Sofia Kolobaev and Julia Rasmussen
Professor Roi Yehoshua
Programming Fundamentals
02 May 2022

# Final Project Report

## 1. Introduction:

We decided to make a Twitter bot for our final project. This Twitter bot takes tweets that it finds and processes them using an algorithm that we designed, turning the text within a tweet into a .wav file and converting that into an .mp4 file. Our program then takes that .mp4 file and uploads it directly to Twitter in the form of a tweet using the developer account tied to the bot.

In order to make this program, we first came up with the idea we wanted to pursue: the music Twitter bot. Then, we determined what we wanted the function of our Twitter bot to be; in this case, it was to be able to take a singular tweet, convert it to a .wav file, and tweet that .wav file back out in a format that Twitter would allow. Correspondingly, we figured out what types of libraries we'd need in order to accomplish this and solidified our idea in our project proposal. Once we had this idea, we met up a couple times a week to work on the code and to try to establish what methods we'd need to incorporate from the libraries we were using. We worked through various bugs together, trying to troubleshoot how to write the code in a way that would work (i.e. applying for Twitter development permissions, translating FFmpeg commands into Python syntax, changing audio control methods, etc). Following our completion of the code, we tested it, built upon it, and deployed it. Our project was overall successful, resulting in a functional Twitter bot that is able to take a Tweet, convert the text to audio, and then Tweet it.

## 2. Application Design:

The main classes for this application are BotTweet, TextControl, and AudioControl. The BotTweet class controls the actual implementation of the Twitter API. It validates the session, finds a Tweet given set search parameters, and feeds the text from that Tweet into the TextControl class, which parses it, and then feeds the relevant data to AudioControl, which makes a .wav file out of it. This .wav file is then converted to an .mp4 in BotTweet, and then

uploaded to Twitter alongside the original text of the tweet and an added message, after which both files are deleted.

BotTweet's main function is generate_tweet(), which, when called, accesses all the other functions within BotTweet and TextControl needed to run the program. BotTweet is initialized with the keys needed to access the Twitter API as strings, which it can then call on as needed when connecting to Twitter to find the original Tweet in find_tweet(), and when uploading the Tweet in upload_tweet(). BotTweet is initialized with a TextControl object, which it can then use to parse the Tweet and generate the audio file.

The TextControl class takes the text of the original Tweet and processes the words in the text in the convert_text_to_wav() function, finding the ASCII value of each character and converting the values into an array of semitones in the convert_word_to_semitones() function, as well as an array of durations. TextControl is also initialized with a AudioControl object, and so once it has everything, it simply sends the relevant details (notes, durations, and word length) to the generate_sound() function within the AudioControl object.

The generate_sound() function then turns the information into a .wav file by scaling the frequency of each subsequent note onto a major scale with the get_frequency() function, and then generating sine waves with NumPy in the generate_wave() function, using these frequencies. This sine wave is concatenated into an array and imported into a .wav file with the Wave library, and saved to the file directory, where it can then be accessed by BotTweet, which then uses ffmpy to convert the .wav into an .mp4 file.

## 3. Class diagram

The class diagram for the program can be seen below in Figure 1. As can be seen in the diagram, BotTweet and TextControl are aggregated with TextControl and AudioControl, respectively. BotTweet relies on TextControl to deal with the inputted text, and TextControl relies on AudioControl to then make the .wav file.

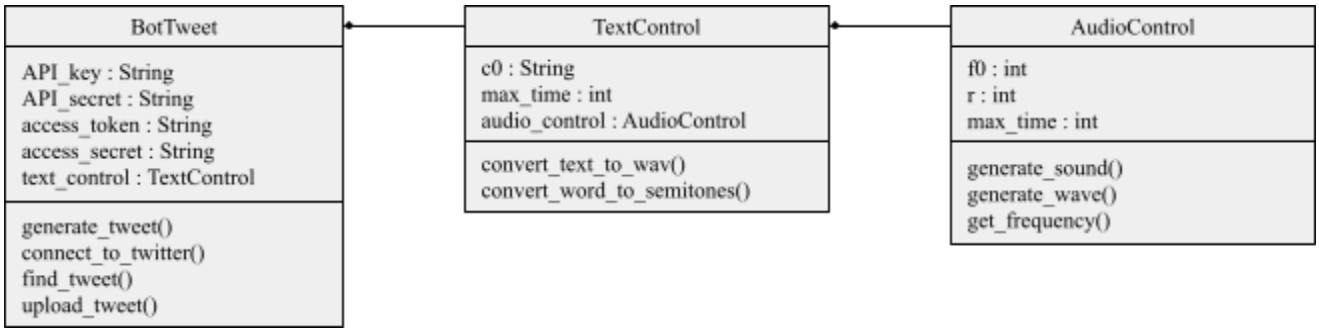


Figure 1: The class diagram for the program.

## 4. GitHub

The GitHub repository for our project is https://github.com/Music-Twitter-Bot/twitter_bot_code. Additionally, the commit history can be found below, in Figures 2 and 3. Note that the commit history does not accurately reflect the contributions of each team member, as the code was worked on together.

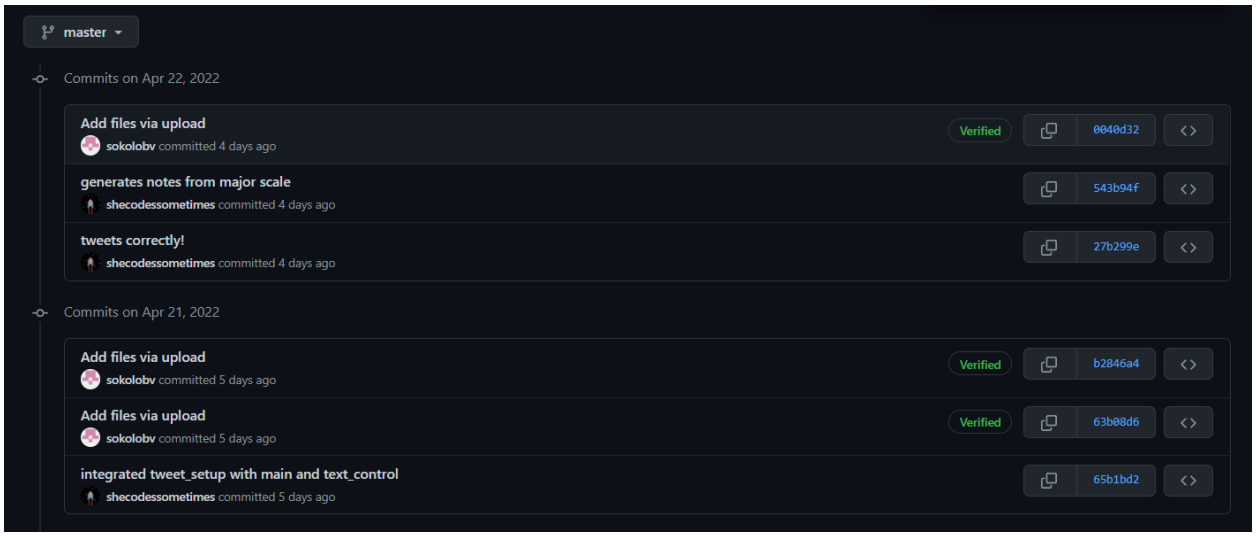


Figure 2: Commits to the repository from April 21st to April 22nd.

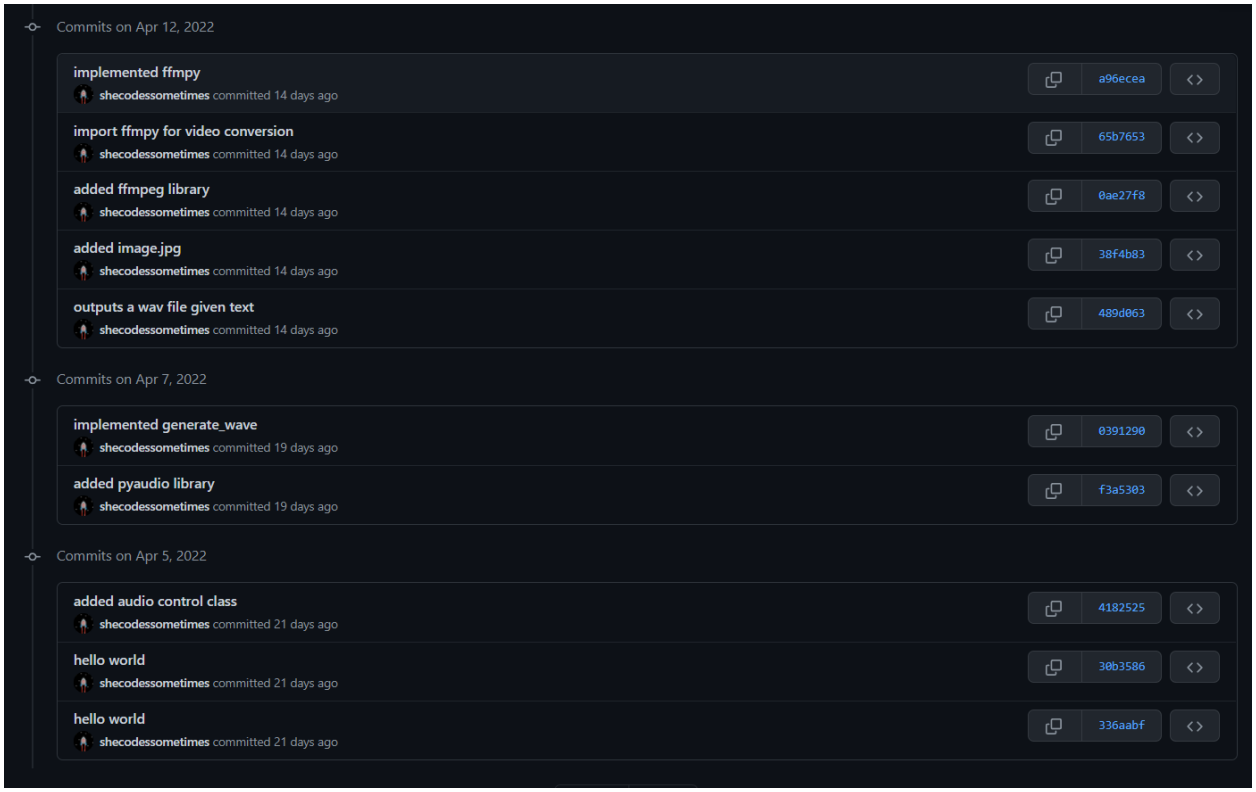


Figure 3: Commits to the repository from April 5th to April 12th.

## 5. Instructions:

In order to run our system, first download it from the Github link provided. Then, you must get a Twitter development account (requiring an application for Elevated Access) in order to run it on your own, and replace all the keys with the keys provided for your account from Twitter. Once you have done this, you can simply run the "main" file. An example of the proper implementation of the music bot can be found at https://twitter.com/2140_musicbot.

## 6. Libraries:

We ended up using six libraries for our project: Tweepy, ffmpy, NumPy, Wave, OS, and Time.

The first library we used, Tweepy (https://www.tweepy.org/), enabled us to connect directly to the Twitter API, and Tweet using the keys generated by Twitter. The next library, ffmpy

(https://pypi.org/project/ffmpy/), is the offspring of the larger library, FFmpeg, which is typically used in command line to edit and convert various types of media. The difference between the two is ffmpy is specifically adapted to Python, which is how we were able to incorporate the FFmpeg conversion tools into our project, to convert the .wav file into a .mp4 file.

We also relied on NumPy (https://numpy.org/) in order to generate a sine wave, and then used Wave (https://docs.python.org/3/library/wave.html) to convert it into a .wav file. The Time (https://docs.python.org/3/library/time.html) library was used for a delay after the video is first uploaded to Twitter, so that the uploaded .mp4 is fully processed before the Tweet that uses it is tweeted. The OS (https://docs.python.org/3/library/os.html) library was then used to delete the generated .wav and .mp4 files from the file system, so that the program could be run again, and the memory would always be cleared.

## 7. Lessons Learned:

In conducting this project, we learned a lot about different applications of Python, namely how to connect to an API and navigate the logistics of doing so. We experienced many issues trying to get the Twitter setup to function as intended, ranging from inconsistencies in the Tweepy docs to difficulties uploading media, but a combination of extensive research and trial-and-error eventually resulted in a functioning product. We also learned more about class relationships, as our project required our three classes to interact in a chain-like manner, allowing our code to be much more organized.

If we had more time on the project, we would have liked to make the bot more autonomous; Tweepy allows full automation of one's bot by streaming and releasing tweets on a schedule, so with more time we could make the bot tweet every day (or some equivalent) instead of needing someone to manually run the program to generate a new tweet each time. We could also expand the range of characters from ASCII values to Unicode, allowing a much broader range of characters/symbols to be read.

In terms of advice, we would probably advise future students to establish what libraries/methods they want to use early on, and research them accordingly to determine their capabilities and limitations. Having a set plan going into the project definitely made the process much more streamlined and efficient for us; without one, we likely would have had a lot more trouble

troubleshooting our project. Likewise, it's always a good idea to have some sort of contingency plan should an external issue arise– for example, our first Twitter bot ended up getting "limited" for supposedly violating Twitter policy (we are still unsure why); had we not had a second account with Elevated permissions whose keys we could easily implement into the code, we might not have been able to present our project as intended. Since APIs and external programs are prone to causing problems that are outside of students' control, having a backup/alternative would be better practice to ensure everything runs smoothly.