Xander Kaylan | Project 2 documentation

**How I met Requirements**

I have the required amount of controls (and a few more that go beyond what was done in the AV HW). I have four sliders (the volume slider which was given, a progress slider that I coded by hand instead of using HTML’s built-in one, a distortion slider, and a slider for adjusting the number that is multiplied by the audio data to manipulate the hue of the Bezier curve), seven checkboxes (three of which are different from what was given in the homework), and two sets of radio button groups. I also have the required canvas shapes (I have the cool Bezier curve rotated around the center, rectangles – the bars that use a random gradient – and two triangles), image manipulation functions, audio nodes (I added distortion and the slider the user can adjust) and displays – waveform and frequency, which the user can switch between – and the required text content. I also use many semantic HTML tags and my JS files follow the ES6 modules format.

**Above and Beyond**

In terms of above and beyond for audio, I coded progress by hand, made sure it worked with all of the audio tracks, didn’t affect the “play/pause” states of the play/pause button (for a while before I was able to fix it if the slider was paused and you changed it, the slider would make the audio play again) and stop playing if the audio track ended. I also added a way to upload your own file, and allowed the user to be able to switch between the track select options and the uploaded file, and have it remember the file that was uploaded, so that if you switched from upload to track select and then back to upload, it would start playing your uploaded song again instead of requiring you to upload another. I also made it so one mode was disabled if the other was active, so the app would not get confused. In terms of above and beyond in display, I added a way to adjust the number that is multiplied by the audio data for the Bezier curve hue value. Here is an example of what I mean, taken from line 156 of my visualizer.js file: ctx.strokeStyle = `hsl(${audioData[i] \* curveHueSlider.value},100%, 50%)`;, where the slider is declared in main.js and added to the call to setupCanvas as a parameter, where I set a script-level variable I declared in visualizer.js (the curveHueSlider variable above) equal to it. What this means is that you can adjust the hue value using the slider, but it will adjust how the audio affects the hue. I then made it so if you hide the curve by unchecking the curve checkbox, the slider becomes disabled, and the text on either side changes to 0.5 opacity, and un-disables and returns to 1.0 opacity if you re-check the curve checkbox. Finally, for my above and beyond I added a way for the user to change the triangle colors, as each was a single random color that only changed if you reloaded the page. The user can change the top one, the bottom one (which I made upside-down to create almost like a diamond) or change both at once. For this I also made it so when you choose to hide the triangles the buttons become disabled, the text above them also changes to 0.5 opacity, and they also return to normal if you re-check the triangle checkbox.

**What went right + what went wrong + if I had more time**

In terms of what went wrong, for a while the bars and the invert colors were way too flashy when you switched to waveform, which is why in my 2nd prototype I have that warning that says they won’t appear when you switch to it, but I wanted to figure out why they were too flashy and find a solution. Eventually I did, but only for the invert – I was setting the globalAlpha to 0.1 before all the shapes were drawn, which was making the inverted color mode really flashy. This was a simple fix, but for the bars I was unable to find an answer and so I just made them decrease in alpha when the waveform is active. I also did this with the curve, just to avoid hurting the eyes! Another thing that went wrong was the distortion and the slider to manipulate it – it took a very long time to get that working. There was one big issue where in main.js it wasn’t recognizing the distortAmt variable which I created, manipulated, and exported in audio.js. I was very confused, and it turned out that instead of exporting the variable and setting it equal to the slider value, I had to create and export a function in audio.js that took the distortion amount as a parameter and then set the distortAmt equal to it. Then in main.js I used the slider value as the parameter when I called the function. I thought it was done, but then a few days later I realized that yes my distortion was working, but it wasn’t actually affecting the canvas, and it turned out that I had the order in which my nodes were being connected (in audio.js) wrong. Once that was fixed, it worked!

In terms of what went right, a big one was the Bezier curve – it wasn’t that hard setting up the code needed for drawing the curve (getting the drawParam setup, creating the necessary variables, and getting the loop working), translating it, and rotating it, but I had to do a bit of trial and error for the bezierCurveTo() method call – which points should affected by the audio data and which shouldn’t. For each point in the method call, I tried adding + audioData[i] or -audioData[i]. After a surprisingly short amount of trial and error, I found a combination I liked that made each curve look almost like a half-moon, which when lined up (in the circle) made a cool 3D shape. There were so many ways to change up the curve using the audioData, and I’m surprised I found a good one so quickly!

Another thing that went surprisingly smoothly was setting up the progress slider, and having the current time and duration appear next to it. I thought that coding my own would be really hard, but it turned out not to be. There were two parts to it – the slider and the text next to it that displayed the current time as it went on and the end time. All it took for the text (after some research, see external sources section below) was the current time (and duration for the end time) % 60 for seconds, and the current time (and duration) / 60 for minutes. One issue with this however is that audio doesn’t do for example :05 if 5 seconds have passed, it’s just :5, so I had to hardcode some checks for if the current second is < 10, add a :0 before the current second, if the end second is < 10, add a :0 before the end second, if they both are, add a :0 in both places, and then a catch-all that displays the time with no 0 before the number if both are >= 10. Once I added these, it looked much better! As for the slider, it turned out that the value was just the current time divided by the duration, and then when you changed the slider, the current time was equal to the slider multiplied by the duration. There were a few more things to add, such as if the timer reaches the end, stop playing, only allow timer changes if the song is playing (so that pausing and then changing the slider doesn’t start the audio again), but overall this was a lot less than I thought it would take!

The app saves what audio file you uploaded, so you can continue playing the file you uploaded even if you go back and forth between modes, but the progress bar resets if you switch to the track select mode. If the user toggles upload again, they could just drag the progress bar to where they left off, but if I had more time, I’d make it a bit easier by adding a second progress bar for the uploaded file, that way however long someone was into it, the time elapsed (and along with it, the current canvas display) on the uploaded file would be saved, and they could resume it from the same point. Switching to the track select dropdown would cause the existing progress bar to be active, and the audio upload one to be disabled, and vice versa. Or I could have it so there is only one progress bar for both that somehow saves how far along you were, but I wouldn’t know how to do that. Another thing I was thinking about doing but ended up not having enough time for was adding a second file upload, that way the user can store some audio files on the page and freely switch between them. If I were to do this, I’d probably go with one progress bar that somehow saves multiple current times.

**External Sources used**

When coding my own progress bar, I wasn’t sure which math equation would get me the correct numbers for the audio’s current time and the audio’s duration (total time). I ended up finding the answer here: <https://mikeheavers.com/tutorials/format_html5_current_time_property/>, but I ended up only needing to do / 60 for minutes, rather than the full equation used in the linked tutorial, which was “parseInt((audio.currentTime / 60) % 60)”. Once I saw how it was done using current time (and realized I only needed the much simpler / 60 equation for minutes), it was easy to do the same thing for the duration property.

**Grade**

Finally, in terms of a grade, I think I deserve an 85-90. I have all the requirements, I made sure I didn’t have any deductions for missing anything – my HTML and CSS passed validation, I submitted the prototypes on time, I have the required controls for both audio and display, semantic HTML tags, text content, file structure, my app runs without errors, and JS module file requirements – and did a good amount of above and beyond. I also think my app is very easy to understand in terms of controls and instructions. Each control either has a clickable label (which toggles the control) or some text next to it that explains what that control does. I also added an HTML title attribute to the ones that required more explanation or some additional notes. However, I think there could be still more I can do in terms of possible controls, and I think there are some more canvas effects I could’ve added, too.