**TunePlay**

**EC 327: Final App Project**

**Repository URL:** [**https://github.com/ACKCEC327/EC327-AppProject**](https://github.com/ACKCEC327/EC327-AppProject)

**Casey Flynn**

**Kristof Fogarasi**

**Austin Lent**

**Connor Sheehan**

**Project Documentation**

When sitting down to discuss what application we wanted to develop for our project, there were many factors taken into consideration. The two main driving factors that weighed heaviest in our discussion were how marketable the app concept would be and how feasible it was for us to develop it. Once we had these two cornerstones defined we then moved on to consider what was something that we were all passionate about. This was important to us because we wanted to make the development process enjoyable. After our discussion, we arrived at the decision to create an app that related to music. Once that was decided we then asked ourselves how we could make an app that was most enjoyable and useful for our users. This led to us deciding that we wanted to make a game which then opened up discussion about games we used to play when we were younger. In that discussion, the game Simon came up. Simon is a handheld game that has four colored buttons. The basic concept of Simon is that the game will play a sequence (each button lights up when hit) and then the user will have to play the sequence back. If they play it back correctly, then Simon will play a longer sequence and keep going until the user messes up. We enjoyed this concept of Simon and decided to apply it to our app idea. This allowed us to arrive at the app idea of TunePlay, a mobile app that expands the concept of Simon with a musical twist.

TunePlay is a game app that incorporates music to test and build memory in a fun way. The TunePlay app consists of seven different buttons for the user to interact with: 3 piano keys, 2 guitar strings, a snare drum, and a cymbal. When the user begins playing, TunePlay will create a tune using the seven sounds offered. Once the user has heard the tune, they will mimic this tune by pressing the buttons that correspond to the sounds that they heard in the correct order. As the levels progress, the tune will become longer and thus (hopefully) harder to repeat.

Home Screen

The first point of contact for the user is the crisp, brightly colored home screen layout. We decided to use a color scheme of white, orange, and blue throughout (minus the instrument buttons) for consistency and for simplicity. We thought that the simplicity would appeal to the user.

*Front End*

The first point of contact for the user is the crisp, brightly colored home screen layout. We decided to use a color scheme of white, orange, and blue throughout (minus the instrument buttons) for consistency and for simplicity. We thought that the simplicity would appeal to the user. On the home screen there are three button options for the user: How to Play, Practice, and Begin. These three buttons are in a relative layout along with a TextView that reads “TunePlay”. The order from top to bottom of the relative layout goes: “TunePlay” TextView, “How to Play” button, “Practice” button, then “Begin” button all vertically-oriented.

*Backend*

The backend of the home screen will navigate the user to a new page depending on which button the press (play screen if Play is pressed, directions if How to Play is pressed, and the practice screen if Practice is pressed). Overall, the backend of the home screen is fairly simple. It only consists of navigational methods that launch the desired screen.

Directions Screen

When the How to Play button is pressed on the home screen, the user is then navigated to the directions. As the name suggests, the directions layout is responsible for teaching the user how to play TunePlay.

*Frontend*

The directions layout consists of several direction page slides superimposed on the main directions layout. The functionality of the directions screen utilizes user swiping to change between direction pages. As far as frontend goes, there are XML files for the main directions layout and each superimposed directions slide. The main directions layout XML file includes a line to differentiate between the direction slide and the slider dots, the dots layout itself and a TextView instructing the user to swipe for the next direction. These three components are sibling views to each other and children views to the main directions layout. Then, as mentioned, a directions slide is superimposed on the main direction layout. This direction slide consists of an orange border, a screenshot of the appropriate gameplay layout and a TextView instructing the user how to play TunePlay. The orange border and TextView are sibling views to each other and are children of the main directions layout. The screenshot is a child of the orange border. All but the last direction slide consist of the same layout just described. The final directions slide has just the orange border as a child view to the main direction layout. There are two children views to the orange border view. These are a Play button that directs the user to the play screen and a TextView asking the user “Are you ready to rumble?” In all of the direction slides, the orange background is rounded using a separate XML file called homebutton1.xml

*Backend*

The backend of the directions screen layout is responsible for the swiping capabilities and the dot layout seen in the Direction Screen Layout. After being given the okay, we used a template code found online that provided functionality for the swiping and dot layout. The code was then manipulated to suit the needs of TunePlay. Building off of this code, a play button was implemented on the final direction layout (activity\_direction7) and a corresponding method to launch the play screen when the button is pressed was created.

Practice Screen

The practice screen allows for the user to play with the seven instrument buttons in order to hear the sounds that will make up the tunes they will need to reproduce.

*Frontend*

The front end consists of four linear layouts, one for each set of instruments (drums, guitar strings, piano keys) and the navigational buttons. These four linear layouts are within a relative layout, piano keys placed above the navigational buttons, guitar strings placed above the piano keys and drums placed above the guitar strings. Within each linear layout there are children views. For the drums, there are two children views. For the guitar strings, there are two children views. For the piano keys, there are three children views. Each child view (each instrument view totalling seven) is a button. When the button is pressed, a method is called in the backend to play an audio file. This is further described in the next paragraph. In addition to these seven buttons, there are two navigational buttons within the bottom linear layout. One button reads “Back Home” and the other “Play”. As the button texts suggest, when the Back Home button is pressed, the user is navigated back to the home screen and when the Play button is pressed, the user is navigated to the play screen. Once again, these buttons are consistent with the color scheme of blue, orange, and white.

*Backend*

For the back-end of the practice screen, when each instrument key is pressed by the user a method is executed that plays the sound file associated with each instrument key. This allows the user to press the instrument keys and learn what each sound is like for each key. The user can press these keys as often as they would like to practice. Once the user has had enough practice time, they have the option to press either the “Back Home” button to return to the home screen or the “Play” button to advance to the play screen and begin playing TunePlay. When the “Back Home” button is pressed, the back-end of the code executes a method with the intent of going to the home screen activity. When the “Play” button is pressed, the code executes a method with the intent of going to the play screen activity.

Play Screen

The play screen allows for the user to actually play TunePlay. The user progresses through increasingly difficult levels after correctly reproducing the tune TunePlay plays for the user.

*Frontend*

The overall layout of the play screen is equivalent to the practice screen. The only difference is the bottom linear layout that consists of a button child view and a TextView child view. The button text reads “Play Tune” and the TextView reads “Your Turn”. When the Play Tune button is pressed, TunePlay plays the tune that the user must reproduce. Once the Play Tune button is pressed, it is set to invisible (so that the user can’t cheat and listen to the tune multiple times) and the Your Turn TextView takes up the entirety of the bottom linear layout. In addition to the play screen there are also three additional layouts that will pop up during gameplay. Which layout is called depends on whether or not the user correctly reproduced the tune they were played or if the user beats the game. If the user correctly reproduces the tune, a “Next Level” layout will appear. The layout consists of a button and two TextViews all within a relative layout. The top TextView shows a congratulatory message “Nice Job!”, the second TextView shows the message “Let’s try a more difficult tune…”, and the button below it reads “Next Level” which allows the user to move on to the next level. When this button is pressed, the user is navigated to the play screen, starting the next level. The corresponding backend java code only contains a method that launches the play screen.

If the user does not correctly reproduce the tune, a “Game Over” layout will appear. This relative layout consists of two TextViews and two buttons. The top TextView reads “Game Over” and the second TextView reads “You didn’t play the correct tune”. Below this TextView is a button that reads “Play Again”. If this button is pressed the user is navigated to the play screen starting over at level one. Below the “Play Again” button is a “Home” button. If pressed, the user is navigated to the home screen. The corresponding java code contains a method that launches the home screen when the Home button is pressed and a method that launches the play screen when the Play Again button is pressed.

If the user makes it through all ten levels (more will be created), a “Congratulations” layout will appear. This is a relative layout consisting of three stacked TextViews above a button. The top TextView reads “CONGRATS!”, the second TextView reads “You have defeated TunePlay!”, and the bottom TextView reads “(More levels coming soon)”. The button below the three TextViews reads “Back to home”. As the text suggests, if the button is pressed, the user is navigated to the home screen. The corresponding backend java only consists of a method that launches the home screen.

*Backend*

The back-end of the play screen is what took up the majority of our time and effort when creating this project. This is the code that allowed the game to execute all of its functions properly. Once inside of the play screen class, we first declare some variables and arrays that will be used throughout the methods in this class. This includes the selected song array, the array size, a count variable, and a boolean variable, donePlaying, that determines whether or not the tune has finished playing for the user. A switch statement is used to determine which song arrays could be used for each level of the game. For each level, there are three different options of song arrays that could be played for the user. A random number generator is combined with an if-statement in order to randomly select one of the three song arrays to play for the user. The same methods from the practice screen back-end code are included in the play screen code in order to play the sounds associated with each instrument button. When on the play screen, the user can press the “Play Tune” button which calls the playSongFunc method. This method uses a switch statement to read through the song array and determine which method to call in order to play the correct sound for the tune selected. This method also includes an exception in order to implement a delay between the different sounds being called for the tune. Once the playSongFunc method has gotten to the end of the song array, the donePlaying variable becomes “true” and the method knows that the entire tune has been played.

Once the entire tune has been played for the user in the play screen, it is then time for the user to press the instrument buttons on the screen to replay the tune that was played for him/her. Within the play screen back-end code, there is a check method that is called once the user presses an instrument button to create a sound. This method checks the user’s selected input against the song array that was played for the user in order to determine whether or not the user has selected the right button at the right time in order to reproduce the tune. The check method uses the count that was initialized at the beginning of the play screen class in order to index into the song array. The method then uses an if-statement to determine whether or not the user input matches the indexed part of the song array. If the user input does match that index of the song array, then the count is incremented. A second if-statement is then called that determines whether or not the user has reached the end of the song array and played the entire tune back correctly. If the condition is not met for this first if-statement, then the home screen’s global level variable is reset to one, and the user is sent to the game over screen. On the game over screen, the user has the option to press two buttons: the play again button or the home button. If the play again button is pressed, a method is called with the intent to return to the play screen and start the game over again. If the home button is pressed, a method is called with the intent to return to the home screen.

If the user input matches the index of the song array, the count is incremented and the second if-statement is called to determine whether or not the user made it to the end of the song array. If the count is equal to the size of the array, then we know that the user has selected all of the instrument buttons correctly and made it to the end of the song array. If this occurs, then the home screen’s global level variable is incremented. Another if-statement is then used to determine whether or not the user has passed the last level of the game. If the global level variable is greater than 10, then a method is called with the intent to send the user to the won game screen. On the won game screen, the user can press a “Home” button which calls a method with the intent to return to the home screen. If the global level variable is not greater than 10, then a method is called with the intent to send the user to the next level screen. On the next level screen, the user can press a “Next Level” button which calls a method with the intent to go back to the play screen with the incremented level variable. This incremented variable will allow the song arrays for the next level to be implemented and one of the newer, harder song arrays will be chosen and played for the user. If the user has pressed the correct button, but has not made it to the end of the song array yet, then the program waits for the user to press another button and run the check method again.

Challenges Faced

While working on this project, we ran into a few problems along the way. One of our first problems had to do with getting the music to play at the correct time when the user was on the play screen. Originally we intended for the game to send the user to the play screen, and then once the user was there the tune would play for the user automatically and all the user would have to do was use the instrument buttons to recreate the tune once it had finished playing. However, we ran into an issue because the song array would play before the play screen appeared for the user. We couldn’t understand why this was occurring, but to fix the problem, we decided to create a button on the play screen that the user could press in order for the tune to play once the user was already on the play screen.

Next, one of our original ideas for this app was for the keys to light up as the sound was played for its corresponding button.That way, the user would know which sound was associated with which instrument button, and it would be clear to the user which button needed to be pressed in order for the correct tune to be imitated. However, we could not coordinate the buttons to light up as the noise for each button was being played for the user. To make up for the fact that we could not show the user which button was creating the sound as it was being played for the user, we decided to create a practice screen, which was not our original plan for this app. The practice screen allows the user to become familiar with the sounds that each button makes before attempting to play the game themselves. This makes the game feasible, but a bit more challenging for the user since they have to rely on the sound of the tune alone.

Our next problem occurred when we were attempting to get the instrument buttons to play sounds when the user pressed them. Originally for the methods in reference to onClick xml attribute, we were not passing any arguments to the methods, which prevented the instrument buttons from playing sound. We learned that we needed to include a view as the input argument parameter for the method in reference to the onClick xml attribute. This connected the sound bytes to their respective buttons.

Another problem occurred when we attempted to create too many media player variables. A media variable represents the tune sound to be played when a button is pressed. The instrument buttons could no longer play sound at a certain point because too many media player variables had been created. We solved this problem by only creating the media player variables at level one of the game. We then called upon these variables in the subsequent levels, so that we wouldn’t have to recreate them. Every time the instrument button was pressed, we would start the media player, have a delay of half a second, stop the media player, and prepare the media player to run again.

Our final problem occurred when we tried to press the same instrument button two times in a row. When we tried to hit the button a second time, we could not get the app to play the second note. This occurred because the time delay between playing the same sound byte twice was shorter than the length of the sound byte itself. In other words, we were not allowing enough time for the first sound byte to play before the second one was being called. We solved this problem by implementing a stop method, which would stop playing the sound clip after half of a second, and a prepare method, which got the sound clip ready to be played when it was called the second time.. This allowed enough time for us to stop the sound byte and set it up to be ready for replay before the button was pressed by the user a second time.