UNIVERSITY OF Waterloo

SE 101 Introduction to Methods of Software Engineering

Fall 2022

Course Project Final Report

Void of Creativity

Full Name	userID	UserName
Raghav Bhasin	20991179	rbhasin
Brandon Harris	21013863	bk2harri
Jiro Kakpovbia	21019684	jjkakpov
Grayson Rechsteiner	21002809	gjrechst
Ryan Schindler	21036368	rschindl

1. Introduction

While brainstorming ideas for our SE 101 project, we team had a strong ambition to create something positive and impactful. This is when we learned about Jake Paul. Mr. Paul is a world famous Youtube star turned boxing world champion with a powerful reach and influence. Our idea was to replicate Jake Paul's legacy and personality into a versatile home assistant device. We believed that through the implementation of various groundbreaking features, the iPauler could surpass all of the competition to become the ultimate home media device.

2. Background Research

Our first steps were to watch a few Jake Paul videos to make sure all group members were familiar with his speech patterns and personality. This was essential to ensure everyone properly understood the subject.

Next, we had to decide on what hardware we were going to use. Our group had little experience with microcontrollers, but our main two candidates were the Arduino or Raspberry Pi. After some research we realized that the Arduino would not have enough processing power in order to compute the voice input, play videos, and achieve many of the other features we had planned. This led us to selecting a Raspberry Pi 3b, which we were able to secure from Facebook.

We had so many great ideas that we wanted to implement on the iPauler. But before we were able to start programming anything, we had to research what programming language to use. After some brief research by all group members, we were quick to conclude that Python would be the best programming language to use due to the vast amount of readily available libraries that could be used to help achieve our goals.

When creating our AI, we scoured the internet for various examples of previously existing AI chatbots from which we could base our own AI off of. We came across several articles outlining the different options we had, and the trade offs associated with them. One particular article that we found to be extremely useful was a "Complete Guide to build your AI Chatbot with NLP in Python'', written by Arnab Mondal. From this article, we were able to create the groundwork for our prototype AI using several snippets of code from his example. We were then able to continue creating our personalized AI on our own.

Once we had our AI working and outputting speech, we needed to find out how to display our mouth animations and other forms of media onto the screen. After some consultation on Reddit and various other websites, we learned that the easiest way to do this would be to use VLC media player for the videos and pygame for the mouth shapes. VLC has an extremely easy to use python library which worked well on the Pi, and Pygame allowed us to quickly blit images on the screen to create the mouth animation.

Lastly, we had to research how to make the mouth lip sync in accordance with the output. This is when we found an existing Github repo including an open source software called Rhubarb lip sync. This powerful software is able to generate proper timestamps where each mouth shape is needed for a specific word or syllable based on audio input.

3. Implementation

Software:

Our project is built almost entirely in Python. The top layers focus on capturing user audio and converting it to text, running the text through various supported response workflows to find the appropriate response, and then outputting this response in audio form.

Going a little deeper, the iPauler first listens for the phrase “Jake Paul” via the Python SpeechRecognition module. Once this phrase is heard, it begins recording raw audio for a short time period. The recording is temporarily saved as a wav file. This file is put into the SpeechRecognition module to create a string into our responses module. At this point there are three main types of responses based on certain keywords.

The first and most basic category will immediately return a preset response. These responses are static and don’t require any advanced workflows. Consider if you asked Jake a simple question such as his name: The responses module would return a string of the response.

The second category are queries that initiate workflows in order to find appropriate responses, whether it be querying APIs to grep for additional info or running our trained AI models to generate unique responses. Similar to the first category, a string of the response is returned.

The third and last category consists of commands rather than queries (if the first category was a GET request, this would be a POST or PUT), and will change the behavior of the bot for the duration of the command. Special cases of the second category consist of affecting future responses by switching into various different presets, displaying any audio/video content included in our database, or playing supported multiplayer games.

If the response is of the first two types, the returned string is then run into our text-to-speech workflow. Here, the output text is first run through a TTS module which saves the response in robot-voice form as an mp3 file (then converted to a wav file). A command is then used to initiate rhubarb lip sync, a special software built on the Raspberry Pi’s home directory.

Rhubarb Lip Sync analyzes the audio and outputs a JSON file containing a dictionary of different mouth shapes to display at specific times throughout the duration of the audio clip. The Pygame audio mixer then plays the audio clip, and at the same time initializes a timer. The JSON file created by Rhubarb is used in combination with the timer and Pygame to blit different mouth shapes onto the display at their correct times. Due to long processing times, lip sync data is locally stored and persisted. Therefore this workflow is only necessary for unique or unheard responses. Finally, the program returns to the state of listening for new user input.

One example of a special response workflow includes our weather function. This function begins by taking the ip address of the iPauler and using it to return the current location and city name. This is then run through the Open Weather Api to get things like temperature, atmospheric pressure, and more.

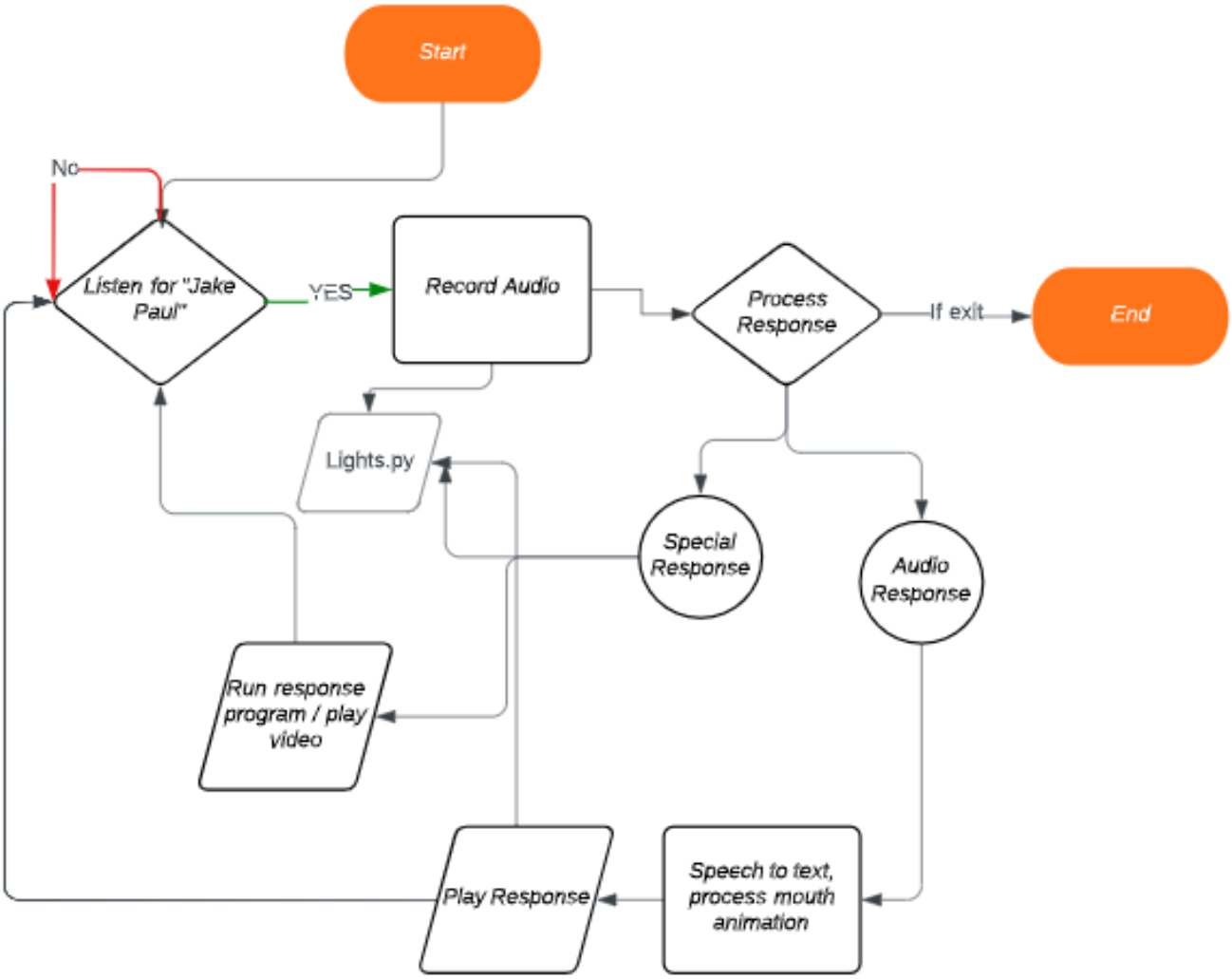
Another notable workflow includes the creation of AI generated tweets. For tweets, we use a recurrent neural network model to generate short snippets of text. To train the model, we first ran a Python script using Tweepy, a Twitter API Python SDK, to download all of Jake Paul's tweets into a csv (barring retweets, video/audio clips, links, etc). Then we inputted the csv into TextGenRNN, a Python module that utilizes Tensorflow to train text-generating neural networks, and exported the model and saved it to our Github repo. Text generation using recurrent neural networking is essentially taking a large dataset, vectorizing the text into a numerical representation, and calculating the probability of a character appearing after a given character or sequence of characters. TextGenRNN, in addition to the process above, utilizes clever strategies such as attention-weighting and skip-embedding to increase performance and decrease training time. Once the model was exported and integrated into our project, creating a tweet is as simple as inputting a prefix and letting the model do the rest.

Hardware

The heart of the iPauler is the Raspberry Pi 3b running the latest version of Raspbian OS lite. This has a small LCD screen mounted directly on top, which pulls power from the Pi's 5v pin while being attached with a small HDMI adapter.

Potentially the biggest selling feature of the iPauler are the RGB lights, which required a combination of hardware and software implementation. We decided to use common cathode LEDs which have four pins each. The longest pin is a shared ground, while the other three are for the colors red, blue, and green respectively. The ground pins were attached to pin six on the Pi (ground), while the other three were attached to their own respective GPIO port. Since the Pi would be outputting power for the LEDs, we also soldered a resistor between each of these pins (3 pins x 10 leds = 30) to prevent too much current from being drawn. All ten were then placed in parallel.

Once soldered, activating the lights was simple. Our lights module contains functions which turn the frequency of the different GPIO pins to low or high; this changes which color the LEDS omit. This doesn't limit us to just red, blue and green, as activating multiple pins at once mixes the colors. Of course, we call for different colors all throughout our code.



4. Group members' contribution

Raghav: My main role on the team was to create the lip sync AI, as well as do background research on Jake Paul. My research on Mr. Paul revolved around finding out about all the good he has brought to our world. Bringing to light all the positive things he has done in his lifetime such as his various initiatives and charity. Regarding the lip sync AI, instead of just animating the lips to move up and down as he is talking, we wanted to accurately lip sync the mouth to every word and syllable being said so that it looks better and is easier to understand what iPauler is saying. Firstly, I tried using pygame to animate a picture of the mouth in certain ways but that did not work very well with the given input from our AI. I then turned to the internet and started researching existing GitHub repos to find an existing program that would lip sync the animation using audio input, and was able to find one that worked with our other software.

Brandon: I was responsible for the twitter module. Everything from applying for the Twitter development portal, getting the tweets, and training the model. Getting the Twitter API access didn't pose many difficulties beyond a back and forth email chain with Twitter ensuring I didn't have any business purposes. Training the model proved quite hard, I mostly experimented with the tensor flow text generation tutorial for a while until I was able to train a model. Some difficulties regarding that was just getting all the required python modules working on my Mac. After the first model was trained, some testing revealed some issues with the dataset that needed refining (mainly filtering out either symbols or other tweets with natures that didn't fit with my intentions), and the text wasn't as coherent as I'd have liked. I eventually discovered a Python module that took a similar but more refined approach to text generation, but the modules were extremely outdated, so I had to migrate my efforts to Google Colab to get the model trained. Even still it required a lot of sifting through stack overflow and github issues to get it working.

Jiro: As a part of the iPauler team, I was responsible for replicating Jake Paul's voice through advanced AI learning, and implementing it into our own AI. I consistently researched different websites, softwares, and programs that would allow me to accomplish this, and I eventually found a software called Tacotron 2. This would allow me to submit clips of Jake Paul's voice along with a transcription of what he was saying to an AI, and it would then recreate his voice in a text-to-speech format. Instead of searching for numerous small clips of Jake Paul's voice from various different videos, I was able to find an hour long interview from Graham Besinger where he spoke with Jake Paul about his career and his experiences as a child. I downloaded the entire interview and split it up sentence by sentence, transcribing each sentence as I went along. I then submitted this to the Tacotron 2 software to reproduce his voice. At one point I had got it working, however, since the software was so outdated (was last updated around 2020), I unfortunately was unable to implement it into our AI as there were too many compatibility errors, outdated APIs, etc. Some other small things I helped out with were filming parts of the demo video and creating the iPauler logo.

Grayson: At the beginning of our design process I was responsible for creating the initial prototype AI, and implementing it with voice recognition software. I spent many hours researching various options for voice recognition software, eventually finding one that would best fit with other aspects of our project. Once I settled on one software, I worked on downloading all the required files and modules to be able to use it, and began creating the AI itself. I created a prototype AI program that would process voice input, and respond accordingly through the terminal with several predetermined responses to common inputs. This is when I passed off the development of the main AI to other group members, and began working on the game feature of the iPauler. I created a Tic Tac Toe program using python and pygame. I built a custom AI to play against, made to simulate playing against Jake Paul himself. I then worked to modify the program so that it could be played using user input, instead of having to type through the terminal. This required additional research, but at the end of the day this was fairly straightforward. Additionally, throughout the design process of the project I planned our weekly objectives to make sure we were on track to achieve our goals so that we wouldn't come up short of our project aspirations.

Ryan: My main role on the iPauler team was hardware wizard. I was responsible for acquiring all hardware including the Raspberry Pi, LCD Screen, frame, poster, etc and getting everything to work properly. For example, when a piece of software such as the Tweet Generator or Tic Tac Toe game was created, I pulled it on to the Pi and made sure it worked properly. Sometimes this was simple, but other times I went through hours of rage and despair trying to get things to run properly. I was also responsible for assembling the poster itself, which required soldering and stringing together the RGB LEDs, building the speakers into the frame, and doing lots of drilling and hot glue application. This also included the software implementation of the lights throughout the code. I also worked on the software side on things such as drawing the mouth images and coding the rhubarb lip sync implementation. I also implemented features such as location, weather, and video playback. During the final week of the project, I was tasked with the role of video editor for the project's demonstration video.

Signatures:

Raghav Bhasin: _____

Brandon Harris: _____

Jiro Kakpovbia: _____

Grayson Rechsteiner: _____

Ryan Schindler: _____

5. Final Product Evaluation

Our goal was to create a talking Jake Paul poster using voice recognition software and a programmed AI to respond in ways the real Jake Paul would. In general, we matched our proposal very well, and there were only really two main components of our original plan that we didn't end up implementing in our final product.

The first feature that didn't end up getting implemented was various mood settings that would dictate how the AI would respond to a given input. Secondly, the feature of having Jake Paul's actual voice outputted instead of the basic computer voice didn't end up getting implemented. The reasoning behind omitting these features is explained below in the "Design Trade-offs" section.

Other than these two components outlined above, we met and surpassed almost every goal that we originally set for our project. All other features are working fairly well, with many bonus features also included. The AI runs smoothly and responds to various input, the mouth is displayed and lip synced with the output, the AI is able to pull tweets from the internet, and you are even able to play Tic Tac Toe against our Jake Paul AI. While there are definitely things we could still improve, we believe that our final product is even more amazing than our original idea/plan.

6. Design Trade-offs

There were really only two design trade offs that had to be made during the project. First of all, in our proposal we outlined the idea of including multiple mood settings to our AI, which would alter how the AI would respond to specific input. This would be cool, but we decided that this feature wouldn't actually enhance our product as significantly as other features. We decided to scrap it fairly early on and instead focused our time towards developing more important features such as the game and lip sync feature.

The second trade off we ended up having to make was not using Jake Paul's actual voice for the AI voice, and instead had to use the basic computer voice. We worked to implement this feature up until the day before the project was due, and actually had it working, however, it was not compatible with some of the other software used for our project. So we ended up having to cut this from our final project, not out of choice, but due to limitations.

It's also worth noting that at one point in our design process we were considering implementing a chess program that one of our group members had previously created as a side project using python and pygame. However, we decided it would not translate well to the small screen attached to the Raspberry Pi. So we didn't end up sinking very much time into this idea.

7. Future Work

In the future, we hope to improve the iPauler AI, making it more realistic and adding new features to keep the user experience fresh and interesting. As for adding more realism to the AI, we could do this by replacing the generic, robotic voice that the AI currently uses with a text-to-speech version of Jake Paul's voice. This would be accomplished using advanced AI learning. We'd have to gather multiple clips of Jake Paul's voice and transcribe each clip. Then, we would submit the clips and the transcription to an AI, allowing it to learn how Jake Paul speaks, his pronunciation, his mannerisms, etc. Thus, the AI would be able to recreate his voice saying whatever text we give it. Additionally, we could find a way to inflict different tones of speech into this AI, allowing it to display different emotions to an extent. This would help bring the iPauler to life as it'd be that much closer to replicating the man himself.

Another feature we could add to improve this AI would be to generally make it work a lot quicker and more efficiently. This would include cutting down the time it takes for the AI to read and respond to input, and making it battery-powered instead of relying on a power outlet. Currently, our AI takes a longer time to process the user's input as it needs to recognize what they're saying, convert that to text, look for specific keywords in the text, respond accordingly through text, and then respond verbally. This delay is especially noticeable when the AI is responding to a new user input for the first time. After the first instance, it is much quicker at responding, however, this quickness should be consistent with every response. Now regarding the AI's main source of power, it currently needs to be plugged into a wall outlet to work correctly. In the future, it'd be ideal to make the AI work with simple AA or AAA batteries – this would make the iPauler much more portable and convenient as it could be used in any situation and at any time.

Lastly, a final improvement that could be made to the iPauler AI is making the speech recognition much more advanced. Currently, the iPauler AI only listens for specific keywords, and then responds accordingly. However, in the future, our voice recognition software would have the ability to detect longer phrases, along with differing tones, volumes, accents, etc. It would then tailor its response depending on all these different factors, creating a unique experience for each user.

8. References

Amos, D. (2022, April 14). *The Ultimate Guide to Speech Recognition with Python*. Real Python. Retrieved from https://realpython.com/python-speech-recognition/

Deangelo, G. (2022, March 30). *How to make a Pygame quiz with speech recognition*. Python Programming. Retrieved from https://pythonprogramming.altervista.org/little-pygame-game-with-speech-recognition/

Karpathy, A., & Grijalva, D. (n.d.). *Easily train your own text-generating neural network of any size and complexity on any text dataset with a few lines of code.* GitHub. Retrieved from https://github.com/minimaxir/textgenrnn

Krishna, A. (2022, March 3). *How to Get Location Information of an IP Address Using Python*. freeCodeCamp. Retrieved from https://www.freecodecamp.org/news/how-to-get-location-information-of-ip-address-using-python/

Mondal, A. (2021, November 9). *AI chatbot: Complete Guide to Build your AI chatbot with NLP in python*. Analytics Vidhya. Retrieved from https://www.analyticsvidhya.com/blog/2021/10/complete-guide-to-build-your-ai-chatbot-with-nlp-in-python/

Raspberry Pi tutorial: How to Use a RGB Led. Autodesk Instructables. (2018, September 23). Retrieved from https://www.instructables.com/Raspberry-Pi-Tutorial-How-to-Use-a-RGB-LED/

Ravikiran, A. (2022, September 27). *Speech recognition in Python - A Complete Beginner's Guide*. Simplilearn. Retrieved from https://www.simplilearn.com/tutorials/python-tutorial/speech-recognition-in-python

Saxena, S. (2021, September 17). *5 Times Jake Paul Proved that He's a Good Human Being*. Sports news. Retrieved from https://www.sportskeeda.com/mma/5-times-jake-paul-proved-good-human

Weather API. OpenWeatherMap. (n.d.). Retrieved from https://openweathermap.org/api

Wolf, D. (n.d.). *Rhubarb Lip Sync*. GitHub. Retrieved from https://github.com/DanielSWolf/rhubarb-lip-sync/

Yanofsky, D. (n.d.). *A script to download all of a user's tweets into a CSV*. GitHub. Retrieved from https://gist.github.com/yanofsky/5436496