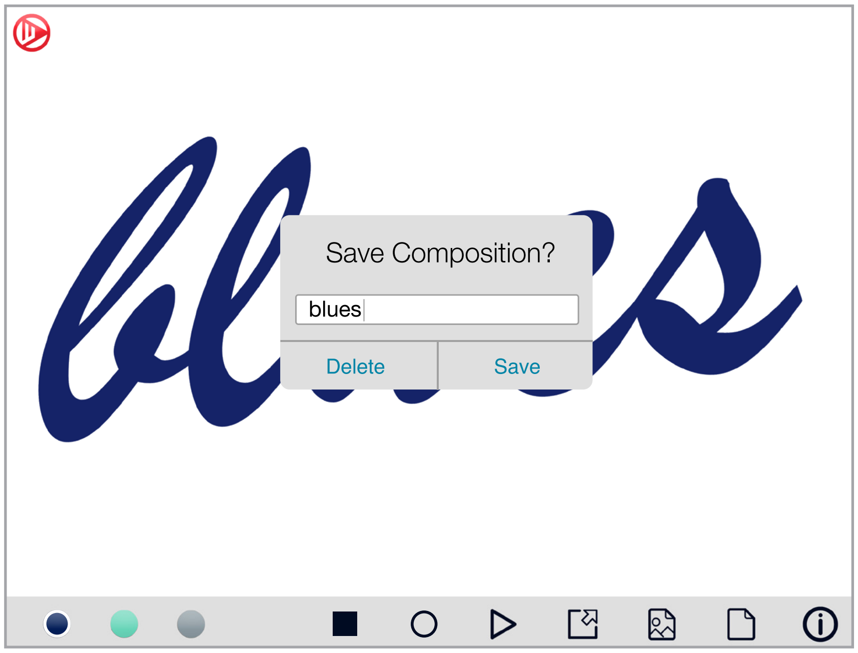
Initial proposal



•User can draw anywhere on their artboard

(The artboard contains an invisible, 16 point coordinate plane. This is how the user generates sound)

•User has 1 brush size (2 point radius) to draw with

•User is able to select from 3 brush colors

Selecting and Triggering The Sound Elements

•First, the user must select a color. Each color contains a unique chord progression, and the chords within that progression will "populate" the points of the 16 point coordinate plane on the user's artboard when they select that color.

•For example, the color TEAL contains the chords: D, B minor, G, and A. This means that when the user selects TEAL, (2,1) on the coordinate plane contains a B minor chord (1,1) contains a G chord, (0,1) contains a D chord, and so on.

•As the user draws a line across their artboard, they will pass over those points with their stylus, "triggering" the chord to play. (Like a trip-wire, or mine-field).

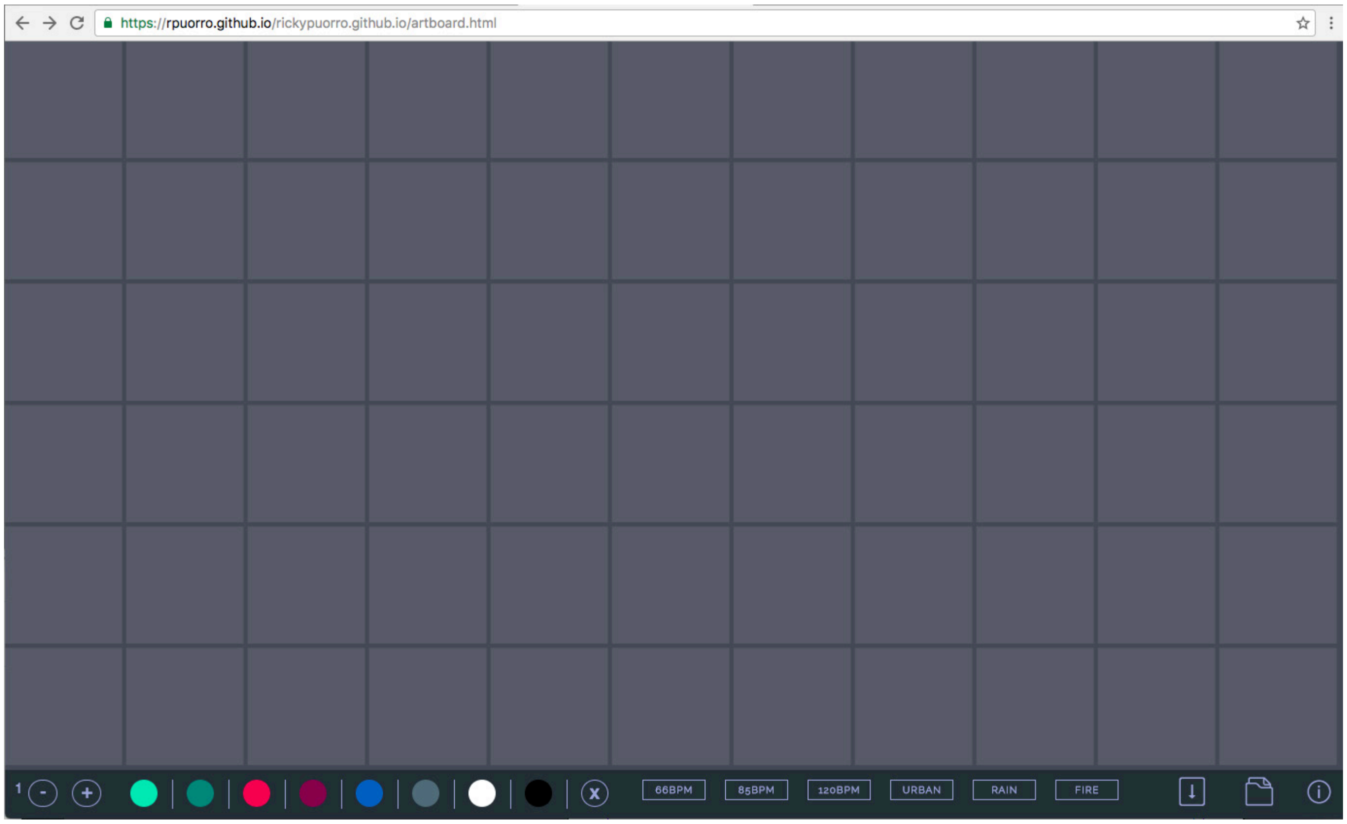
-User can RECORD composition
••

The screen which the user can draw on is called the ARTBOARD
The artboard is a coordinate plane with 16 POINTS

The user can select from;

˚ 1 brush size: 2 point radius
˚ 3 COLORS: teal(T), midnight blue(MB), light gray (LB)
 Each COLOR contains SOUND:
  • Each SOUND contains CHORDS
   • T contains the following chords: D, B min, G, A
   • MB contains the following chords: C#Dim, A, B minor, E minor
   • LG contains the following chords: A#dim, B

Capabilities upon initial release


https://rpuorro.github.io/rickypuorro.github.io/artboard.html
1
66BPM
85BPM
120BPM
URBAN
RAIN
FIRE

-User can select from 9 different brush sizes

-user can select from 8 different colors

-background color changes depending on color selected

-66 different sounds on artboard

-Sound loops play in background(6 to choose from)

-HOWEVER, sounds don't change depending on colors.

-user CANNOT record composition

PROCESS

When I initially proposed the idea for VectorOCtave, I was unaware of the capabilties of the programming languages I'd be using to make it happen. I set a rule for myself early on that I wouldn't "copy and plaste" code and would only utilize functions that I could make sense of. In a way, this was a bit limiting as far as functionality was concerned but seemed more honest.

Throughout the process, I was surprised by the things that should have been simple but were inceredibly complex, and the things that should have been incredibly complex that turned out to be simple. This, as I learned, is because language is not an intuitive process—for some reason I thought that through trial and error I could figure certain things out in these programming languages. It was like learning to speak cantonese by uttering random arrangements of syllables until I were understood. It was impractical. I needed to read the book.

Regarding the unexpectedly complex — when I came up with the idea for VectorOctave, I thought I'd be able to implement a "record screen" function easily, like what is in Quicktime Media Player. Not so much — it turns out that giving a web-based application access to record what's going on on a users computer screen compromises security, so it would't be as easy as say, saving the final drawing. There was a Google Chrome extension for their Chrome development environment "Chrome Canary" that would work, but I couldn't wrap my head around it, so opted to omit the function altogether. Instead of a screen recording of the whole composition, the user can save their final drawing.

However, a part of development that I thought would be much more complicated than it was was uploading and arranging sounds within artboard. As soon as I successfully implemented the function once, I could repeat it indefinitely. Throughout this however, I had to remember that when it DIDN'T work, it was either a "logical or syntactical" error.

Takeaway from the experience—

I'm surprised by how much I've managed to learn in just a few months of working on VectorOctave. Some of the big ones were

• The importance of correct spelling and punctuation: On many occassions, I spent hours researching why a particular function wasn't working, convinced that it couldn't be done, only to realize that I'd left the statement unterminated. It was a semicolon, or lack thereof, causing the whole operation to come to halt.

• Don't get too confident, check functionality frequently, and TAKE IT SLOW: Especially when implementing the sounds (all 66 of them) this was a problem. So for example, sounds 1 through 5 worked, so I copied the variables, event handlers, and did the next 5. However, I missed re-labeling one or two, and then had to go back and check each individually to find the "typo".

On a more general scale, I learned how to —

• design a website from scratch using HTML and CSS

• In-depth of understanding of HTML5 Canvas

• Implement basic Javascript functions, add event listeners, and call the function to act when triggered.

• Use APIs and external JS libraries such as CreateJS.

• USE THE INSPECTOR! A vital asset in the development process once I stopped the guessing game and started "talking to myself" via the console log.

What I would do Differently

If and when I re-write some of the application, I intend to pre-load the sounds and use much smaller sound files to speed up load time and functionality. Also, on the canvas I plan to use “shapes” as opposed to text as the elements which hold the “mousedown” event listeners that play sound when they are drawn over. I used text elements originally because I found documentation which explained how to create “rollover” text on the HTML5 Canvas. I could make sense of what was happening, and I knew that if, in Javascript, I could “trigger” the text to change appear-ance when I hit it with the cursor, I could “trigger” it to play a sound. Thus I used text, and for a font, used “webdings” which is are just shapes. The problem is, these text elements are not editable, and do not display on all browsers. In addition, if I used a shape, it would enable more visual customization.

Next Steps —

Implement “ontouch” functionality so the drawing application works on tablet. Ultimately, I feel that the kinetic aspect of drawing will enhance the experience.

Instead of text-based elements which “contain” the sounds, use shapes.

More options for brushes — make the lines that the user is creating more interesting; adding shadows, randomizing line width...etc.

More dynamic elements: changing sounds within artboard based on color selected, utilizing smaller sound files

Conclusion

Altogether I’m thrilled with the VectorOctave MVP, and will continue to build on it, as well as develop other applications based on the knowledge I’ve gained thus far in the HTML/CSS/Javascript development environment.