Sound Sampler

Senior Design Group 56

Zachary Besta, Eric Stablein, Dalton Sherratt

Engineering Standards and Design Practices

- Java Code Conventions [1]
- MP3 and .WAV audio formats
- Android Core app quality standards [2]
- IEEE terminology
- Google Play Store requirements

Summary of Requirements.

- The application must be able to run on a fairly modern Android device
- The application must target Android 8.0 or newer per Play Store requirements
- The application must have an easy-to-use, intuitive UI
- The product must sound good to musicians
- The application must have the following features:
    - Pitch-shifting
    - Speed-shifting
    - Envelopes
    - Adjustable band-pass filter
    - Ability to record and save application output
    - Haptic feedback
    - Ability to save and load presets
- The application must conform to the requirements of the Google Play store
- The application needs to have an appealing sound to musicians, both hobbyist and professional

Applicable Courses from Iowa State University Curriculum

- Com S 227/228 - introduction to Java programming
- Com S 309 - Software Development Practices
- EE 224 (Signals & Systems I) - introduces the Fourier transform and use of MATLAB for digital signal processing
- EE 285 - introduction to good coding practices using C

- EE 321 - further explored signal process concepts and working with Simulink

New Skills/Knowledge acquired that was not taught in courses

- Android development
- Generation of envelope functions
- Audio processing on Android

Table of Contents

List of figures/tables/symbols/definitions

Figures:

1 Introduction

1.1 Acknowledgement

- Drs. Randall Geiger and Degang Chen served an advisory role and offered technical advice.

1.2 Problem and Project Statement

While there exist fully-featured samplers on iOS, samplers on Android have a limited feature set. Many of the samplers on the market do not feature an envelope function, for example. Such a function is used heavily in the world of samplers.

As well, existing sampler applications for smart devices fail to offer users an intuitive view of the effects of their actions. Users can work with the applications but need to rely on external documentation and knowledge to understand how to use them.

The existing sampler apps for Android lack important features and are inaccessible. This project is a fully-featured Android sampler app geared around offering a clear picture of what is happening and an accessible UI.

1.3 Operational Environment

The end product will be an application for Android devices. The operational environment is a tablet, phone, or other smart device running the Android OS. One of the constraints that comes with this is the wide variation in processing power between different devices.

1.4 Requirements

Functional requirements

- The application must be able to run on a fairly modern Android device

- The application must have an easy-to-use, intuitive UI
- The application must have the following features:
    - Pitch-shifting
    - Speed-shifting
    - Envelopes
    - Adjustable band-pass filter
    - Ability to record and save application output
    - Haptic feedback
    - Ability to save and load presets

Economic requirements

- The application must conform to the requirements of the Google Play store
- The application needs to have an appealing sound to musicians, both hobbyist and professional

1.5 Intended Users and Uses

The product has two major user groups: hobbyists and performing musicians. Hobbyist musicians will use the app as a more robust version of existing sampler apps that better substitutes for a physical sampler. Performing musicians will use the app to create and perform music on a more portable device.

1.6 Assumptions and Limitations

Assumptions:

- The application will run on a fairly modern (released within the last five years) Android device
- The maximum number of simultaneous inputs to the app is less than ten

Limitations

- The application will need to be lightweight enough to run on legacy devices
- The application will need to output audio in a format compatible with how Android interfaces with speakers

1.7 Expected End Product and Deliverables

End product: sampler application for Android

Delivery date: week of April 27, 2019

The end product will be a sampler application for Android with the features listed in section 1.7. It will also include refinements based on feedback from alpha and beta testing.

Deliverable: speed-shifting component

Delivery date: week of January 20, 2020

This component will be the code to change the speed of a piece of audio.

Deliverable: envelope component

Delivery date: week of February 3, 2020

This component will be the code and UI to apply a user-defined ADSR or ADHSR envelope to a piece of audio.

Deliverable: pitch-shifting component

Delivery date: week of February 10, 2020

This component will be the ability to change the pitch of a piece of audio while maintaining its tempo.

Deliverable: equalizer component

Delivery date: week of February 23, 2020

This component will be the code and UI to apply an equalization curve to a piece of audio.

Deliverable: UI

Delivery date: week of March 9, 2020

This component will be the user interface that ties together all the app's functions.

Deliverable: feedback from beta testing

Delivery date: week of March 23, 2020

Beta testing will take place with a group of musicians, and their feedback will be recorded.

2. Specifications and Analysis

2.1 Proposed Design

The proposed design is an Android sampler app with a simplified UI that increases the amount of user control. The application will be split into several sub-components, which are controlled either from the main screen of the app or from their own screen, depending on their size. The application will also have a dedicated "play" mode that hides and disables editing features to save screen space.

We have studied existing sound sampling devices and software in order to understand the current designs and develop an idea of what our final application will include. We have experimented with Android Studio in order to become familiar with using it as an IDE. We have developed a set of use cases to implement as a starting point for app development.

Much of our time was spent on developing the functional and non-functional requirements of the application.

Some of the functional requirements include:

- Modifying the pitch, volume, and speed of the sound

- Implementing an equalizer in order to modify the treble, bass, and mid-range frequencies of the sound
- Saving samples of the modified sound
- Assigning samples of sound to buttons on the virtual drum-pad
- Being able to record the track created by the user as they manipulate the drum-pad and dials

Some of the non-functional requirements include:

- Appearance of the application
- Layered design approach for the classes within the application
- Modular design for the classes within the application
- Compatibility with the majority of Android devices

2.2 Design Analysis

So far, nothing has been created or tested. However, there are still several changes that have been made:

- Per feedback from Dr. Geiger, the app's scope was changed to only be a sampler. Initially, the app would've been controllable via guitar, but the group worried the scope was too large, as did Dr. Geiger. The approach of removing the guitar from the equation allows a functional sampler to be created that another group could later add guitar control to.
- Initially, the application would not have a separate mode just for playback.

2.3 Development Process

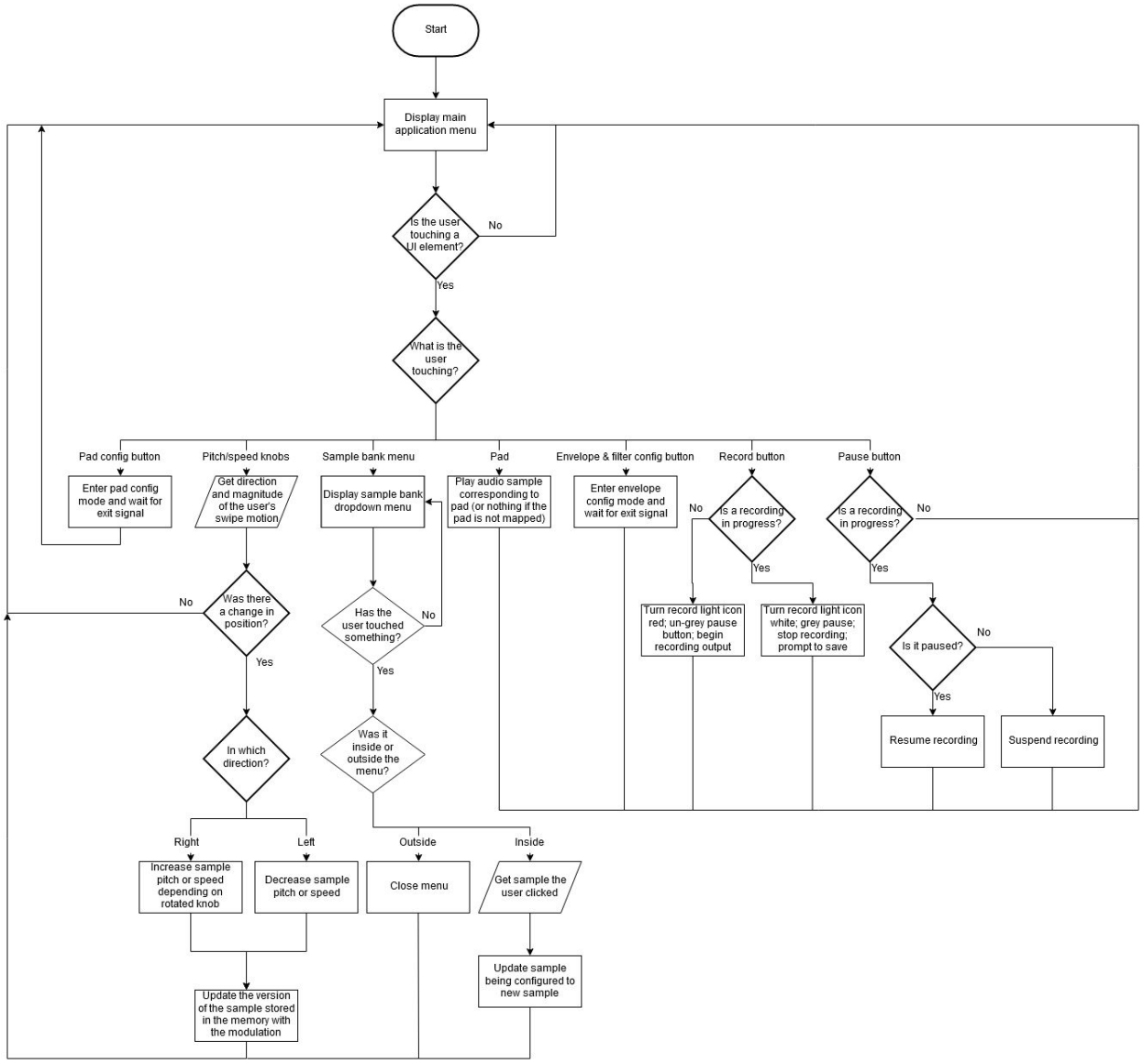
The project will follow a Waterfall process. The main reason for this decision is the small group size. Because the group only has three members, and only one member has Java experience, the group will often be working on the same thing together. The linear workflow of the Waterfall process fits well with this.

2.4 Design Plan

The design plan is to implement a test-first design for the functionality within the application. This will include writing test cases for different functions of the app, then writing code to meet those test cases. The structure of the application code itself will follow the layered design approach. This will ensure that the user interface will be a separate class from the class that reads/writes the files to memory. These classes will be separate from the class that modifies the pitch of the sound, etc. This design approach will ensure that we have modular code that will be easy to maintain and scalable.

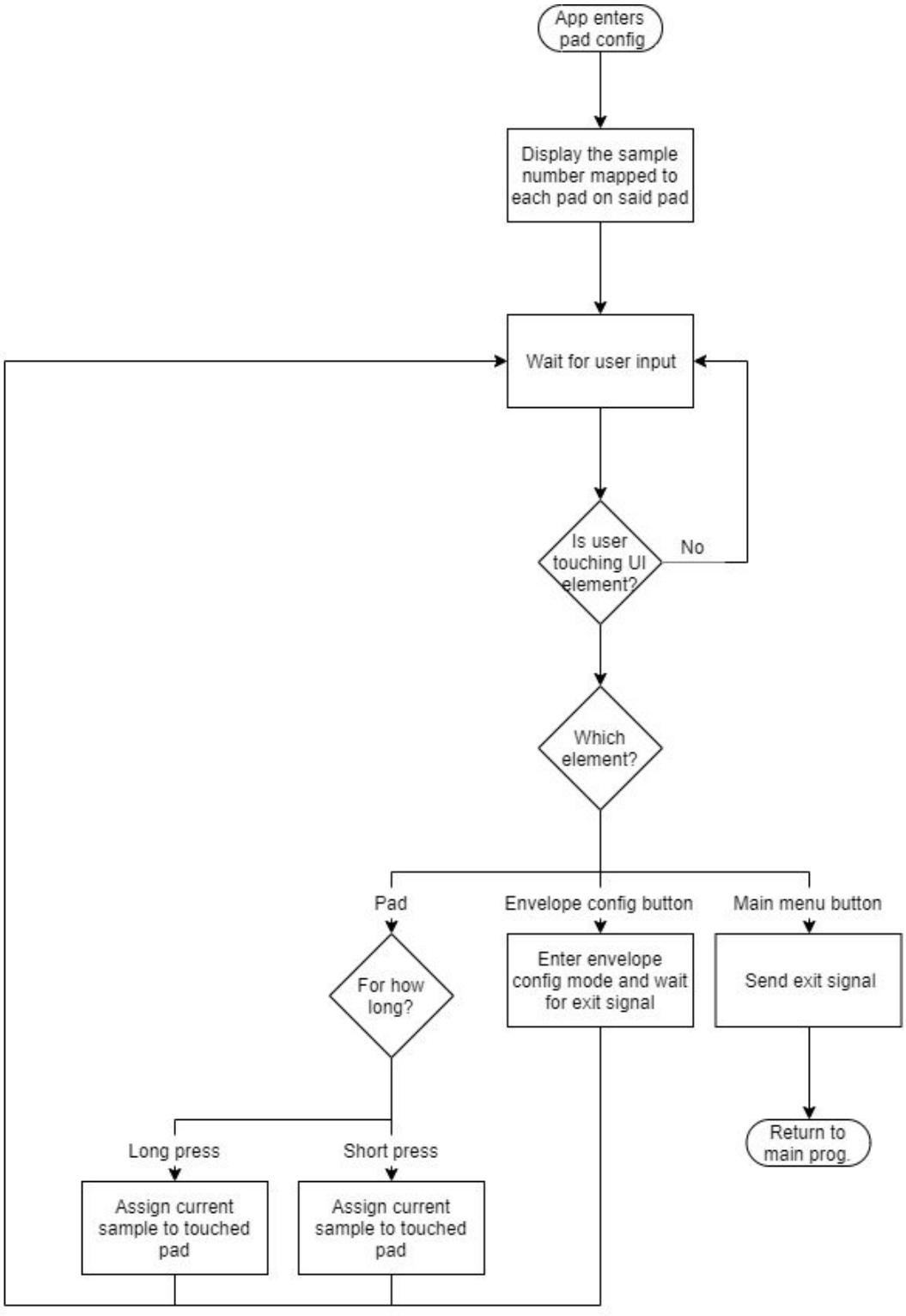
We have defined each use-case and will complete functionality for each use case separately.

We have developed a flowchart diagram that will illustrate the flow between the main menus:


Start
Display main application menu
Is the user touching a UI element?
No
Yes
What is the user touching?
Pad config button
Pitch/speed knobs
Sample bank menu
Pad
Envelope & filter config button
Record button
Pause button
Enter pad config mode and wait for exit signal
Get direction and magnitude of the user's swipe motion
Display sample bank dropdown menu
Play audio sample corresponding to pad (or nothing if the pad is not mapped)
Enter envelope config mode and wait for exit signal
Is a recording in progress?
No
Yes
Is a recording in progress?
No
Yes
Turn record light icon red; un-grey pause button; begin recording output
Turn record light icon white; grey pause; stop recording; prompt to save
Is it paused?
No
Yes
Resume recording
Suspend recording
Was there a change in position?
No
Yes
Has the user touched something?
No
Yes
In which direction?
Was it inside or outside the menu?
Right
Left
Outside
Inside
Increase sample pitch or speed depending on rotated knob
Decrease sample pitch or speed
Close menu
Get sample the user clicked
Update the version of the sample stored in the memory with the modulation
Update sample being configured to new sample

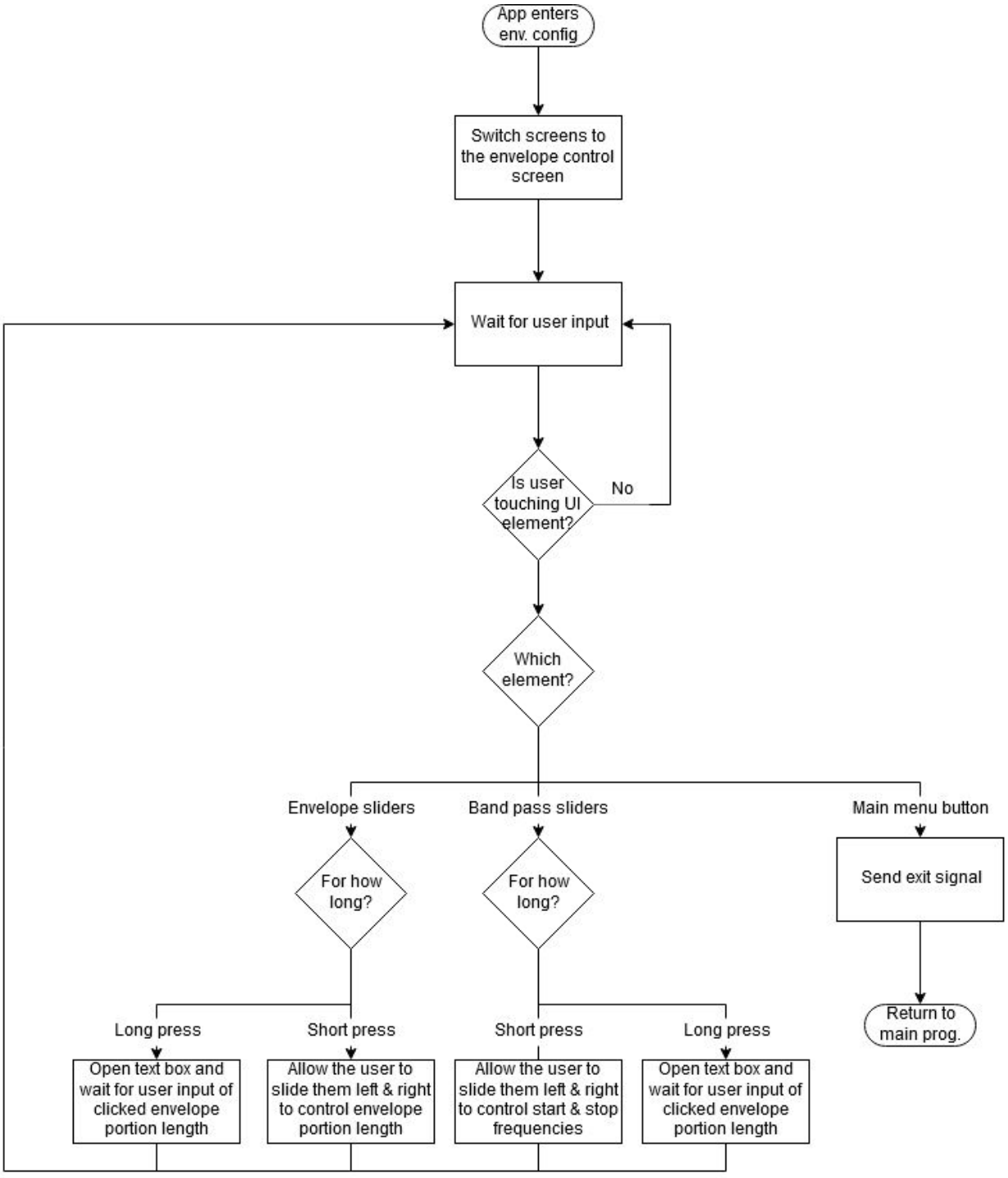
(Figure 1)

We have also developed a flowchart to illustrate the drum-pad functionality:



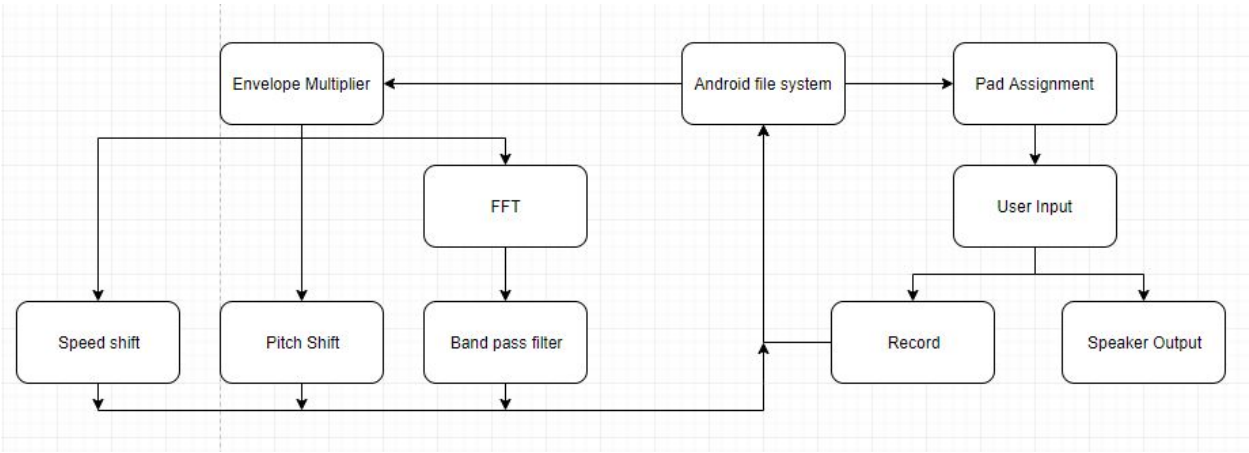
(Figure 2)

We have developed a flowchart to illustrate the envelope functionality:



(Figure 3)

Below is a block diagram that illustrates the flow between functionalities within our app:



(Figure 4)

3. Statement of Work

3.1 Previous Work And Literature

The Akai MPC series is a line of samplers created in 1988 that allows musicians to take existing sounds and music then map them to 16 rubber pads. When a pad is hit, the MPC plays the mapped sound. MPC series samplers can cost anywhere from 120 dollars to upwards of 1000 dollars. [3]

The Akai iMPC sampler app, which emulates the MPC, is no longer available for Android as of October 2019. [4]

Some examples of existing sampler applications include the following:

App	**Price**	**Functions**
Nanoloop [5]	$3.49	Envelope (AD only) Start offset

Pocket Sampler - DJ Launchpad [6]	$2.99	Online database
iMPC (Apple only) [7]	$6.99	Time correction Effect modules Recording and overdubbing Step sequencer
G-Stomper Studio [8]	$12.99	Step sequencer Effect modules Real-time modulation

3.2 Technology Considerations

Android Studio

Strengths:

- Intuitive UI that allows all members to contribute
- Simplifies UI creation
- Commonly used for Android Development
- Features a debugger

Weaknesses:

- Requires the members to learn a new IDE

Android Studio was chosen over other Java IDEs because of its overall simplicity. The way it simplifies creating a UI was a large plus given the team's lack of experience in Android development. As well, it provides similar features to many other IDEs. Even though the team is more experienced with Eclipse, Android Studio provided a large number of benefits.

Monkey

Strengths:

- Allows for stress-testing the app via pseudo-random inputs

- Pseudo-random nature allows for tests to be repeated

Weaknesses:

- Does not target specific cases

3.3 Task Decomposition

Most of the tasks in this project require that we have a working way to load audio from the Android file system, then play it from our app's memory. Loading and playing audio will serve as the backbone of the app. Once that works, the audio processing tasks can be implemented individually. These tasks are the pitch-shifting, speed-shifting, filtering, and envelope functions. Then, they need to be combined and tested to ensure that they function alongside each other. Finally, they and the playback features need to be implemented together into a UI.

3.4 Possible Risks And Risk Management

Our team has limited business experience. This means that we may not have the knowledge to judge the economic requirements well. We also have limited knowledge of the app marketplace and its business practices. To mitigate this, we will try to keep updating our economic requirements as we learn new things and monitor the app marketplace.

One smaller risk that our project faces is that the implementations of our equalizer and pitch-shifter could be more complex than initially envisioned. To handle this, the communications/signal-processing-focused students can consult with professors in the area. The equalizer is less likely to be an area of risk than the pitch-shifter as the students have experience working with equalizers.

Another lesser risk is our team's limited Java programming experience. To mitigate this, the team is using Android Studio for app development because it is a fairly simple IDE. Also, the team will be making heavy use of pseudocode and paired programming. This also serves the purpose of allowing the most experienced coder of the group to receive help in understanding signal-processing functions.

3.5 Project Proposed Milestones and Evaluation Criteria

The first key milestone is having a functional equalizer. In order to test the equalizer functionality, we plan to compare the audio output to the output of an existing equalizer. We also plan to test a series of random inputs using Monkey.

The second key milestone is having a working sample speed shifter. This can be tested using samples of various lengths and comparing the output of our app to an existing equivalent.

The third milestone is implementing frequency-shifting. This will be tested again using a set of samples of various lengths, initial frequencies, and amplitudes. This will let us cover a wide variety of types of inputs.

The next milestone after this is implementing an envelope function. This will require that the software is capable of creating and applying an ADSL envelope based on user-defined parameters.

The final milestone after this will be having a working user interface. This will be a user interface that integrates all of the effect modules and lets the user control them.

3.6 Project Tracking Procedures

Our group will use Google Drive and Git to track progress. Slack will be used to coordinate weekly meetings.

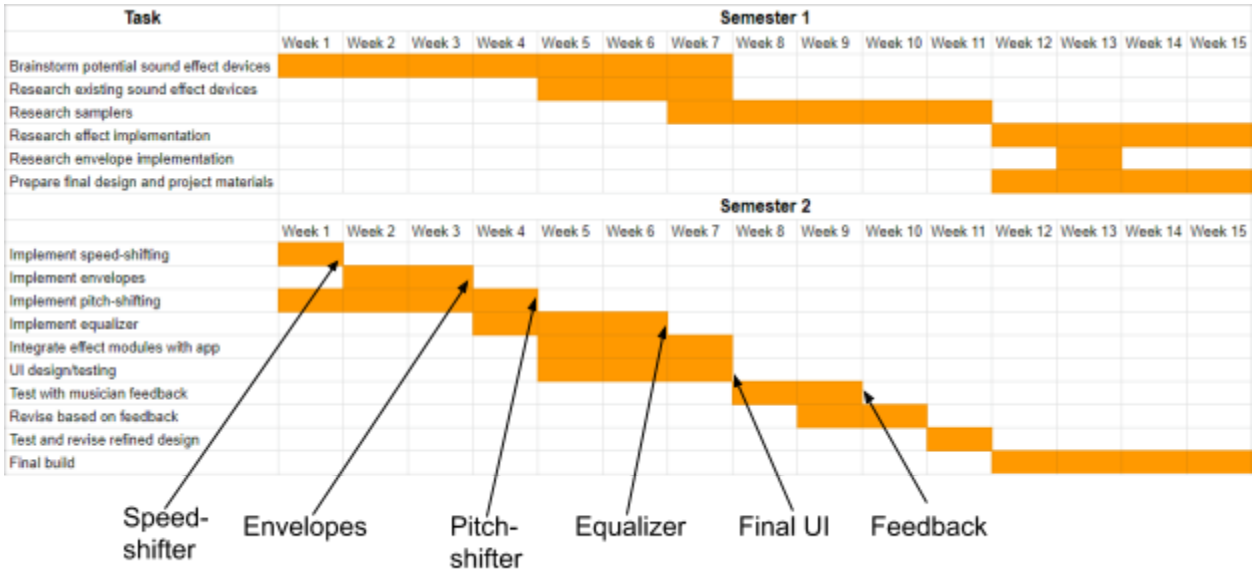
3.7 Expected Results and Validation

The desired outcome is that we would be able to create a sampler app that lets users understand and feel involved in the process of performing with a sampler. Our plan to ensure our solutions work is to perform beta and final testing with performing and hobbyist musicians.

4. Project Timeline, Estimated Resources, and Challenges

4.1 Project Timeline

Project Gantt chart:



(Figure 5)

The group began the project by brainstorming potential sound effect devices. Once the group had a number of ideas, market research was performed to narrow the list. The group decided to work on an Android sampler app based on the problems with existing offerings.

One of the major problems with existing offerings was the lack of effect modules. The group spent the remainder of the first semester researching how said modules work and determining how to implement them. At the same time, they worked on the end-of-semester deliverables.

In the second semester of the project, the group will spend the first six weeks implementing the effect modules. In week 5, the group will begin integrating the effect modules into the application

and designing a UI. The group will follow a process of testing alongside development during this phase.

4.2 Feasibility Assessment

The project will be an Android sampler application, including several effect modules and an intuitive UI. Challenges that will be faced throughout the project process include implementing envelopes, UI design, and working with Java.

To handle implementing envelopes, the group has sought educational resources on the topic, including a page from McGill University [9]. Though the advisors foresee this being difficult, the educational resource was quite helpful, and this seems like a less major challenge than the others.

UI design is a new challenge for the group. In order to create an accessible UI, the group has to think from a new perspective. The group will research existing sampler applications as well as apps for different purposes to determine and mitigate elements that are hard to understand.

While the group has one member who is a CprE student, the other members do not have experience working with the Java programming language. To bridge this gap, the group plans to make extensive use of pseudocode. This also has the benefit of translating many of the signal processing algorithms into an accessible form, as the CprE student does not have signal processing experience.

4.3 Personnel Effort Requirements

Task	Roles	Estimated time (total hours)
Sound effect device brainstorming	Full team - ideas for interesting sound effect devices	20
Sound effect device research	Eric and Zach - comparison of different sound effect devices Dalton - examples of additional devices and comparison	Zach - 12 Eric - 12 Dalton - 14
Sampler research	Eric - comparison of different samplers	Eric - 32

	Zach - compilation of sampler features Dalton - information on sampler UIs	Zach - 24 Dalton - 24
Speed shifting	Zach - code that shifts speed of samples	Zach -2
Pitch shifting	Eric - create classes for shifting pitch of samples	Eric - 4
Envelope	Zach - create classes for applying envelopes to samples	Zach - 4
Equalizer	Zach & Eric - create equalizer that takes desired function and applies it to sample	Zach - 3 Eric - 3
Graphical User Interface	Dalton - create .xml files for screen flow and app layout, including equalizer, envelope, and drum pad screens	Dalton - 6
Save to memory and retrieve from memory	Dalton - implement classes to write to/pull from the device's memory	Dalton - 2
Playback	Dalton - implement a playback interface	Dalton - 2
Testing	Eric, Zach, and Dalton - Debugging, Mockito, and J unit	Eric - 2 Dalton - 2 Zach -2

4.4 Other Resource Requirements

- Computers with the Android Studio IDE and Java Development Kit
    - The group members will be using their personal computers
- Android devices for testing
    - Can use emulator within Android Studio IDE

No other resource requirements are predicted.

4.5 Financial Requirements

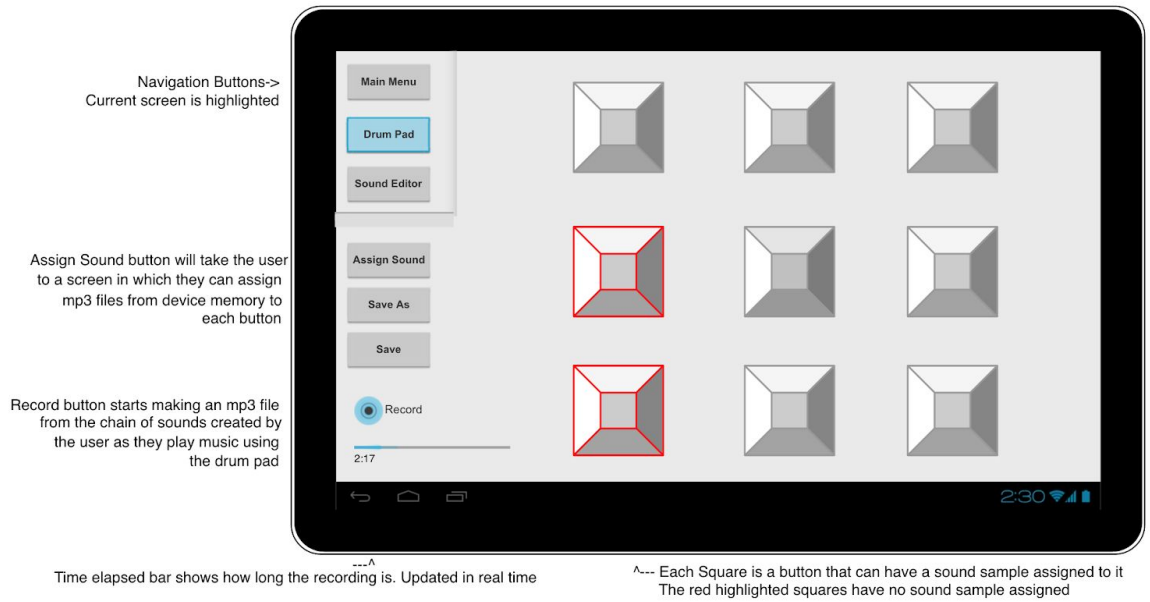
All software used in the project is free, so that will not contribute to the project cost. One potential cost will be acquiring devices to test the software. While the group can use their own devices, some additional devices may be necessary.

5. Testing and Implementation

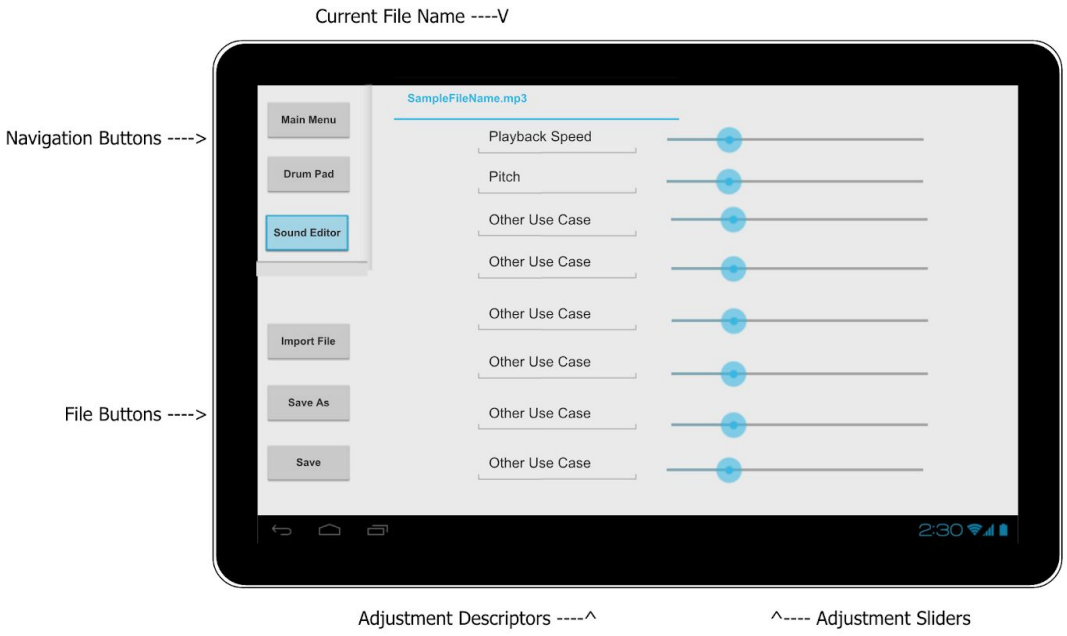
For testing our application, we will utilize Mockito and JUnit testing. We will use a test-first design approach. In which, we will write our JUnit tests, then write modules to satisfy those tests. For portions of the application that are not feasible to design with a test-first design, we will use Mockito testing to test for runtime errors.

5.1 Interface Specifications

We will implement a graphical user interface for the user to be able to manipulate the drum pad more easily. See figures below for screen sketches:



(Figure 6)



(Figure 7)

5.2 Hardware and software

Monkey is a piece of software that allows a pseudo-random set of inputs to be sent to an app for testing purposes.

Android Studio is an IDE for Android development that includes a debugger functionality.

5.3 Functional Testing

- The application will be compared to existing software for PC that performs similar effects to ensure effects work as expected
- Expected output and obtained output will be compared for a variety of length, amplitude, and frequency content values for each effect module

5.4 Non-Functional Testing

- Monkey will be used to test the app against a random series of inputs
- The app was originally going to be tested with musicians to obtain feedback and make any necessary changes
    - This phase would've mainly focused on the UI and features, but was canceled after the university closure

5.5 Process

The testing process will include unit testing as each module is developed. This will be used to ensure each part is working and finished. After that, the group will integrate the components into a control application. This will require the group to test different combinations of the components if one of them is not working with the others.

Once all of the components are working together, the control UI will be tested by using Monkey to send a random series of inputs. Then, the group will perform alpha testing, mainly focusing on cases that could potentially cause errors and overall functionality. Once the alpha testing is finished and any necessary changes are made, the group will start a beta test using actual musicians. Based on their feedback, any final revisions will be made. The end of the semester has deliberately been left open to allow more time for revisions and testing them.

5.6 Results

We have tested the layout of the application by using the Pencil application for creating screen sketches and flowcharts. We learned from these models exactly how the objects on our screens must be oriented in order for the best usability for the customer. The models gave us an accurate idea of how to avoid the application being difficult or "clunky" to use.

6. Closing Material

6.1 Conclusion

At the end of the first semester, the group had a strong design framework. Much of the app was already planned and designed, so the future left only implementation and testing. Creating a fully-functional Android sampler app with an intuitive UI addresses the two largest shortcomings of the currently available apps.

In the second semester, the group began implementing the effects. Then, the group designed a user interface. Finally, the group entered a phase of testing and refining the product with alpha and beta phases. While there were some setbacks and delays, all of the app's basic functionality is working, and most extra functions save the equalizer and envelopes work well within the app.

6.2 References

[1] "Java Code Conventions." Sun Microsystems, Inc.. Available: https://www.oracle.com/technetwork/java/codeconventions-150003.pdf. [Accessed Dec. 8, 2019]

[2] "Core app quality." Android Developers. Available: https://developer.android.com/docs/quality-guidelines/core-app-quality. [Accessed Dec. 8, 2019]

[3] "Meet the unassuming drum machine that changed music forever." A. Aciman. Available: https://www.vox.com/culture/2018/4/16/16615352/akai-mpc-music-history-impact. [Accessed: Dec. 8, 2019].

[4] "iMPC for Android," AKAI Professional. [Online]. Available: https://www.akaipro.com/impc-for-android. [Accessed: Nov. 18, 2019].

[5] “nanoloop,” Google Play Store. [Online]. Available: https://play.google.com/store/apps/details?id=com.nanoloop&hl=en_US [Accessed Dec 12, 2019]

[6] “Pocket Sampler - DJ Launchpad,” Google Play Store. [Online]. Available: https://play.google.com/store/apps/details?id=info.superkiki.pocket.sampler&hl=en_US [Accessed Dec 12, 2019]

[7] “iMPC on the App Store,” App Store. [Online]. Available: https://apps.apple.com/us/app/impc/id584548447 [Accessed Dec 12, 2019]

[8] “G-Stomper Studio,” Google Play Store. [Online]. Available: https://play.google.com/store/apps/details?id=com.planeth.gstomper&hl=en_US [Accessed Dec 12, 2019]

[9] “Audio Envelopes,” McGill University. Available: https://www.music.mcgill.ca/~gary/307/week1/envelopes.html. [Accessed: Oct. 27, 2019].

6.3 Appendices

This link is to the project shared folder. It requires an Iowa State University login.
https://drive.google.com/drive/folders/1J6CoUSB8kGlspMl5Rx_ZoIqqcnhsiKBc?usp=sharing

The project website is available at http://sdmay20-56.sd.ece.iastate.edu/.