# Sound Effect Device for Musicians

Team: sdmay20-56
Website: http://sdmay20-56.sd.ece.iastate.edu
Client/Advisor: Dr. Randy Geiger, Dr. Degang Chen
Members: Dalton Sherratt, Eric Stablein, Zach Besta

## Goals

Create an easy to use sampler for the Google Play Store that offers musicians high-end functionality with a simple user interface

### Functional Requirements

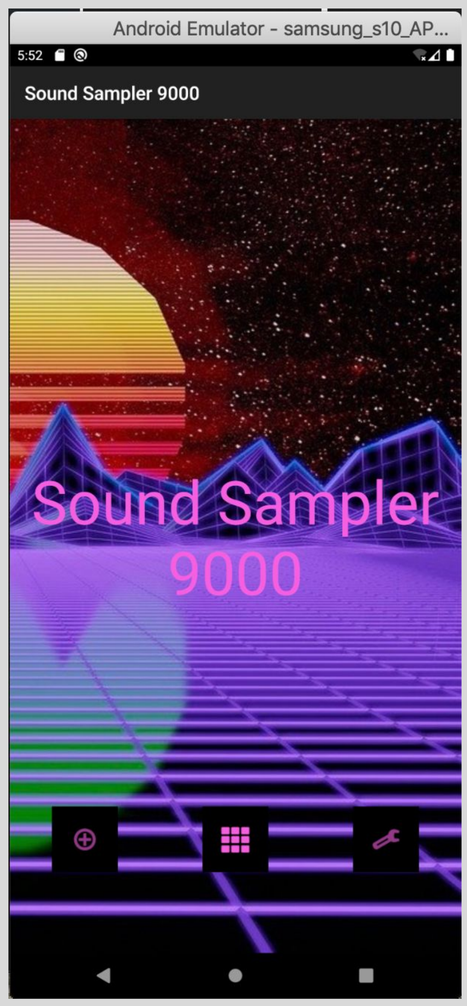
- Capable of recording and looping audio
- Include features to adjust Pitch, Speed and Tempo of audio files
- Can Save tracks to Android file system

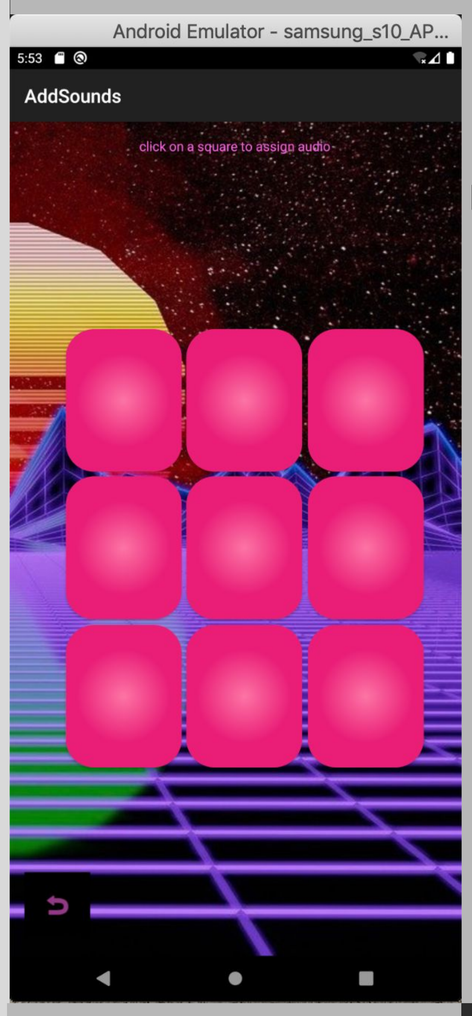
### Other Requirements

- Warn users about hearing damage at high volumes
- Make sure users know to take breaks to avoid repetitive stress injuries

### Environment

- Run on modern Android devices including smartphones and tablets
- Meet the requirements of the Google Play Store
  - Must target API 28+


## Intro/Motivation

Existing samplers on the Android Marketplace are either expensive and hard to use or lack major features required by musicians.

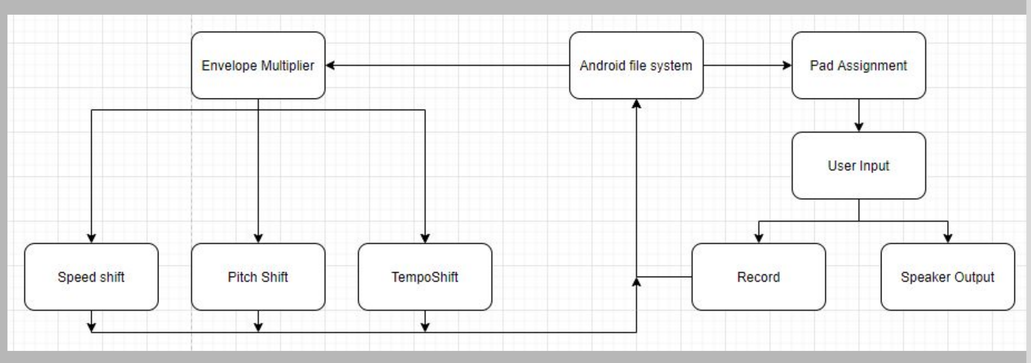
Beginner musicians need a way to easily create sampler-based music on their Android devices without being overwhelmed or committing to a high price.

Our solution to this problem is to create an application for Android that presents all the features of a high-end sampler in a more intuitive way, then offer it at a price point occupied by low-end samplers.

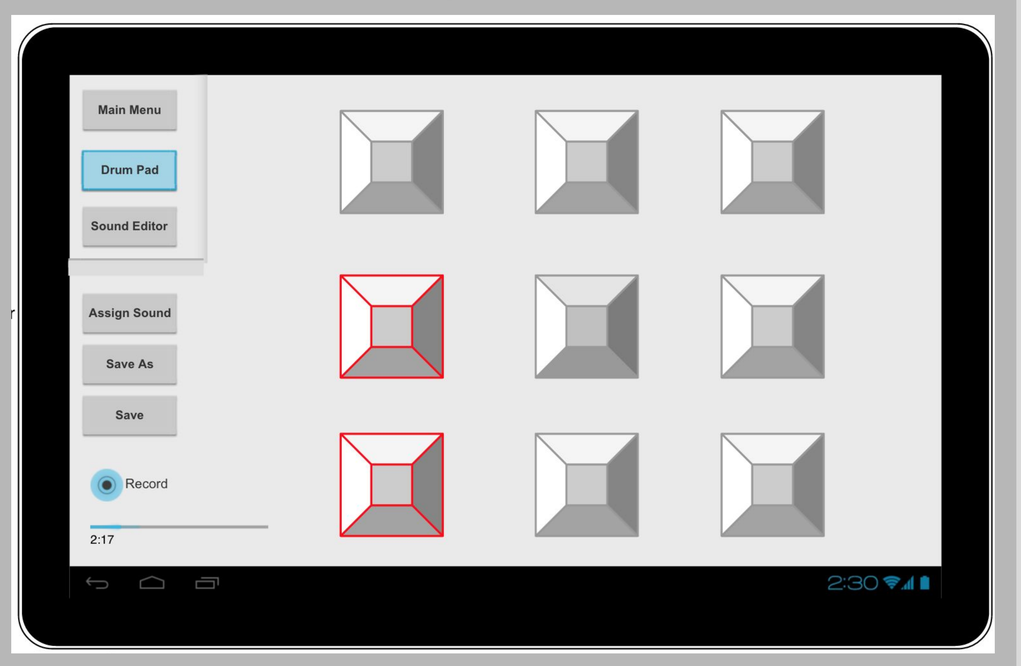
Potential users of this product include new musicians who want a simple and inexpensive sampler and experienced musicians looking for a lightweight sampler for on-the-go use.

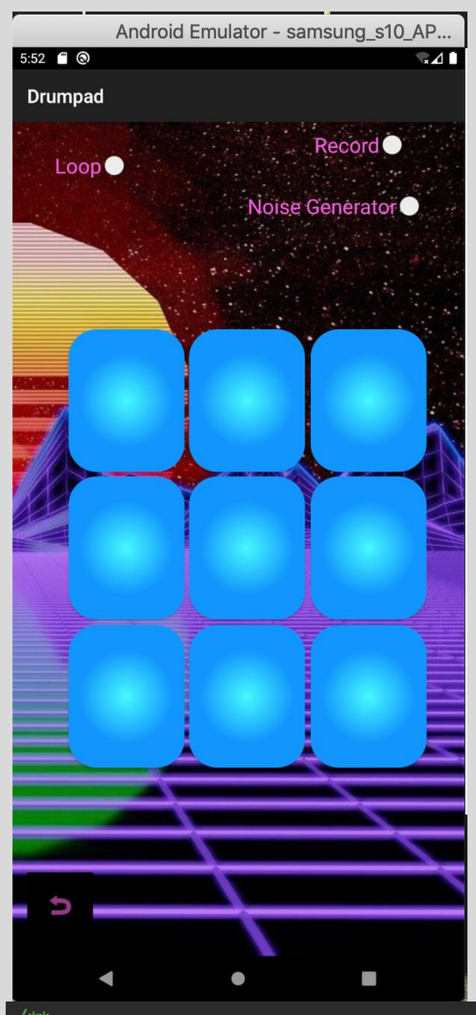
## Design Approach

The sampler was broken into a number of smaller effects modules. Then, the modules could be developed individually and integrated later. All modules were implemented in Java.




### Conceptual Sketch




## Project Resources

### Programming Language

Java

### Android Libraries

MediaPlayer
SoundPool
AudioTrack

### Development Tools

Android Studio

### Source Control

GitLab

### Testing

Mockito
JUnit

```
ImageButton speedup = findViewById(R.id.speedup);
speedup.setOnClickListener(mixerLogic);
ImageButton speeddown = findViewById(R.id.speeddown);
speeddown.setOnClickListener(mixerLogic);

ImageButton tempoup = findViewById(R.id.tempoup);
tempoup.setOnClickListener(mixerLogic);
ImageButton tempodown = findViewById(R.id.tempodown);
tempodown.setOnClickListener(mixerLogic);

    mp1 = MediaPlayer.create( context: this, R.raw.sound1);
}//END OF ONCREATE

/**
 * This logic manipulates the sound file so that the user
 * can pitch shift, change temp, etc
 */
private View.OnClickListener mixerLogic = new View.OnClickListener(){
    public void onClick(View v){
        switch (v.getId()){
            case R.id.pitchup:
                pitch1++;
                break;
            case R.id.pitchdown:
                if(pitch1>=1)
                    pitch1--;
                break;
            case R.id. speedup:
                speed1++;
                break;
            case R.id.speeddown:
                if(speed1>=1)
                    speed1--;
                break;
```

```
/**
 * Method to generate sine wave white noise
 * @param frequency is the frequncy of the wave in Mhz
 * @param duration is the number of seconds that the wave is generated
 */
private void generateSoundWave(int frequency, int duration) {
    int sampleRate = 5000; //sampleRate, increasing this changes pitch
    byte soundData[] = new byte[sampleRate * duration];
    for(int i = 0; i < soundData.length; i++) {
        double originalWave = Math.sin(2 * Math.PI * frequency * i / sampleRate);
        double harmonic1 = 0.5 * Math.sin(2 * Math.PI * 2 * frequency * i / sampleRate);
        double harmonic2 = 0.25 * Math.sin(2 * Math.PI * 4 * frequency * i / sampleRate);
        // Add all the waves
        byte sample = (byte)((originalWave + harmonic1 + harmonic2) * 255);
        soundData[i] = sample;
    }
    track = new AudioTrack(AudioManager.STREAM_MUSIC,
            sampleRate,
            AudioFormat.CHANNEL_OUT_DEFAULT,
            AudioFormat.ENCODING_PCM_8BIT, soundData.length,
            AudioTrack.MODE_STATIC);
    track.setVolume(1);
    track.write(soundData, offsetInBytes: 0, soundData.length);
    track.play();
}
private void endSoundWave(){
    track.stop();
}
```

## Engineering Standards and Design Practices

Google Play Store standards and recommendations
Java Code Conventions
Android Core app quality standards
.mp3 and .wav audio formats
IEEE terminology

## Testing

Originally, testing was to be performed with College of Music students to collect their feedback on the application's features. However, due to the university closure, testing was switched to be focused primarily on functional testing.

Functional testing was split into multiple stages. The project was split into several smaller modules, then each module was tested for functionality on its own. Next, the group performed integration testing.

We also implemented Mockito and JUnit tests.