Rithum Post-Mortem

Initial Goal:

We initially set out to create a multi-track recorder application. We intended to have four tracks on the screen at once, with the user only ever actively recording one track. However, all four would be able to play at the same time. A stretch goal was to be able to merge already created and recorded tracks, thereby virtually extending the four track recorder to an infinite track recorder. We intended to have a pull down menu that could control effects such as reverb and delay – as well as mute and solo settings. We also intended to have a left-side pull menu where the user could access the record, play and stop buttons.

Findings:

When we first began work, our first act was to set about simply getting one button on the screen that, when pressed, could record audio. We were actually able to accomplish this in a relatively short amount of time. Within just over a couple hours we had a very ugly little one-button app that recorded very poor quality audio. We couldn't name the audio file that was being saved or specify where to save it. The quality sounded like plugging a radio shack microphone into a commodore 64. Over the course of the next few work sessions we set about fleshing out the app and adding the additional functionality we desired and it wasn't too long before we realized that the Media Recorder library we were using for recording functionality wouldn't be capable of being utilized to create a multi-track recorder. This was probably our biggest set back of the entire semester. A concise explanation for this is that when MediaRecorder's recording function is active it isn't just storing the incoming audio in random access memory – it's simultaneously writing the data to disk (at least it will if you tell it to, and of course we were, because we wanted to permanently save recordings). Because it does this however, you cannot have two instances of MediaRecorder running at the same time. Thus, a multi-track app couldn't exist as long as we were using this library.

Revised Goal:

Investigating a solution to the multiple instance problem, we found ourselves looking at Android's AudioRecord library. MediaRecorder is the higher-level, fully fleshed out, has-helpful-methods-already-written cousin of AudioRecord. In fact, I don't know this for a fact, but I'd be more than surprised if MediaRecorder wasn't actually using AudioRecord down in it's lower depths to handle the raw digital byte stream. We dug deep and discovered that AudioRecord could handle our multiple instance problem. It gives very fine control of the audio to the programmer and with it, we could also accomplish one of our early goals for the app – drawing the screen a waveform visualizer for the recorded / playing audio. MediaRecorder wouldn't be capable of that either (at least to the best of our knowledge).

Findings:

For weeks we tried to come to grips with AudioRecord. It was a very complex task. Every step that was seamless with MediaRecorder was multiple classes worth of code for AudioRecord. Ultimately we did get recording functionality working. The sound quality was very good. We were pretty stoked. That sound quality alone re-lit the proverbial fire under our asses for days. Unfortunately, that was basically as far as we got. You see, **saving** the recording to disk is (unbeknownst to us prior) a monumental task. It's not as simple as taking what's been saved in memory and writing it – one must (for reasons I still don't fully comprehend) **create** (read: write; code) the audio file you intend to save it to / as. This means that in order to save it we'd need to learn how .wav files are coded and operate. How their headers are uniquely written and what that syntax is like. Not to mention what the code that would save it **to** that created file would even begin to look like. We managed to get the application recording the incoming audio into memory, and saving it as a header-less, raw audio PCM file. With this, we could play the audio back in-app, but nothing else. Again, **saving** what was recorded was the issue at play here. There are people out there that have solved these two problems. But they aren't numerous. And they aren't sharing (mostly, I'd imagine, because it's making them money).

The Pivot:

So we had a decision to make. We went over all of the details multiple times. Thought and re-thought various paths we could take to make this thing work and eventually settled on two different ways we could proceed:

a.) We could continue working with AudioRecord. With it we can make the app we set out to make, and even leave open the possibility of some really cool fine-grain-control level features such as our own sound effects and a waveform visualizer. The drawback to this is that, based on how things are progressing, and given that there's only a few weeks left in the semester, an honest prediction leaves us potentially showing up on presentation day with an app that consists of a single button that only records but doesn't save audio permanently. Even if we managed to get a little more than that, it wouldn't be much more. This approach leaves open the door for **anything** we wish to do in the future, but closes pretty tight the door that sees us presenting an app worth anyones time or consideration at the end of the semester.

b.) We could go back to MediaRecorder. With it we cannot make the app we set out to make. However, we can make a different kind of app. And we can actually make something that has multiple functionality and looks like something a lot of work went into. If we can come up with additional features it can have besides just one track recording, we can make something that, while not as fancy as our initial idea, could still actually be useful to people.

Not knowing what the future held. And knowing for sure from prior experience that whatever you think you can accomplish in a certain amount of time is probably about twice what you actually will – we chose option **b.)**.

Up To Present Day:

After making the pivot, we set about multiple tasks:

- Creating play/stop functionality (with corresponding buttons)
- Creating a new file directory on the users phone upon first launching the app that wouldn't be overwritten with each subsequent launch.
- Saving the recorded files in particular folders depending on the type of recording the user was conducting
- Launching a pop up menu upon application start that would ask the user to name the file and choose the type of recording (which would dictate where it would be saved)
- If it all possible, raise the quality of the recording. This would serve two purposes: the first is purely coincidental to the (potentially) cosmetic feature of saving the files in different directories depending on the "type" of recording: if the recordings weren't actually different qualities, then you're only cosmetically marking some kind of difference between audio files. If I say "this" file is a voice memo and "that" file is a music file but the files are the

same quality of audio, that's just a difference in organization. But if they **were** different qualities, it would give the different directory functionality it's raison d'etre.

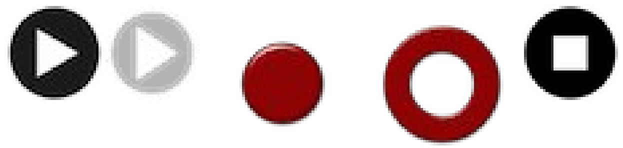
Fortunately for us, we put a lot of time and effort these last few weeks into accomplishing these goals, and got them all done. Roughly in the order you see above (except switch the last and second to last tasks). Some of the things that seemed straightforward at first glance proved to be somewhat problematic:

- An example would be configuring the logic of the app such that the exact right kind of toast pop up message would appear after the user pressed 'stop' after recording vs. when the user hits stop after playing.

While some of the things that seemed hard were done swiftly:

- Such as overwriting recordings in-app.

We decided fairly early that, even though we were now creating a single track recorder, and not a multi-track recorder that would obviously cater more to music makers, we'd keep the name Rithum. Naming things is hard and we couldn't come up with something else that fit – nor did we really feel like it.
Another thing we kept the same, at least until a week ago, was landscape mode. We initially had the app in landscape mode because it would be necessary to display four tracks at once (especially if each track was large enough to have a waveform visualizer).
The very first version of the app (including the version that utilized AudioRecord) had a simple white background and generic red record button. Once we made the pivot and started adding other buttons and functionality, we found the background we're currently using (which we probably won't stick with long term). Speaking of which: for most of this app's development, it's had the background it still retains, with these buttons:



Not all at once, mind you. What you see here are both the enabled and disabled versions of play and record.

Ultimately, near the end of development, we decided that portrait mode would be a much better fit, given the amount of things we would now have on screen. We also weren't crazy about the general visual aesthetic the app had. We wanted it to have a more unique, modern look. So I went in search for a more minimalist, modern approach to buttons, and slowly pieced together the current all-black setup you can now see in the final version.

The one task from the list above that we really wanted but really (really, really) didn't believe we'd be able to get was to raise the quality of the recordings. We spent one whole session on that problem, thinking going in that it wasn't possible, but just wanting to try on the off-chance. We first realized that the file type that we had always been saving in (.3gp) would always be less-than-optimal quality. So our first step was to set out to change the type. After trying some extensions (some of which worked, others which didn't) we settled upon MPEG-4, encoded with AAC. This **is** a lossless compression, but by raising the sample rate to 122,000 we were able to get incredible sound quality. We kept the quality for the voice memo option much lower (though we did change that to .mp4 as well) not because we love poor audio quality, but because the quality is plenty good enough for voice memos and most importantly, it has a very small footprint. Disk space on phone's is still at a relative premium, so we figured it'd be nice to have an option that would save files where quality didn't matter at a much smaller size – allowing for a greater number of tracks to be recorded. If, however, quality does matter, the user would now have that option as well.

Summary:

In conclusion I'll just say that John was a fantastic partner – it's crazy he and I didn't even set out to be partners, we just both needed one and I answered his class-wide email looking for one – because we're both musicians who loved the idea of making an audio app. We both put in roughly the same amount of time and effort on this thing, and I think that's because we both actually enjoyed this process. I've personally been saying it to everyone who's been asking me about this class: this has been the one class I've taken at UNO that has felt the most like real-world experience. I love that we were expected to put in work and show it, but that we could decide what we wanted to make and then set out to accomplish it. I also know from the limited industry experience that I have that in the real world you are almost always faced with challenges you didn't foresee, and you have to solve problems on the fly. This class felt like that. Most classes have assignments where the pitfalls are known ahead of time because it's already been done (and indeed the teacher expects it done a certain way). This experience was a very welcome departure from that kind of work structure. I enjoyed it.

-Chris Toups (w/ John Driggers)