PUI Fall 2019
Brandon Fiksel

Assignment 8; Final Website Project: BAMPLE
Link: https://bfiksel.github.io/BAMPLE/index.html

Part 1

The purpose of my website, **BAMPLE**, is to provide an interface that *quickly* and *simply* lets users 'sample' any audio file. Specifically, this is great for the technique of micro-sampling, which takes tiny snippets to the point where the original source of the audio is unimportant. This is a common technique for many different types of electronic music production, especially realms like remixes, lo-fi hip-hop. indie electropop, etc. The website allows quick inspiration by randomly selecting samples from an audio clip, and it can also act as an online "sample pad" allowing users to play their samples back in a familiar 'pad' format.

The information that I hope to convey through my website is how users can load in an audio file, randomize samples from it, tweak the length of the samples, and save certain samples. Labels also illustrate that the sample pads can be played with the keyboard, in addition to mouse-click.

The website is certainly interesting and engaging, as it allows users to completely customize their experience— from the audio file they load in, to which samples they save, to how long they make their samples. The randomization element also helps keep it exciting and engaging—you never know what sounds are going to pop out. This also gives users a clear reason to come back to try new audio (whether it's songs they love or audio they recorded themselves)

The target audience is twofold: music producers (both amateur and professional) who utilize sampling in their music and want quick and easy inspiration on how to chop up their audio files for sampling; and more casual music-lovers who want to experiment on creating a 'remix' with one of their favorite songs or want a quick introduction to the possibilities of sampling. This website is useful both for masters and complete novices!

Part 2

How a user should interact with BAMPLE:

- All interactions are standard web page interactions, utilizing audio.
- Step 1: User is new to the site, so clicks the "?" icon in the top right. This opens a modal explaining how to use the website.
- Step 2: User presses the "Choose File" button and loads in an audio file from their own device (.mp3, .wav, etc)
- Step 3: User presses the "BAMPLE" button in the center of the page, or the "b" key. The button will delightfully bounce while active. This creates a random set of 4 samples from the audio file in the top 4 pads.
- Step 4: User can use the slider to change the length of the samples, ranging from very short (~15 milliseconds) to 1.5 seconds (1500 milliseconds). In general, I'd recommend making the sample length shorter, so that the original audio source is less recognizable, and the focus is more on the sound. This slider length changes the length of the 4 top sample pads.
- Step 5: User picks a sample that they like and drags it to one of the bottom 4 pads. This saves and locks the sample, so that it will stay even as they randomize the top 4 pads. The color of the pad, sample, and length of sample, get locked to that saved pad until it is overwritten.
- Step 6: User fills up all 4 bottom pads with saved samples, then continues to press the BAMPLE button to get new samples (shown clearly through the new random colors).
- Step 7: User uses the keyboard shortcuts to play the samples like a 'sample pad': top samples: "q", "w", "e", "r' / bottom 'saved' samples: "a", "s", "d", "f"

Part 3

External tools used:

- Web API: **Web Audio API**
    - Using this API was a no-brainer; I chose to use it because almost every JavaScript library out there for audio uses the Web Audio API. It also does not require any external assets/resources to be loaded in, it works automatically.
    - It adds the overall ability to play audio to my website.
- JS Library: **Howler.js**
    - I searched through many different libraries for working with audio in JavaScript, but in the end, Howler was the most reliable. I chose it because it had good reviews, but also because it was one of the few audio js libraries that had examples would reliably work in Google chrome.
    - With Howler, I was able to incorporate the "sprite" function, which takes a snippet of a larger song and stores it as a specified "region". It can take the starting point and length, in milliseconds, to create a 'sample'. This inherent 'sprite' functionality included in Howler was integral to my design experience, to

allow users to quickly and easily create new samples and adjust the sample length.

Part 4

Iteration from initial prototype:

I iterated a *lot* from the initial prototype and pivoted more than once. My first pivot was to remove the randomized word component and focus instead on segmenting out an audio track into various samples, which could then be easily played and managed in a sample-pad format. This was my direction for a long time, and I tried many different methods to get this to work (including struggling with many different JavaScript libraries, such as Wavesurfer.js to visualize the audio waveform, and mapping the visuals for Howler's audio-sprite functionality onto the waveform accurately). In the end, due to the many woes I faced with wavesurfer, I scrapped the visualized audio wave form section and made the website more about randomized samples— I loved this new direction, as it combined some of the 'computer-doing-the-work-for-me' elements of my first iteration with the random words, along with my love of 'micro sampling' and urge to make the sampling process as easy as possible for the user.

Part 5

Challenges faced during implementation:

The biggest challenges I faced during implementation was simply getting the audio functionality to work— Google Chrome has some inherent restrictions from a recent update that many audio plugins and examples did not take into account, and thus I could not even get many 'tried and true' examples to work reliably. The other big challenge was how to map the audio snippet's location on the waveform visual of the audio— this was so challenging that I ended up pivoting away from this approach in my final iteration. Once I had my final direction in place, a challenge I faced was ensuring that a saved sample pad could be overwritten by another sample pad— in the end, I had first delete the pad being overwritten, then make a clone of the top pad and append it to the bottom pad.