Task 21: Turn it in
Task 22: Breathe...

GitHub Username: michaeloverman

Ms. Count

Description

Ms. Count is an odd-meter metronome, solving the problem of practicing music when the beats are not always constant, even, and regular. It contains a traditional metronome (click every beat, different sound on downbeats, measures of various lengths), and an odd-meter loop (click every beat, but beats can be set to different lengths). It also contains a programmable metronome, with access to a database of pieces with changing meters, which can be downloaded to the app. Any musician who plays music in odd-meters, or pieces that change meters frequently, has struggled to work on that music with their metronome - you either keep stopping to reset the metronome, or muddle through passages with the click in the 'wrong' place. That struggle has come to an end: Ms. Count is here to save you!

Intended User

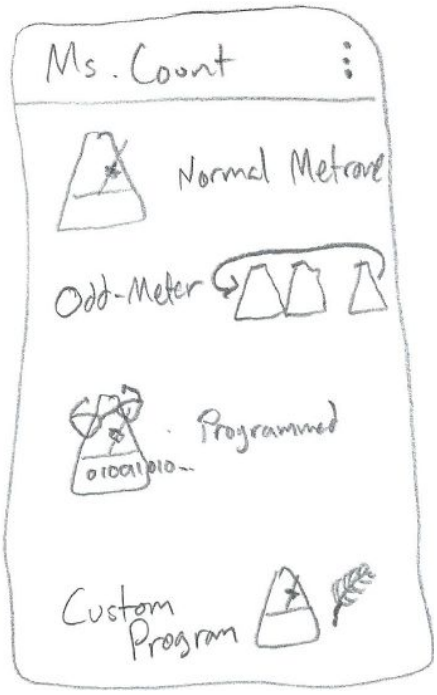
Musicians of all levels: professional, amateur, student, beginner. Anyone playing music that contains changing or irregular meters. **Teachers** of any of the above. Band and Choir Directors.

Features

- 'Normal' metronome
    - tempos between 15 and 400 beats per minute
    - precision to tenths of a beat, if desired
- 'Odd-meter' metronome
    - programmable, looping patterns, with beats of differing lengths
    - stores 'last pattern' used each time, ready for next use
    - consider small database to store 'favorite' patterns
- Preprogrammed metronome
    - access to database of preprogrammed works
    - instant database update
- Numerous, interchangeable click sound options
- Create your own programs, and store them locally, and/or submit to cloud database for other users to appreciate!
- Android Wear start/stop button

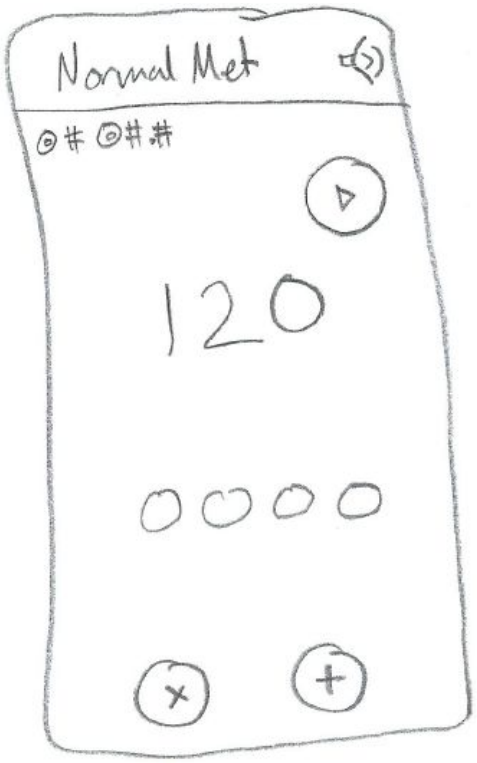
    - if user has a Wear device, app includes small wear application
    - a nice big button will start and stop the metronome
- Instructions
    - instructions for each metronome type
    - instructions for entering custom program data

User Interface Mocks



Screen 1

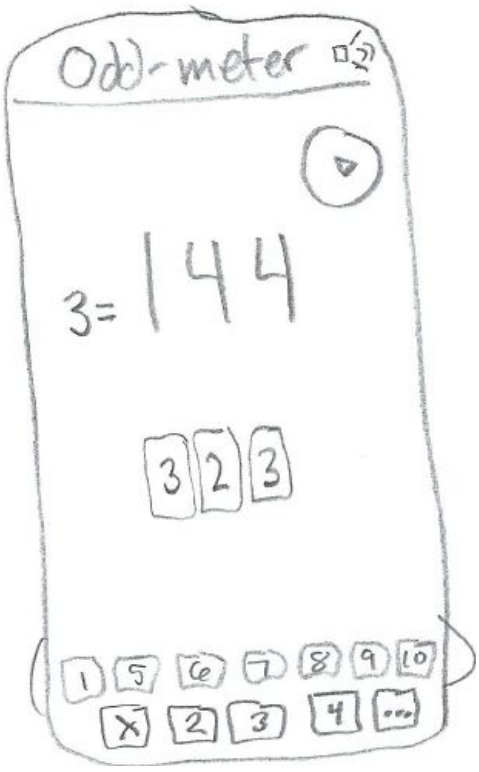
The “home” page: presents three metronome options, and the custom program creation option. Though initially these are simple text buttons, the “graphics” shown at right are more along the lines of what I would like to get in here.



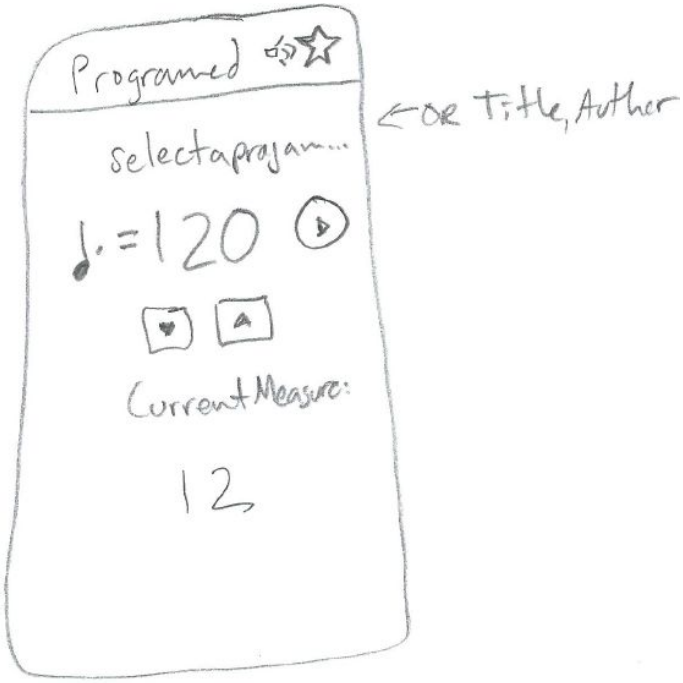
Screen 2

The normal metronome. Upper left corner, radio buttons, to toggle between integer and tenths on the metronome tempo. Upper right, FAB for starting and stopping the met. Center, “120” shows current metronome tempo. Swipe gestures change the tempo: side to side changes it by a little bit; up and down changes it a lot. Under the tempo view, small “buttons” for each subdivision. The number of these is controlled by the “+” and “x” FABS on the bottom. Subdivision buttons, when touched, control individual volume of each subdivision being clicked. Change colors accordingly. Menu options, upper right corner, include options for changing the click sounds available. (This is on each metronome.)

Screen 3



Odd-meter loop metronome. Click option menu item. Start/Stop FAB. Main tempo view, with swipe gesture control.
Beneath Tempo view is the loop, showing number of subdivisions per “beat”. Next to the tempo display is the beat-length which is getting the tempo marking.
Touching this view allows user to change its duration, using the subdivision buttons on the bottom of the screen.
Subdivision buttons: Bottom row is always present. “X” deletes last beat. Others add a beat of that length. “...” button brings up the top row of subdivision options, which disappears after use.

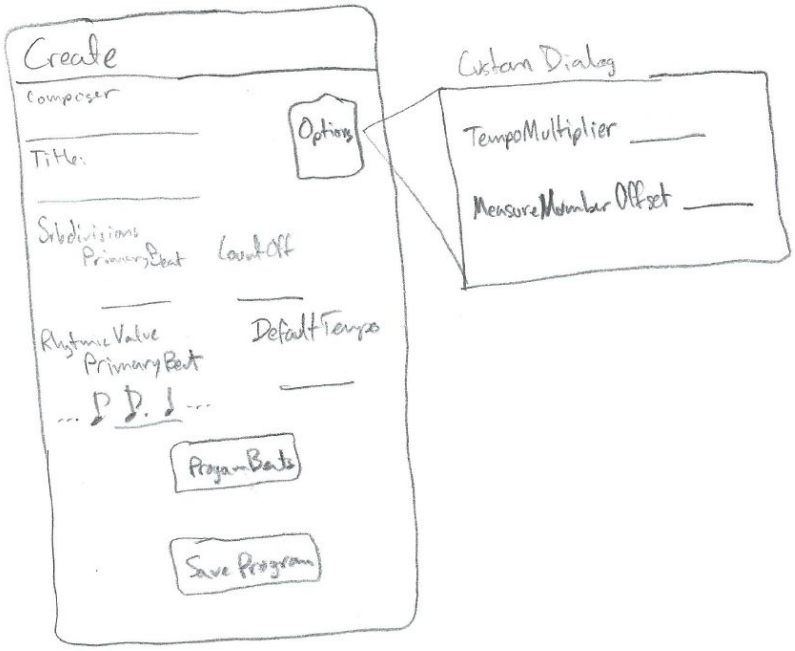


Screen 4

Preprogrammed Metronome: Menuitem, click change. Start/Stop FAB. Star or heart some place for marking program as a favorite. Favorited programs are listed in the widget.
Top TextView either says “select a program” or lists a composer and title. Touching the view opens a simple recycler view to access programs in the database.
Tempo display, as usual. This time it includes a note/rhythm image, showing what beat length receives the click.
Tempo changed with buttons - not enough room for swipe controls to be effective. (I don’t think.)
Current Measure shows what measure is being played. (Possibly…?! this will be ‘controllable’ with swipes, allowing a user to start in the middle of a program.)

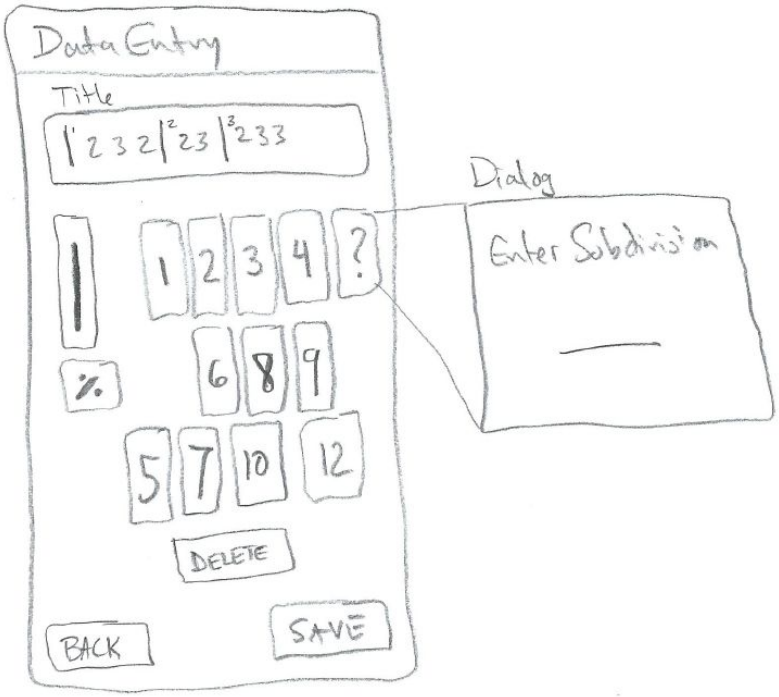
Screen 5

Custom program entry:
First screen allows for entry of "meta-data": Composer name, title, baseline subdivisions, primary rhythmic value, default tempo, etc. An "options" button opens a custom dialog box, to entry other options: tempo multiplier, measure number offset.
Two buttons: Program Beats button opens next screen, for entering the actual beat data. Save Program saves the program to the database.



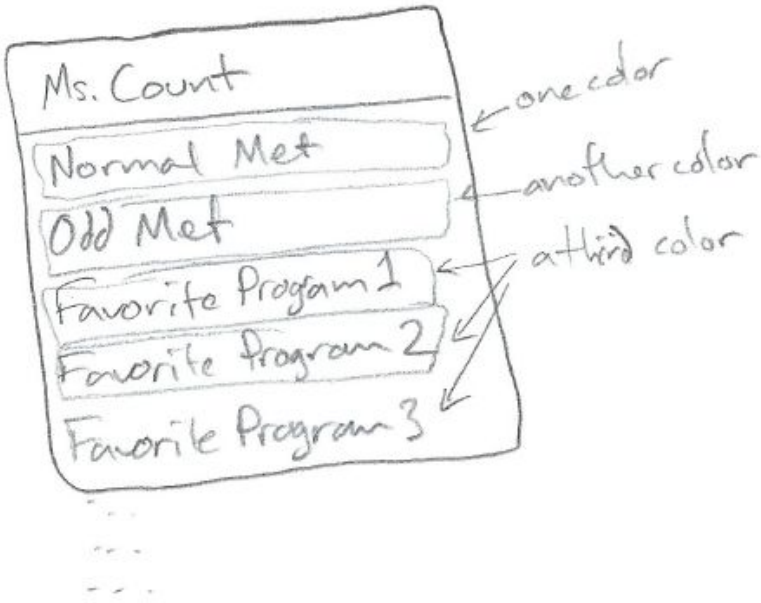
Screen 6

Screen where "real" data entry occurs. Top display area is a horizontally scrolling recycler view, showing the data as it is entered. Swiping allows user to see whole program.
Many buttons below, for entering data: numbers are beat lengths; long line is a barline; "funny" symbol is a one measure repeat, if the next measure is the same as the previous. Other buttons have self-explanatory names. Data is selectable, and can be deleted, or new data inserted. Numbers are out of order, because the bottom row are very rarely needed.
"?" button opens a dialog box allowing for entry of any other beat length, so anything is possible.



Screen 7

WIDGET: Widget will be vertically resizeable. It will be a list of “direct connections” to either the normal metronome, the odd-meter metronome, or a “favorited” program. Programs marked as favorites in the pre-programmed metronome will be included in the widget list, which can scroll to as many marked programs as one likes. If I can get it to list them in most recently used order, that would be best, but otherwise, alphabetical.



Key Considerations

How will your app handle data persistence?

Biggest data issue is the programs for the pre-programmed metronome. A Firebase database will be created specifically for the app, holding programs. SQLite will be used for local storage of custom programs (and maybe for offline use?), and for the list of favorited programs, for inclusion in the widget.
I must consider a search mechanism, for later, when database is, hopefully, considerably larger. But that will wait for now.

Describe any corner cases in the UX.

Metronome should STOP whenever user moves to a different activity/fragment.
From data entry fragment, app must ASK user, if data not saved prior to backing out.
UX must gracefully handle program selection, if not connected to internet for database access.
I must decide how to “permit” read/write (particularly write) access to database. Considering subscription or purchase model. But for now, free range with sign-in/authorization. Once database is big enough, maybe then start charging....?

Describe any libraries you’ll be using and share your reasoning for including them.

Butterknife - to simplify wiring up the UIs.
Timber for logging simplicity.
Android Debug Database - for debugging the SQLite database.
Various Firebase libraries (firebase-database, firebase-auth, firebase-ui-auth) for connecting to /using Firebase.

Describe how you will implement Google Play Services.

Authentication: to access the database of 'programs'
Wearable: will be used to create a "start/stop" button on a wearable device, so users don't need to have the mobile device right there. addApiIfAvailable(Wearable.API)
Admob: will place 'banner' ads across bottom of screen - hopefully not prohibitively annoying, just slightly so…
Analytics - probably, to keep track of what users access, but maybe not until later...

Next Steps: Required Tasks

Task 1: Project Setup

Configure known libraries and other dependencies (Butterknife, various Firebase apis, Google Apis)
Configure Firebase database, connect it to app.

Task 2: Metronome class (needed for "proof of concept"...)

- Create functioning metronome class, with basic, consistent, click
- Expand metronome to accommodate different clicks
- Expand metronome to accept 'programs' for changing click lengths
- Expand metronome to accept odd-meter 'loops'

Task 3: Implement UI for Each Activity and Fragment

- Build "front page" UI
- Build basic metronome UI
- Build programmed metronome UI
- Build custom program/data entry UI
- Build odd-meter looping metronome UI

Task 4: Basic Metronome

- Wire up basic metronome UI to Metronome
- Handle tempo changes (swipe gestures, buttons)
- Handle beat groupings (measures) with different clicks
- Handle changes in beat groupings

Task 5: Preprogrammed Metronome

- Wire up Programmed UI to Metronome, using dummy data
- Connect UI to database
- (Get a little bit of 'real' data in the database)
- Connect 'real' data to Metronome

Task 6: Odd-meter Metronome

- Wire up Odd-Meter UI to Metronome
- Arrange for 'storing' current settings
- (Consider a small database (SQLite) for storing 'favorite' patterns.) ((Nixed. Too easy to make a pattern - easier than taking the time to look up an old one…))

Task 6.5, 7.5, 8.5, etc: Get More Data

- Get more data in database, so there's more to show test-users

Task 7: Custom Program

- Wire up Data Entry UI
- Create logic for entering/editing data
- Create logic for converting from new/editable data to "Metronome form"
- Create method for sharing to Firebase

Task 8: Validate Data

- Validate all necessary data is present
- Validate accuracy of all data, number ranges, format, etc.

Task 9: Local Database

- Create SQLite database
- Connect programs to “favorite” star or heart on UI, and add to database
- Save custom programs

Task 10: Add new click sounds

- Create menu item within each metronome fragment for changing click sounds
- (Do this after I have new click sounds to work with…)

Task 11: Wear Start/Stop Button

- Configure UI for Wear
- Wire up UI with appropriate Messaging tasks
- Wire up App to receive Messages from Wear

Task 12: Add Instructions

- Create menu item within each metronome fragment for help pages
- Create help pages / instructions for each metronome/activity/fragment

Task 13: Transitions

- Once layouts/UIs are complete and functioning, create ‘slick’ transitions between fragments/activities

Task 14: Add Admob

- Configure for Admob
- Determine where to put ads (banner or interstitial page)
- Put ads in using test ads

Task 15: Test with Users

- Get app, in functioning form, in front of users
- Take good notes
- Convince users to contribute new programs to database

- Convince users to convince new users to download, use, and contribute further

Task 16: Favorites Widget

- Create Widget Layout/UI
- Connect widget to Favorites data

Task 17: Accessibility

- Find all the places I forgot to deal with accessibility issues as I was first creating things
- Find all the places where I need to tell it where to tab next to
- Find all the places where I forgot contentDescriptions

Task 18: Double Check Material Design

- Find all the places I didn't follow Material Design guidelines the first time
- Look for other places to add cool 'magical' elements

Task 19: Release on Google Play Beta

- Switch from test ads to real ads
- Release version in Google Play
- Tell everyone I know it is there
- Try to get people to try it, and then TALK to me, too!

Task 20: Address problems

- Figure out all the stuff I forgot
- Deal with all the places that cause crashes
- Find more places that crash/ mess up
- Fix those, too
- Find the littler bugs that don't crash, but cause problems of various kinds.
- Fix those

Task 21: Turn it in

- Reread instructions about exactly what format to submit
- Make it the right format to submit
- Submit

Task 22: Breathe...

- Have a drink
- Go to bed early

Submission Instructions

1. After you've completed all the sections, download this document as a PDF [File → Download as PDF]
2. Create a new GitHub repo for the capstone. Name it "**Capstone Project**"
3. Add this document to your repo. Make sure it's named "**Capstone_Stage1.pdf**"