Final Programming Assignment Report - Jacob Walton

Requirement

The project had a handful of requirements that needed to be met to fulfill all obligations needed of the chatbot. The chatbot was intended to be able to give information in regards to the 10-K reports of two different companies. The chatbot was required to be able to be interacted with by a user by entering a sentence, to which the chatbot would then be able to read the sentence, determine what the user was attempting to ask the chatbot, and then give an appropriate response. The information must have been the parts and items of the 10-K reports of the two companies, and needed to be extracted from an external source. This data must then be processed and extracted based on questions asked of the user in a command-line interface. The chatbot must also be able to handle any user query and report the statistics on the interactions of a session, across the use of multiple sessions. Any further specifics were up to the discretion of the programmer, as long as the limitations and scope are properly documented. Said proper documentation can be found in the following section. There were some items and features that were not up to scope and the chatbot must be able to handle if prompted by the user. This includes the ability to be able to ask multiple questions without the program ending, being able to end the program if prompted, to give all information regarding a company, to be able to communicate what companies are supported by the chatbot, to handle chit-chat as well as unknown queries, and the usage stats of the program. The usage required stats include a summary of all uses of the chatbot, with the total amount of utterances for both the user and the system, as well as the total duration of all chats. The program must also be able to give a report of

individual chats, as well as being able to display the visual chats. There must also be a proper system in place for not only ensuring there are no errors that could abruptly end the program, but also be able to notify the user if what they entered is invalid (such as asking for chat 200 when there are only 198 of them). As mentioned earlier, these were the general guidelines, with the rest of the process of how the chatbot would operate being allowed to be determined by the programmer themselves.

Specification

For my project I knew pretty early on that I wanted to use Java. Though I had initially wanted to use C++ in an attempt to better familiarize myself with the program, that did not last long. I realized that not only due to the fact that I had much more experience with Java, but that I also had code already made from prior projects and classes that could be reused and repurposed for the chatbot. This decision proved to be the correct one as I did end up going back to prior code and reshaping it to fit this new program. The two companies I chose were PepsiCo and The Coca-Cola Company. The two are quite similar companies and I thought a chatbot that analyzes the data from two directly competing companies could provide interesting results. When it came to deciding the scope of my project, I am going to be honest and say I did not want to try and get out of my comfort zone. I am still relatively new to coding, and knew that some of the existing requirements would be a challenge for me, so I tried not to stray too far from what was required of me. I decided that for the scope I would make it so that the user would be able to enter a sentence into the command line and the system would look for keywords that correlated with each of the possible responses. First, the program creates a string file that will house the project's "showchat" (this is updated

after every utterance, both user and system) as well as a matrix that will house the contents of the csv file that will be reached to get a summary of the chatbot. Then, the chatbot determines what company is in the prompt by using a checker function; the checker function takes two inputs and sees if input B can be found within input A, and then outputs a true or false response depending on if it does. The system would match the user's input with each and every possibility to see what keywords were included (there were no overlaps in what could be said and deemed a keyword). The queries supported were an option to view everything a company has, the option to view the contents of each 10-K file, as well as the ability to see the contents of each item for either 10-K file. The rest of the queries possible were ones directly stated in the prior section as being required (such as determining the chat statistics through “showchat” and “summary”, or being able to quit the program). The system can still access those other queries if no company is given, and if a request query is prompted with no company, the system is able to use the company last given. Essentially, once a company is mentioned, the program will use that company until stated otherwise. Most outputs are given in a command line, except for any output that is giving contents of a 10-K file, as they are quite large. To display any part of the 10-K file, the system takes the starting and ending line of a specific item. This is obtained when the system matches the user input to a specific query, each query assigns two variables as the starting and ending line, which are pulled from an editable array so that the program can be used with other companies. The system then loops through the 10-K file of the company chosen, begins to print to an output file when it reaches a line that is the same as the starting line variable and stops when it reaches a line that is the same as the

ending line variable. The program will loop until the user decides to quit. After the user quits, the system reads from the csv files, populates the matrix accordingly, before updating it with the new data for the current system usage. Before Going with just the expected requirements provided a slightly more straightforward program for me to be able to spend the time to learn and understand these new coding techniques instead of trying to figure out how to use code that was beyond my expertise. I think it would be interesting to see how I would do this project over again now that I have learned those coding techniques.

Development Highlights

My code was broken up mainly by various different functions that would call each other back and forth, but I did have two different classes that I used. There was the main class of program6 and then UserUtterance. UserUtterance would be called every time the user gave an input, if the user's input matched one the many regular queries, it would give an appropriate starting and ending line through the array stored in the class, if not, it would give one of the many 'special functions', which basically acted as an else-if statement that transferred over between classes. Testing the program was actually quite annoying if I am being honest. As a lot of times the report given for the error would be either incredibly vague or not even present until after I hand tested my code. My method of testing was simply running it in my IDE (and then subsequently console as a last check) and seeing where it displayed any errors. While there were a handful of problems encountered throughout the process of making the chatbot, I will share some of the key ones that took the most time to fix. When first setting up UserUtterances I realized that the section numbers were not lining up right.

Unfortunately fixing it was not a quick process, and required me to go and manually alter every single instance in the else-if function. While it was tedious, this made any alterations to the code in the future a lot easier to accomplish. I also ran into trouble when I tried to alter my code so that it ran by using word matching percentage instead of key words. The problem I encountered was that the matches would be so low due to the fact the amount of words in a sentence is high compared to the one or two key words or phrases that could identify what the specific query is. That thought process right there is what led me back to working with my keyword method, as in the way I was building the chatbot, it really only needed to identify the core words or phrases tied with each query. While it was a shame to leave behind all the work I had put into trying to get the matching percentage method to work, I wanted to avoid getting caught in the sunk-cost fallacy, continuously trying and failing to make a method work. Perhaps if this was a project with a larger timeline to work on it (as I had only about a week to work on this part of the code), I may have been able to brute force my way into getting that method to work. However, given the time frame I had available, I think shifting gears into a strategy I was more comfortable with was ultimately the right decision. The last big problem I encountered was working with a csv file. I have never really worked with a csv file before, so it was a struggle to try and figure out how it works. I initially believed it would be just like working with a text file, which I soon found out to be an incredibly naive assumption to make. After a bit of trial and error, I had to turn to the internet to figure out how exactly to get it to work. That was not a simple process either, requiring a lot of trial and error as well with finding code, testing it to see if it worked, getting a program breaking error, and having to go back to step one of the search. After a while I

did eventually find code that worked and was able to make the needed altercations to it to ensure that it worked properly with my own code. In the end, the need for this entire process was made null and void as it would actually reuse code from another student for this part of my software, which I will discuss in the next section.

Reuse

I actually tried my best to keep my code modular, not only so that other people would be able to pick it up, but so that I could easily remove parts if need be so that I could reuse other people's code. I tried my best to split the code up into various functions that would be called from the main functions. I especially attempted to ensure the code was easily reusable in my UserUtterance class. The entire reason it is its own class was so that it could be easily reusable and editable by others if they so desire. One of the key ways I did this was with the two section arrays. Since each query returns two strings, the starting line and ending line of a section of information, I did not want to just manually type out what the lines were every single time it needed to be called, especially since some lines needed to be called multiple times. So instead I created the arrays so that the exact lines from any new 10-K report could be easily slotted in with no issue. Since most 10-K reports have the same general format, differing in minor semantics, this would allow the entire section to be reused and ported to a new program with minimal effort. But this modular style of coding also helped me when I reused code from fellow students. The first bit of code I reused was from Jarett, whomst code for creating a file log of a system's usage was an improvement over mine. I originally had an array that was 999,999,999 indices long, then populated each index with a string each time an utterance occurred. Jarret had his code set up as a string that would be

added on to with character returns. While this may not seem like a big deal, this minimized the amount of constant memory that my system allotted every time it ran, no longer having an array just shy of one billion indices long. I altered the code for this section accordingly to make it better connect with my program, and it noticeably shortened the run time of my program. The other code I reused was from Cody, in which I repurposed their code that printed to a csv file. The code I had previously been using had been poorly put together from a primitive understanding I had of printing to a csv file from various forums and YouTube videos. So, being able to take code directly from someone who knows how this worked made the end result a much smoother and simpler process. In the end, these two code reuses may not seem like a lot, but they were very helpful in smoothing out some of the quite rough portions of my code. They may not have been entire sections, but they were able to pull up the slack where my own code was lacking. The two even reused some of my code that they found helpful. Jarett reused the code I had that checked to see what company the user had included within their prompt, and Cody reused my code that gave some of the statistics of the program. While I think both of those sections were well made, I was especially proud of the portion that was able to determine a prompt's company without having a separate prompt needed just for it. The biggest challenge in all of this was connecting the new code to the existing code. While I made my program modular, there were still difficulties in merging different styles of code. It felt like I had to connect two different electrical devices with wires, there's a clear difference in the two sections, with code bridging the two together at various points. Overall, however, I am very pleased and proud of how not only I reused code, but how my code was reused.

Future Work

While I am extremely happy with how my chatbot turned out, there are still plenty of more things that could be done with it. I think the biggest thing would be to update the keywords system into becoming a percentage based check system. It was something I wanted to do, but was unable to due to limitations on time and my own capabilities. I strongly believe that if I were able to do this project again in the future, I would be able to get that idea to work. I would also like the ability to get more specific responses to questions. Most of the responses give a large quantity of information to cover all bases, so to speak. However, I would like to be able to minimize the amount of information the user is presented to just the specifics of what they are looking for. The main other part I would like to add would be more variety to the chit-chat applications of the chatbot. Unfortunately, the chit-chat portion is lumped in with the incorrect prompt portion of the code, as I could not get the code to differentiate between the two. And while this is not a problem as the ability to differentiate the two was neither a requirement nor part of my scope, it is something I would like to add to breathe more life into my chatbot. In regards to how the code will need to be changed over time, apart from what I said, I think a lot can be done in condensing the code. There are a few times in which I reuse the same function, only with slight modifications, for different parts of the code. I am sure that with more time, this could all be one single function with different parts that are used depending on what part of the system is accessing the function. I also think that some of the other parts of my code dealing with the csv file could be improved. As mentioned multiple times, my understanding of working with a csv file is very minimal at the moment, and while I was able to reuse code for part of my project, there are other parts

of my system that rely on code that is a little clunky. In the end, I am quite happy with the end result of this chatbot. While there are parts that can be improved upon with time, for my current knowledge and understanding of programming and the techniques involved with it, I am proud of the product I have created.