PLD 450 – Lesson 1 – Applied IT Project: Morse Code App

Victoria Langlais

April 22, 2022

Northern Arizona University

PLD 450 – Lesson 1 – Applied IT Project: Morse Code App

The project I made for the Applied IT Project is a Morse Code translator app. The app will allow users to learn and work with Morse Code. They can use the app to enter text, and have it translated to Morse Code or enter Morse Code and have it translated to text. They can improve their Morse Code skills by reviewing the Morse Code reference chart and taking quizzes that test their recognition of Morse Code.

Project Scope/Deliverables

For the Applied IT Project I designed a functioning Morse Code app. There are a few deliverables for my project. First, there needs to be a functioning app that allows the user to enter text or Morse Code and have it translated. There needs to include six quizzes to help the user improve their understanding of Morse Code. A second deliverable is the final code that makes up the app's various features (translators, reference charts, and quizzes). A third deliverable is documentation of my development process, as saved in my development journal. Lastly, there needs to be screenshots depicting the app running on a real device, to prove that it is functional. All these deliverables need to be uploaded to GitHub so that they can be easily accessed and viewed.

Cost/Budget

For my Morse Code app project there is no cost or budget. Since I am the one creating the app, there is no one who needs to be paid. The software I am using to develop my app (JetBrains' IntelliJ and Android Studio) are both free programs. The only thing that this project cost was time. If I were making this app for a real client, a budget would be required; however, since I am the client and the app has no real costs, a budget is not needed.

Planning

To begin the development process, normally I would meet with the key stakeholders (clients, investors, developers, etc.); however, since I fill each position, a meeting was not necessary. Instead, I worked on brainstorming ideas for the app and wrote out all the features and requirements I wanted for it. The app needs to allow the user to input either text or Morse Code and have it translated to the reverse. For instance, if text were entered, the output would be in Morse Code and if Morse Code were inputted, the output would be in text. There needs to be a way for the user to switch between the two translator methods. A cheat sheet also needs to be available in case they need help remembering the Morse Code for certain letters, numbers, or punctuation. I also sketched out potential interface formatting ideas.

The main purpose of my Morse Code app is to serve as a translator. The translator needs to have a place where the user can input text. For the text to code translation method, when the user clicks on the input text field, the phone's keyboard should appear. The user can input text, which will be displayed in real-time in in the output text field. For the code to text translation method to work, there needs to be buttons that can be clicked to enter either dots, dashes, delete characters, and enter a space or indicate a new word. As the buttons are clicked, the corresponding Morse Code characters will appear in the input textbox while the translation into text will appear in real-time in the output text box. Having the text appear in real-time will help the user determine if they have made any errors in their text. Rather than entering the text, realizing the error, and needing to rewrite the entire message to make the correction, they can easily see their mistake and fix it immediately. There needs to be a switch that will allow the user to jump between the text to code translator and the code to text translator. On each translator

method, there also needs to be a cheat sheet button that, when clicked, will make a cheat sheet pop up. The cheat sheet will display a reference chart of Morse Code translations.

There needs to be a section that helps the user learn Morse Code. This needs to be a chart that has each letter, number, and punctuation and their Morse Code translations. This reference chart can be used as the cheat sheet for the translator section. To make sure the text is big enough for users to read, a scroll view can be used. This will allow the user to move the list up and down, letting them view the entire list of Morse Code.

The app also needs to have a quizzes section, which will allow the user to improve their Morse Code recognition skills. There needs to be 6 quizzes: text to code letters, text to code numbers, text to code punctuation, code to text letters, code to text numbers, and code to text punctuation. The two letter quizzes will have 26 questions each, while the numbers quizzes will have 5 questions each, and the punctuation quizzes will have 10 questions each. The quizzes will be formatted with a question and four option buttons. The questions will be posed as “What letter/number/punctuation is this” and the corresponding character will be displayed, either in text or morse code according to the quiz. The four buttons will provide the user with four different options. If the correct answer is selected, the button will turn green, and if the wrong answer is selected, the button will turn red. After an answer has been selected, the Next button will appear to take the user to the next question. At the end of the quiz a score sheet will appear, informing the user of how many questions they got correct. A button will appear on the score sheet, prompting the user to take another quiz. The Take Another Quiz button will take the user to the list of quizzes so they can select which one to take next.

There needs to be a home page for the app to display navigation buttons, which, when clicked, will take the user to the various content pages. There needs to be three buttons, the

Translator button to take the user to the translators, the Learn Morse Code button to take them to the Morse Code reference chart, and the Quizzes button to take them to the list of quizzes. A back button on each page of the app will redirect the user to the home page, except on the individual quizzes. The back button on the individual quizzes will take the user back to the list of quizzes.

Timeline/Schedule

After writing out the project requirements for my mobile app, I developed a timeline to help keep me on schedule. From March 22nd until March 26th, I will conduct research for my app by studying other morse code apps. Doing this research will help me see what features, formats, and colors are used in other apps. I can take this research and apply it to my app to improve upon my initial design. After doing research, I can start developing the app. From March 27th until April 2nd, I will develop the code for the text to code and code to text translators. I will also begin designing the interface for the app. From March 3rd to April 9th, I will finish the interface design and combine it with the translator code. I will also develop the code for the quizzes and reference chart. From April 10th to April 16th, I will combine the completed code for the quizzes and reference chart with the app interfaces. Once the development of the app is completed, the testing phase of my app development will begin. I will test the app, looking for bugs in the programming and any formatting issues that need to be fixed. When the testing and corrections are done, the app will be completed and exported. The app needs to be finished by April 24th to be submitted along with the final report.

Design

Before designing my Morse Code translator app, I started by researching other Morse Code apps. I examined several apps to see what colors, formats, and features they used. There

were five apps that I looked at: Morse Cody by predefault, Morse Chat: Talk in Morse Code by Dong Digital, Morse Mania: Learn Morse Code: by Dong Digital, Morse Code-Learn and Translate by Pavel Haleček, and Morse Code Flashlight by AC Form. I found that most of these apps had an input text field, used the phone keyboard, and an output text field where the text appeared in real-time. They also had features that allowed the user to switch between entering text and Morse Code. They used a variety of color schemes, such as green/white, blue/white/yellow, blue/white/grey, and dark blue/yellow. I found it interesting that the apps had additional features that allowed the user to physically communicate with Morse Code. The apps connected with the phone's flashlight and audio functions allowing it to blink the flashlight, make beeping noises, or vibrate the phone to transmit the messages.

I found several features from these apps that I could incorporate into my own design. I liked the format of Pavel Haleček's Morse Code-Learn and Translate and predefault's Morse Code because they were simple. They had two text fields (one for input, one for output) and it was easy to switch between the two by clicking a button. They both had easily accessible reference lists. I liked the reference charts on those apps (the had separate pages where the list was found) and on AC Form's Morse Code Flashlight I liked the reference chart because it was a pull-out panel. The pull-out panel was similar to what I want to do with the cheat sheet on my app. I also liked the sound, vibrate, and flashlight features these apps had. I would like to include them in my design; however, I decided against adding them because it was causing project scope creep, and I was not sure if I would have the time to add those features to my design. On a future version of the app, I may decide to include them, but for now the app will work without those additional features.

Development

I decided to begin developing my app by creating the translator code in command line format. I figured doing this would allow me to make sure that I can get user input and have it translated to the correct output (either Morse Code or text). I chose to write my code in Java since that is the programming language with which I am most familiar. The program I used to develop my Morse Code translator programs in was JetBrains' IntelliJ IDEA. I used this program on previous Java programs, therefore I was familiar with it. The code needs to take user input by using the scanner function, translate it, and print out the output. I was going to use an if/else statement to translate the code. The if/else statement would check the input and for each letters it would display the assigned output text/code. I ended up going with a switch case statement instead because it was easier to work with when retrieving individual characters. I used the format for the switch case that I used on previous Java projects. The switch case works the same way as the if/else statement, by taking user input and checking it to see if it matches a specific case, and then displaying the correct output.

I created two translator programs, one that is text to code and the other is code to text. I had some issues with the text to code translator because it kept giving me an error for trying to use strings when chars are needed. I solved this issue by converting the string input to chars, and then outputting them. I also used IntelliJ to create a quiz, testing out how I could do it for the quizzes on my Morse Code app. I used the Geeks for Geeks video tutorial, "How to Make a Quiz Application Using Java?" to create a quiz program. Following along with the video, but using my own questions and answers, I was able to create a quiz that presents the user with a question and four choices. The user then enters their answer, and the program will tell them if the answer they entered is correct or incorrect. The next question then appears. At the end, the quiz informs

the user of how many questions they got correct. The video tutorial helped me figure out how to use an array to store questions and answers. I was able to take the knowledge I learned from creating and testing my programs in IntelliJ and transfer them to the actual code for my Morse Code app.

After creating the translators and quiz program in IntelliJ, I moved on to creating the real code for my app. I decided to create my Morse Code Translator App using Android Studio. This program is designed to help users create apps for android devices. I have experience working with Android Studio when I created mobile apps for previous assignments. To begin the development of my app, I decided to start with the home page. The home page is where the app will load to and where the back buttons on other pages will redirect the user to. In the activity_main.xml file, I created the interface design for the home page. I created three buttons on the page, Morse Code Translator, Learn Morse Code, and Quizzes. I adjusted the width, height, and margins until I had the buttons formatted in a way that I liked.

The color scheme I decided on using is a grey/white/green scheme. At first, I was going to use green buttons, but it did not look exactly right. I ended up just using the default button color because it complimented the shade of dark grey I chose for the background. I found the color code, #3B444B, on the website HTML Color Codes. I used the website since it provided me with a variety of colors and the corresponding codes. It was easier to scroll through the website to find color shades that I liked, rather than using Android Studio's color picker feature. The color picker would have required me to click through colors and adjust the RGB values until I found a shade that would work, which could have taken a long time.

Once the home page interface was completed, I moved on to writing the code for it. In the MainActivity.java file, I created the code that controls the buttons. I had to assign each

button a content view, connecting it with the corresponding button in the XML file. Then I added the onclick function, telling each button to follow a specific function, which would open an activity. To create the opening functions, I used the intent operation to tell the button which activity file to open. For example, when the Translator button was clicked, it would run the "open translator" function. This function would move the user from the home page to the Translator page. The Learn and Quizzes buttons would do the same thing, just redirect the user to the corresponding pages.

The next thing to do on my app was create the Translator pages. I needed to create two translator pages, one to convert inputted text and output it as Morse Code, and the other to convert inputted Morse Code and output it as text. Once again, I started by creating the XML files for the translators, since I needed to create the buttons and text fields before I could write the code controlling each one.

I started with the Text to Code translator, creating the Translator_T2C.java and activity_translator_t2c.xml files. Before programming it to translate code, first I needed to get the text fields to accept text. On the XML file, I changed the input TextView to EditText. This would allow the user to click the text box and the phone's keyboard would appear, letting them enter their text. Next was adding the function that would take the inputted text and translate it into Morse Code. I tried using the translator code I wrote in IntelliJ; however, I ran into some issues. The emulator would not run the translator program due to it using ". nextLine" with the scanner. I had the same issue when developing the code to text translator. I kept having issues getting the program to read and translate each character that the user inputted, rather than just the first one. To solve this issue, I changed the program so that instead of using a switch case, it would use an array. I did some research and used the Geeks for Geeks article, "How to Create a

Morse Code Converter Android App?" as a basis for my translators. I liked how it used arrays to store the text and Morse Code characters. It was also useful in showing how to use a for loop to get the next character. The loop takes each character, translate it, and displays the translated text in the output TextView.

Next, I added the code that would allow the translated text to appear in real-time. Using the Stack Overflow post "How to update TextView with EditText real-time using java" I was able to emulate the code for making text appear in real-time. I combined the real-time code with the code to take the inputted text and output the translation. This involved converting the inputted text characters to a string and using public void functions/TODO methods to generate the translated text and make it appear in the output TextView instantly.

Once the Text to Code Translator was done, I moved onto the Code to Text Translator. I created a new activity, which created the Translator_C2T.java and activity_translator_c2t.xml files. In the XML file, I copied the layout I had for the input and output field from the Text to Code Translator. The input field I changed to a normal TextView; if it were and EditText field, it would make the keyboard appear. To get user input, I created several buttons. There are a dot and dash button that will allow the user to enter the dot and dash Morse Code sequences. I also created a button that allows the user to create a space indicating a new letter. For new words, there is a button that creates a backslash, to separate the Morse code into individual words. A back space button was also created, allowing the user to delete characters in case they made a mistake. In the Java file, I copied the code for the Text to Code Translator, altering it when needed to accommodate the different input method.

I created a switch button that would allow the user to navigate between the Text to Code Translator and Code to Text Translator. In the drawable folder I created two files, toggle.xml and

track.xml to change the color of the switch's toggle button and background track. These XML files were connected to the switch's design in the translator's XML files. I positioned it at the top of each translator and labeled each side as either text or code. On each translator's Java file, I added the code that would control the switch. Using the intent operation, I programmed the switch to open the Code to Text Translator when activated and to return to the Text to Code translator when deactivated.

With the translators completed, I started working on the Morse Code reference chart. I created a new activity, making the LearnCode.java and activity_learn.xml files. In the XML file, I used a table layout to create a table. The table has four columns, two to text and two for code. In the first column, each row contains a text letter, with its Morse Code counterpart in the second column. In the third column, each row has a number or a punctuation with the corresponding Morse Code in the fourth column. I added a ScrollView to the table, which allowed me to make the text font bigger. The ScrollView allows the user to move the table up and down to view the entire table. I decided to make the background of the reference chart an off-white color, #fdf5e6, since it complemented the dark grey background. I also added some green accents to the table, to add some contrast. Going back to the home screen, I added the Learn.java file to the Learn Morse Code button's intent operation, so that when clicked, the user is redirected to the Morse Code reference chart.

Using the Morse Code reference chart, I was able to create the translator's cheat sheet. I created a new activity, pop.java and popup_window.xml, where I copied the reference chart to the XML file. I removed the ScrollView and made the font slightly smaller, so that the chart would appear in full on the screen, without the need of scrolling. I also added a hide button, so that when selected it will make the cheat sheet disappear. In the Java file, I connected the layout

to the pop-up reference chart and programmed the hide button. I also set the dimensions for the pop-up window; so, it would not take up the entire screen. I had to go into the Styles.xml file to customize the pop-up window's appearance. I made it so that when the cheat sheet appears, the translator can still be seen in the background. I then went back to the translators and created a Cheat Sheet button. In their Java files, I added the intent operation for the buttons programming them so that when clicked, the pop-up cheat sheet appears.

To make sure that the translators would work, I ran them in the Android Studio virtual device emulator. This allowed me to see how my app and the translators would appear and work on a real device. I ran the translators on two devices: a Pixel 3 XL API 23 and a Pixel XL API 23. Using the emulator, I was able to check if the formatting of my app would work and if the buttons worked. All the buttons worked, allowing me to interact with the app and input text/Morse Code in the two translators. I was able to debug the program, fixing any errors that were preventing the code from running properly. One issue I ran into was that the code would not run at first. The app would crash whenever I clicked the Translator button. After some troubleshooting, I realized that the two translator activities were not listed in the app's Android Manifest. Since the activities were not listed there, the emulator could not access/run them. By adding the activities to the Manifest, the app started running properly in the emulators. I used this experience to make sure that all the other activities were listed in the Manifest to prevent the error from occurring again.

After completing the Morse Code reference chart and adding it to the Translators, I was able to move on to the quizzes. First, I created a new activity, where in the activity_quizzes.xml file I created six buttons. I assigned text to each button to identify the six different quizzes: text to code letters, text to code numbers, text to code punctuation, code to text letters, code to text

numbers, and code to text punctuation. I used the default button color to match the buttons used throughout the rest of the app. In the Java file, Quizzes.java, I assigned the intent operation for each button, programming them to open their corresponding quizzes (Quiz#.java).

Next, I started working on the quizzes. I had trouble adapting the quiz code I had made in IntelliJ, so I had to create something new. I followed along with the video tutorial, "Android Studio: Create a Multiple-Choice Quiz" to create the quiz. I was able to create the buttons on my own, but the video helped me figure out how to create arrays and store my questions, choices, and answers in them. Each quiz is made up of three files, the activity_quiz#.xml, quiz#.java, and quiz#QuestionLibrary.java. The quiz#QuestionLibray.java files are where the arrays are stored, as well as the code that tells the app to retrieve the questions, choices, and answers from the arrays.

Each activity_quiz#.xml file has the same layout. Each quiz has a TextView that displays the questions. The question either asks the user to identify the correct Morse Code translation of a specific letter/number/ punctuation or asks them to identify the correct text translation of a specific letter/number/punctuation Morse Code sequence. The question TextView has the same off-white background used on the reference chart. Below the question TextView there are four buttons. The text on each button will change when the choices are retrieved from the choices array in the quiz#QuestionLibrary.java. The choice buttons use the default color button, used on other buttons in the app. I created a Next button, positioned below the choices buttons. I set its visibility to invisible, so that when each question appears, the Next button is invisible. Once a choice button has been selected, it will be made visible and allow the user to proceed to the next question. The Next button is the same off-white color of the TextView to contrast with the background and stand out from the choice buttons.

After I created the layout for each of the six quizzes, I moved on to the Java files. These files control the quizzes, telling the app how to run them. First, I declared the various TextViews and buttons in the quiz#.java file and assigned them their layouts from the activity_quiz#.xml file. Then, I created the code to control what happens when each choice button is selected. Each button, if selected, checks its assigned text (retrieved from the choices array) to see if it matches the answer (retrieved from the correct answers array). If the choice matches the answer, the quiz score is increased by one and the button turns green. If the choice and answer do not match, the quiz score is not updated, and the button turns red. After a button is selected, the choice buttons are locked to prevent the user from entering another answer and the Next button becomes visible. For the Next button, I programmed it so that when clicked it runs the update function, unlocks the choices buttons, and makes the Next button invisible again.

The update function will retrieve the next question, choices, and answer from the quiz#QuesitonLibrary.java and set the question and choice button text to reflect the new question and answer. The background color of the choice buttons will be reset to the default button color. The number of attempted questions is updated. Updating the number of attempted questions will help the app determine when the quiz has reached its end. When the quiz has reached the specified number of questions attempted (26, 8, or 10) the quiz will end. When the quiz ends, the score card appears.

For the score card, I created a bottom score card so that it will appear at the bottom of the screen. I based my score card on the one used in the “How to Make a Quiz App in Android?” Geeks for Geeks video tutorial. I created a new XML file called score_bottom_card.xml. In this file, I created the layout of the score card. The background of the score card is the off-white color I chose to use. In green text, the message “Your Score is #/#” will appear. Beneath the score is a

Take Another Quiz button. In the quiz#.java files I programmed the button so that when clicked it will redirect the user to the list of questions, where they can select another quiz to take.

After I created the code for the quizzes, I ran them in the virtual device emulators. This allowed me to see how the quizzes would work in a real device. I was able to discover a few issues. One issue I had was that when I tried to get the questions to shuffle, so that each time the quiz is taken they are in a random order, it would not work. For some reason it would randomly shuffle the questions, but make some questions appear multiple times, while omitting others. It would also shuffle the choices, so that at times, the correct answer was not given as an option. I was able to fix this issue by programming the questions, choices, and answers in a random order in the arrays. While the questions will not appear in a random order every time the quiz is taken, it will seem to be in a random order.

Another issue I had was that the score card would appear at the wrong time. Rather than appear after the last question has been answered, the score card would when the question first appeared, preventing the user from answering it. I tried fixing this issue, but it made it so that the quizzes would not run. Instead, I got an error claiming that the quizzes were out of bounds. After some research, I figured out I needed to adjust the arrays. I was able to solve both problems by adding an entry into each array that was a duplicate of the last question. The duplicate entry made the arrays be in bounds; plus, it made it so that the user can answer the last question and have the score card appear at the right time. There was probably another way to fix the issues, but that was the one that worked first. If I continued trying to solve the issues, I may have found another solution.

Once all the codes for the translators, reference chart, and quizzes were completed, I was able to put it altogether. I continued to run the app in the emulators, looking for errors. I ran the

Android Studio Debugger, to detect any bugs in the program. No issues were found. I went through all the code files, reorganizing and editing the code. I removed any lines of code that were no longer active. I also reordered some of the code on the XML files to make sure that the properties appeared in the same order. As I cleaned up the code, I went through and added comments explaining what each part of the code does.

Implementation/Testing

The next phase of my development process was implementation. I needed to implement my project to make sure that it would work. Therefore, when the app was done, I decided I needed to test it on a real device. Using Chapter 10: Deploying Programs in Mike McGraths' *Java in Easy Steps*, I was able to connect my phone, a Samsung Galaxy S8, to Android Studio. Using a USB connection, I was able to download my app to my phone. Running it on my phone allowed me to see how it ran on a real device. I reviewed the app, testing it to make sure everything was running properly. I took all the quizzes to make sure that the questions and answers were easy to read and that the correct answer was given as a button option. Testing the app on my own device helped me identify some issues. First, the format layout was distorted. I realized that when I developed the code, I did not design it with the Action Bar (the navigation bar at the top of the app) in mind. I did not notice the issue when the app ran in the emulators. A second issue was that in the Code to Text Translator, the back space button was cut off. I had programmed the dot, dash, new letter, new word, and back space buttons to appear in a row. On my phone, the screen width was different than those of the Pixel 3XL and Pixel XL, thus making the button appear on the emulators but disappear on my device. A third issue I found was that on the two punctuation quizzes, the buttons were shifted down, cutting off the Next button.

I went back into Android Studio to fix the code. To fix the layout problem, I added an Action Bar function to each Java file. I was able to get the Action Bar to appear and set the title text. On each activity, the title text would be different, identifying which activity was currently running. For example, on the Text to Code Translator, the title would identify it as the Text to Code Translator. I also assigned the Action Bar with a back button. On most of the pages, the back button would redirect the user to the home screen. On the individual quizzes, the back button would redirect the user to the list of quizzes. I decided on making the Action Bar green to match the green accents I used on the reference chart.

To fix the Code to Text Translator button issues, all I had to do was reformat the buttons. Rather than having them appear in one row, I broke them up into two rows. The row consisted of two buttons, the dot button and dash button. The second row consisted of the new letter button, new word button, and back space button. I had to adjust the positioning of the buttons, but I finally got them lined up correctly.

As for the formatting issues on the punctuation quizzes, I determined that there were two causes. The first cause was that the margins between the buttons was off. I checked the other quizzes and noticed that the margins were a different value. I changed the margins on the punctuation quizzes to match the others. The other cause of the issue was that in the question, the word “punctuation” caused the question to appear as three lines instead of two. Reducing the text size allowed me to fix this issue. After fixing those issues, I ran the Debugger again, ran the app in the emulators, and uploaded it to my phone again to continue testing.

Project Closeout

With my app completed, the project closeout process could begin. Project closeout is where the project is finalized. With the testing process over, I deemed the code complete. There

was probably more I could have done on it to improve my app, but the final product met my needs and requirements. On future versions of the app, I may add more features, but for now it is satisfactory. Using the *Java in Easy Steps* book, I was able to change it to a release version and deploy the app as an Android Application Package (APK). Using Android Studio's Version Control System (VCS), I uploaded my Morse Code app to my GitHub account to share it with others. In addition to my project files and code, I also uploaded screen shots of my app in action on my Samsung Galaxy S8. This way if people view my project, they can see that it works on a real device. Another deliverable needed for the project closeout is my journal, which documented my progress on my Morse Code app. I uploaded my journal, as well as my final project report to my GitHub account, so that others can follow along with my progress on the project. Also included within my project report is a list of resources that I used to create my app. My Morse Code app and its resources can be found on my GitHub account at the following link: https://github.com/vlanglais/MyMorseCodeApp

Resources

Android Studio: Create a Multiple Choice Quiz (2016). [Youtube Video]. Wooding, Victor. Retrieved April 11, 2022, from https://www.youtube.com/watch?v=4g1_UH_6VQc

How to Create a Morse Code Converter Android App? (2020, September 1). Retrieved March 29, 2022, from Geeks for Geeks: https://www.geeksforgeeks.org/how-to-create-a-morse-code-converter-android-app/

How to Make a Quiz App in Android? (2021). [Youtube Video]. GeeksforGeeks. Retrieved April 11, 2022, from https://www..com/watch?v=5lmhxob61eg

How to Make a Quiz Application Using Java? - Java Project (2021). [Youtube Video]. GeeksforGeeks. Retrieved April 4, 2022, from https://www.youtube.com/watch?v=utC-8xeEQQA

HTML Color Codes. (2022). Retrieved March 29, 2022, from HTML-Color.Codes: https://html-color.codes/

McGrath, M. (2019). In youtube *Java in Easy Steps.* Retrieved April 12, 2022