box=item
            section=Vnput('what section is the item in?\n')
            region=Vnput('what region of the room?\n')
```

```
MB.Store(item.lower(),quantity.lower(),box.lower(),section.upper(),region.lower())
    elif y=='9': #Remove a box
        box=Vnput('What box are you removing?')
        MB.Removebox(str(box))
    elif y=='0':
        MB.Info()
    elif y=='END' or y=='end' or y=='End': #Ends the program
        x=0 #Ends while loop
        MB.Backup()
    elif y!='':
        print('\n\nnot input, but got:')
        print(y)
        print('\n\n')
    else: #Something was inputted that is not on the list
        MB.Invalid()
        time.sleep(1)

#Main() #Runs main function can be used to bypass main function
```

Appendix D: MBFunctions.py

```
#MBFunctions
#Riley W Cavitt
#4/10/2023
#A variety of functions to store, move and remove items and boxes from
#the MBDatabase.CSV

import csv
import datetime as dt
from datetime import date

def quantity(item,box,newquan):
    #Changes item's quantity
    details=Remove(item,box)
    Store(item,newquan,box,details[3],details[4])
    return

def Find(x):
    #Finds a specific item in the moonbuggy room.
    Things={}
    found=0
    with open('MBDatabase.csv',encoding='utf-8',newline='') as f:
        reader=enumerate(csv.reader(f)) #Set up reader
        f.readline()
        for i, row in reader: #For all rows...

newdict=dict({'Item':row[0],'Quantity':row[1],'Box':str(row[2]),'Section':str(row[3]),'Regi
on':str(row[4]),'Date':str(row[5])})
            Things[i+1]=newdict
        for item in Things.keys(): #Print data

            id=item
            if Things[item]['Item']==x:
                Location(Things,id)
                found=1
        if found==0:
            print('Item not found')
    return

def Location(Things,i):
    item=i
    id=item
    print('There are',Things[item]['Quantity'],Things[item]['Item'],\
        '\nLocated in the', Things[item]['Box'], 'Box.\nIn section:',Things[item]['Section'],\
        '\nLocated in the',Things[item]['Region'],'Of the room. \nIt was catalogged on:',\
```

```
        Things[item]['Date'],'\n\n')
    return

def Store(item,quantity,box,section,region):
    #Displays all contents of the moonbuggy room.
    Things={}
    with open('MBDatabase.csv',"a",encoding='utf-8',newline='') as f:
        writer=csv.writer(f) #Set up reader
        day=date.today() #Gets current date
        writer.writerow([item,quantity,box,section,region,day])
    return

def Invalid():
    print('Invalid Selection\n')
    return

def Return():
    print('Returning to main selection\n')
    return

def All():
    #Displays all contents of the moonbuggy room.
    Things={}
    with open('MBDatabase.csv',encoding='utf-8',newline='') as f:
        reader=enumerate(csv.reader(f)) #Set up reader
        f.readline()
        for i, row in reader: #For all rows...

newdict=dict({'Item':row[0],'Quantity':str(row[1]),'Box':str(row[2]),'Section':str(row[3]),'
Region':str(row[4]),'Date':str(row[5])})
            Things[i+1]=newdict
        print('ID, Item, Quantity, Box, Section, Region, Date')
        for item in Things.keys(): #Print data
            id=item
            print(id,Things[item]['Item'],\
                Things[item]['Quantity'],\
                Things[item]['Box'],\
                Things[item]['Section'],\
                Things[item]['Region'],\
                Things[item]['Date'])
    return

def Remove(item,box):
    #Finds a specific item in the moonbuggy room.
    x=item
    y=box
```

```
    Things={}
    found=0
    details=0
    with open('MBDatabase.csv',encoding='utf-8',newline='') as f:
        reader=enumerate(csv.reader(f)) #Set up reader
        f.readline()
        for i, row in reader: #For all rows...

newdict=dict({'Item':row[0],'Quantity':row[1],'Box':str(row[2]),'Section':str(row[3]),'Regi
on':str(row[4]),'Date':str(row[5])})
            Things[i+1]=newdict
    with open('MBDatabase.csv','w',encoding='utf-8',newline='') as f:
        writer=csv.writer(f) #Set up reader
        day=date.today()
        writer.writerow(['Item','Quantity','Box','Section','Region','Day'])
        for i in Things.keys(): #Print data
            id=i
            item=Things[i]['Item']
            quantity=Things[i]['Quantity']
            box=Things[i]['Box']
            section=Things[i]['Section']
            region=Things[i]['Region']
            day=Things[i]['Date']
            if Things[i]['Item']==x and Things[i]['Box']==y and Things[i]['Quantity']!='box':
                found=1
                q=Things[i]['Quantity']
                s=Things[i]['Section']
                r=Things[i]['Region']
                details=[item,q,box,s,r]
            else:
                writer.writerow([item,quantity,box,section,region,day])
        if found==0:
            print('Item not found')
        else:
            print('Item found')
    return details

def Moveitem(item,startbox,endbox,section,region):
    details=Remove(item,startbox) #Use remove to remove original location
    Store(item,details[1],endbox,section,region) #Add item in new location
    return

def Info():
    #Reads off the Readme.txt file
    with open('Readme.txt','r') as f:
        Lines=f.read()
```

```
        print(Lines)
    return

def Movebox(box,newsection,newregion):
    Removebox(box) #Remove box
    Store(box,'box',box,newsection,newregion) #Readd box in new location
    return


def Removebox(box):
    #Finds a specific item in the moonbuggy room.
    y=str(box)
    Things={}
    found=0
    details=0
    with open('MBDatabase.csv',encoding='utf-8',newline='') as f:
        reader=enumerate(csv.reader(f)) #Set up reader
        f.readline()
        for i, row in reader: #For all rows...

newdict=dict({'Item':row[0],'Quantity':row[1],'Box':str(row[2]),'Section':str(row[3]),'Regi
on':str(row[4]),'Date':str(row[5])})
            Things[i+1]=newdict
    with open('MBDatabase.csv','w',encoding='utf-8',newline='') as f:
        writer=csv.writer(f) #Set up reader
        day=date.today()
        writer.writerow(['Item','Quantity','Box','Section','Region','Day'])
        for i in Things.keys(): #Print data
            id=i
            item=Things[i]['Item']
            quantity=Things[i]['Quantity']
            box=Things[i]['Box']
            section=Things[i]['Section']
            region=Things[i]['Region']
            day=Things[i]['Date']
            if Things[i]['Box']==y and Things[i]['Quantity']=='box':
                found=1
                q=Things[i]['Quantity']
                s=Things[i]['Section']
                r=Things[i]['Region']
                details=[item,q,box,s,r]
            else:
                writer.writerow([item,quantity,box,section,region,day])
        if found==0:
            print('Item not found')
        else:
```

```
            print('Item found')
    return details


def Backup():
    #Finds a specific item in the moonbuggy room.
    Things={}
    found=0
    details=0
    with open('MBDatabase.csv',encoding='utf-8',newline='') as f:
        reader=enumerate(csv.reader(f)) #Set up reader
        f.readline()
        for i, row in reader: #For all rows...

newdict=dict({'Item':row[0],'Quantity':row[1],'Box':str(row[2]),'Section':str(row[3]),'Regi
on':str(row[4]),'Date':str(row[5])})
            Things[i+1]=newdict
        name1='MBDatabase'
        name2=str(date.today())
        newname=name1+name2 #Combine MBDatabase and current date
    with open(newname,'w',encoding='utf-8',newline='') as f:
        writer=csv.writer(f) #Set up reader
        day=date.today()
        writer.writerow(['Item','Quantity','Box','Section','Region','Day'])
        for i in Things.keys(): #Print data
            id=i
            item=Things[i]['Item']
            quantity=Things[i]['Quantity']
            box=Things[i]['Box']
            section=Things[i]['Section']
            region=Things[i]['Region']
            day=Things[i]['Date']
            writer.writerow([item,quantity,box,section,region,day])
    return
```

Appendix E: MBVoiceFunctions.py

```
#MBVoice Functions
#Riley W Cavitt
#4/10/2023
#Modified version of text functions to be used with
#Deepspeech and Espeak

import csv
import datetime as dt
from datetime import date
import TTS
import Getaudio as DS
import time

def quantity(item,box,newquan):
    #Changes item's quantity
    details=Remove(item,box)
    Store(item,newquan,box,details[3],details[4])
    return

def Find(x):
    #Finds a specific item in the moonbuggy room.
    Things={}
    found=0
    with open('MBDatabase.csv',encoding='utf-8',newline='') as f:
        reader=enumerate(csv.reader(f)) #Set up reader
        f.readline()
        for i, row in reader: #For all rows...

newdict=dict({'Item':row[0],'Quantity':row[1],'Box':str(row[2]),'Section':str(row[3]),'Regi
on':str(row[4]),'Date':str(row[5])})
            Things[i+1]=newdict
        for item in Things.keys(): #Print data

            id=item
            if Things[item]['Item']==x:
                Location(Things,id)
                found=1
        if found==0:
            TTS.main('Item not found')
    return

def Location(Things,i):
    item=i
    id=item
```

```
    TTS.main('There are',Things[item]['Quantity'],Things[item]['Item'],\
        '\nLocated in the', Things[item]['Box'], 'Box.\nIn section:',Things[item]['Section'],\
        '\nLocated in the',Things[item]['Region'],'Of the room. \nIt was catalogged on:',\
        Things[item]['Date'],'\n\n')
    time.sleep(1)
    return

def Store(item,quantity,box,section,region):
    #Displays all contents of the moonbuggy room.
    Things={}
    with open('MBDatabase.csv',"a",encoding='utf-8',newline='') as f:
        writer=csv.writer(f) #Set up reader
        day=date.today() #Gets current date
        writer.writerow([item,quantity,box,section,region,day])
    return

def Invalid():
    TTS.main('Invalid Selection\n')
    return

def Return():
    TTS.main('Returning to main selection\n')
    return

def All():
    #Displays all contents of the moonbuggy room.
    Things={}
    with open('MBDatabase.csv',encoding='utf-8',newline='') as f:
        reader=enumerate(csv.reader(f)) #Set up reader
        f.readline()
        for i, row in reader: #For all rows...

newdict=dict({'Item':row[0],'Quantity':str(row[1]),'Box':str(row[2]),'Section':str(row[3]),'
Region':str(row[4]),'Date':str(row[5])})
            Things[i+1]=newdict
        print('ID, Item, Quantity, Box, Section, Region, Date')
        for item in Things.keys(): #Print data
            id=item
            print(id,Things[item]['Item'],\
                Things[item]['Quantity'],\
                Things[item]['Box'],\
                Things[item]['Section'],\
                Things[item]['Region'],\
                Things[item]['Date'])
    return
```

```
def Remove(item,box):
    #Finds a specific item in the moonbuggy room.
    x=item
    y=box
    Things={}
    found=0
    details=0
    with open('MBDatabase.csv',encoding='utf-8',newline='') as f:
        reader=enumerate(csv.reader(f)) #Set up reader
        f.readline()
        for i, row in reader: #For all rows...

newdict=dict({'Item':row[0],'Quantity':row[1],'Box':str(row[2]),'Section':str(row[3]),'Regi
on':str(row[4]),'Date':str(row[5])})
            Things[i+1]=newdict
    with open('MBDatabase.csv','w',encoding='utf-8',newline='') as f:
        writer=csv.writer(f) #Set up reader
        day=date.today()
        writer.writerow(['Item','Quantity','Box','Section','Region','Day'])
        for i in Things.keys(): #Print data
            id=i
            item=Things[i]['Item']
            quantity=Things[i]['Quantity']
            box=Things[i]['Box']
            section=Things[i]['Section']
            region=Things[i]['Region']
            day=Things[i]['Date']
            if Things[i]['Item']==x and Things[i]['Box']==y and Things[i]['Quantity']!='box':
                found=1
                q=Things[i]['Quantity']
                s=Things[i]['Section']
                r=Things[i]['Region']
                details=[item,q,box,s,r]
            else:
                writer.writerow([item,quantity,box,section,region,day])
        if found==0:
            TTS.main('Item not found')
        else:
            TTS.main('Item found')
    return details

def Moveitem(item,startbox,endbox,section,region):
    details=Remove(item,startbox) #Use remove to remove original location
    Store(item,details[1],endbox,section,region) #Add item in new location
    return
```

```
def Info():
    #Reads off the Readme.txt file
    with open('Readme.txt','r') as f:
        Lines=f.read()
        TTS.main(Lines)
    return

def Movebox(box,newsection,newregion):
    Removebox(box) #Remove box
    Store(box,'box',box,newsection,newregion) #Readd box in new location
    return

def Removebox(box):
    #Finds a specific item in the moonbuggy room.
    y=str(box)
    Things={}
    found=0
    details=0
    with open('MBDatabase.csv',encoding='utf-8',newline='') as f:
        reader=enumerate(csv.reader(f)) #Set up reader
        f.readline()
        for i, row in reader: #For all rows...

newdict=dict({'Item':row[0],'Quantity':row[1],'Box':str(row[2]),'Section':str(row[3]),'Regi
on':str(row[4]),'Date':str(row[5])})
            Things[i+1]=newdict
    with open('MBDatabase.csv','w',encoding='utf-8',newline='') as f:
        writer=csv.writer(f) #Set up reader
        day=date.today()
        writer.writerow(['Item','Quantity','Box','Section','Region','Day'])
        for i in Things.keys(): #Print data
            id=i
            item=Things[i]['Item']
            quantity=Things[i]['Quantity']
            box=Things[i]['Box']
            section=Things[i]['Section']
            region=Things[i]['Region']
            day=Things[i]['Date']
            if Things[i]['Box']==y and Things[i]['Quantity']=='box':
                found=1
                q=Things[i]['Quantity']
                s=Things[i]['Section']
                r=Things[i]['Region']
                details=[item,q,box,s,r]
            else:
                writer.writerow([item,quantity,box,section,region,day])
```

```
        if found==0:
            TTS.main('Item not found')
        else:
            TTS.main('Item found')
    return details


def Backup():
    #Finds a specific item in the moonbuggy room.
    Things={}
    found=0
    details=0
    with open('MBDatabase.csv',encoding='utf-8',newline='') as f:
        reader=enumerate(csv.reader(f)) #Set up reader
        f.readline()
        for i, row in reader: #For all rows...

newdict=dict({'Item':row[0],'Quantity':row[1],'Box':str(row[2]),'Section':str(row[3]),'Regi
on':str(row[4]),'Date':str(row[5])})
            Things[i+1]=newdict
        name1='MBDatabase'
        name2=str(date.today())
        newname=name1+name2 #Combine MBDatabase and current date
    with open(newname,'w',encoding='utf-8',newline='') as f:
        writer=csv.writer(f) #Set up reader
        day=date.today()
        writer.writerow(['Item','Quantity','Box','Section','Region','Day'])
        for i in Things.keys(): #Print data
            id=i
            item=Things[i]['Item']
            quantity=Things[i]['Quantity']
            box=Things[i]['Box']
            section=Things[i]['Section']
            region=Things[i]['Region']
            day=Things[i]['Date']
            writer.writerow([item,quantity,box,section,region,day])
    return
```

Appendix F: Getaudio.py

```
#Getaudio
#Riley W Cavitt
#4/11/2023
#Modified version of Assembly AI Deepspeech Code
#when Main() is run, it will get an word from speech and return a string


import pyaudio
import deepspeech
import webrtcvad
import halo
from halo import Halo
import numpy as np
import scipy
import queue
import collections

class VADAudio(object):
  """Filter & segment audio with voice activity detection."""

  FORMAT = pyaudio.paInt16
  # Network/VAD rate-space
  RATE_PROCESS = 16000
  CHANNELS = 1
  BLOCKS_PER_SECOND = 50

  def __init__(self, callback=None, device=None, input_rate=RATE_PROCESS,
file=None, aggressiveness=3):
        def proxy_callback(in_data, frame_count, time_info, status):
          if self.chunk is not None:
            in_data = self.wf.readframes(self.chunk)
          callback(in_data)
          return (None, pyaudio.paContinue)
        if callback is None: callback = lambda in_data: self.buffer_queue.put(in_data)
        self.buffer_queue = queue.Queue()
        self.device = device
        self.input_rate = input_rate
        self.sample_rate = self.RATE_PROCESS
        self.block_size = int(self.RATE_PROCESS /
float(self.BLOCKS_PER_SECOND))
        self.block_size_input = int(self.input_rate / float(self.BLOCKS_PER_SECOND))
        self.pa = pyaudio.PyAudio()
        self.vad = webrtcvad.Vad(aggressiveness)
```

```
        kwargs = {
            'format': self.FORMAT,
            'channels': self.CHANNELS,
            'rate': self.input_rate,
            'input': True,
            'frames_per_buffer': self.block_size_input,
            'stream_callback': proxy_callback,
        }

        self.chunk = None
        # if not default device
        if self.device:
            kwargs['input_device_index'] = self.device
        elif file is not None:
            self.chunk = 320
            self.wf = wave.open(file, 'rb')

        self.stream = self.pa.open(**kwargs)
        self.stream.start_stream()
    #============================
    def resample(self, data, input_rate):
        """
        Microphone may not support our native processing sampling rate, so
        resample from input_rate to RATE_PROCESS here for webrtcvad and
        deepspeech

        Args:
            data (binary): Input audio stream
            input_rate (int): Input audio rate to resample from
        """
        data16 = np.fromstring(string=data, dtype=np.int16)
        resample_size = int(len(data16) / self.input_rate * self.RATE_PROCESS)
        resample = signal.resample(data16, resample_size)
        resample16 = np.array(resample, dtype=np.int16)
        return resample16.tostring()
    #============================
    def read_resampled(self):
        """Return a block of audio data resampled to 16000hz, blocking if necessary."""
        return self.resample(data=self.buffer_queue.get(),
                             input_rate=self.input_rate)

    def read(self):
        """Return a block of audio data, blocking if necessary."""
        return self.buffer_queue.get()
    #================================
```

```
  def write_wav(self, filename, data):
        logging.info("write wav %s", filename)
        wf = wave.open(filename, 'wb')
        wf.setnchannels(self.CHANNELS)
        # wf.setsampwidth(self.pa.get_sample_size(FORMAT))
        assert self.FORMAT == pyaudio.paInt16
        wf.setsampwidth(2)
        wf.setframerate(self.sample_rate)
        wf.writeframes(data)
        wf.close()

#====================================

  frame_duration_ms = property(lambda self: 1000 * self.block_size // self.sample_rate)

#====================================

  def frame_generator(self):
        """Generator that yields all audio frames from microphone."""
        if self.input_rate == self.RATE_PROCESS:
            while True:
                yield self.read()
        else:
            while True:
                yield self.read_resampled()
#====================================
  def vad_collector(self, padding_ms=300, ratio=0.75, frames=None):
        """Generator that yields series of consecutive audio frames comprising each
utterence, separated by yielding a single None.
            Determines voice activity by ratio of frames in padding_ms. Uses a buffer to
include padding_ms prior to being triggered.
            Example: (frame, ..., frame, None, frame, ..., frame, None, ...)
                      |---utterence---|        |---utterence---|
        """
        if frames is None: frames = self.frame_generator()
        num_padding_frames = padding_ms // self.frame_duration_ms
        ring_buffer = collections.deque(maxlen=num_padding_frames)
        triggered = False

        for frame in frames:
            if len(frame) < 640:
                return

            is_speech = self.vad.is_speech(frame, self.sample_rate)

            if not triggered:
```

```
            ring_buffer.append((frame, is_speech))
            num_voiced = len([f for f, speech in ring_buffer if speech])
            if num_voiced > ratio * ring_buffer.maxlen:
                triggered = True
                for f, s in ring_buffer:
                    yield f
                ring_buffer.clear()

        else:
            yield frame
            ring_buffer.append((frame, is_speech))
            num_unvoiced = len([f for f, speech in ring_buffer if not speech])
            if num_unvoiced > ratio * ring_buffer.maxlen:
                triggered = False
                yield None
                ring_buffer.clear()
#================================
def main():
  DEFAULT_SAMPLE_RATE = 16000
  # Load DeepSpeech model
  #CHANGE THE NEXT TWO LINES TO THE LOCATION OF YOUR MODEL
AND SCORER
  model = 'C:/users/riley/My stuff/Python/DS TEST/deepspeech-0.9.3-models.pbmm'
  scorer = 'C:/users/riley/My stuff/Python/DS TEST/deepspeech-0.9.3-models.scorer'
  gotword=0
  #print('Initializing model...')
  #print("model: %s", model)
  model = deepspeech.Model(model)
  if scorer:
    #print("scorer: %s", scorer)
    model.enableExternalScorer(scorer)

  # Start audio with VAD
  vad_audio = VADAudio(aggressiveness=3,
                       device=None,
                       input_rate=DEFAULT_SAMPLE_RATE,
                       file=None)
  print("Listening (ctrl-C to exit)...")
  frames = vad_audio.vad_collector()

  # Stream from microphone to DeepSpeech using VAD
  spinner = Halo(spinner='line')
  stream_context = model.createStream()
  wav_data = bytearray()
  for frame in frames:
    if frame is not None:
```

```
            if spinner: spinner.start()
            stream_context.feedAudioContent(np.frombuffer(frame, np.int16))
        else:
            if spinner: spinner.stop()
            print("end utterence")
            text = stream_context.finishStream()
            print("Recognized: %s" % text)
            stream_context = model.createStream()
            gotword=1
            return text
    return text
if __name__ == '__main__':
    DEFAULT_SAMPLE_RATE = 16000
    #x=str(main())
   # print(x)
#x=main()
#print(x)
```

Appendix G: TTS.py

```
#TTS code:
#Riley W Cavitt
#4/11/2023
#Outputs an generated audio from a string
#Uses Espeak
#ONLY WORKS ON RASPBERRY PI
from subprocess import call


def main(text):
    cmd_beg= 'espeak '
    #Adds the necesary beginning to the command
    print(text)


    #Replacing ' ' with '_' to identify words in the text entered
    text = text.replace(' ', '_')
    #Outputs a command to command prompt for TTS output
    call([cmd_beg+text], shell=True)
    return
#TTS(This is a sample text)
```

Appendix H: Readme.txt

MB STORAGE PROGRAM information:
Created by: Riley W Cavitt
Created in: Spring 2023
For use in the UAH Moonbuggy Room

This program is designed to modify a CSV file (Comma seperated Value)
To provide information on where things are in the Moonbuggy room.
It should be used alongside the reference sheet detailing the various
sections of the room.

There are many functions which are designed to do the various things
necesary to manipulating the CSV file.
These will be shown after chosing either to use text or voice commands.

entering will cause the codee to backup the database with the current
date in its title.