Interactive Computing / Museum Studies Final Project

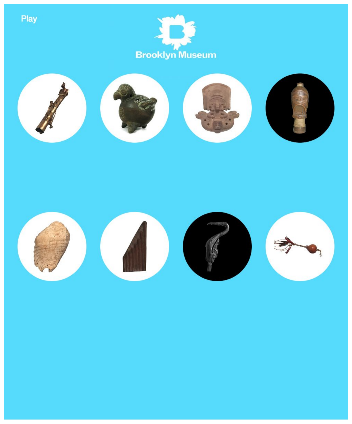
Musical Kiosk: Matthew Tessler (IC), Christian Owen (IC), Dylan Colby (IC), Hope Chen (MS)
Link: https://matthewtessler.com/interactive/icms/

General Instructions:

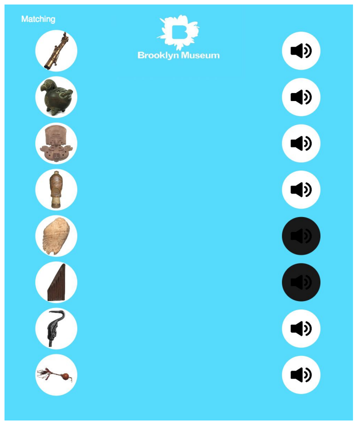
The prototype is divided up into three sections.

(1) Play the instrument sounds and look at their pictures.
(2) Matching game - match the sound to the instrument
(3) Music mixing - using the available sounds and some beats to make new music

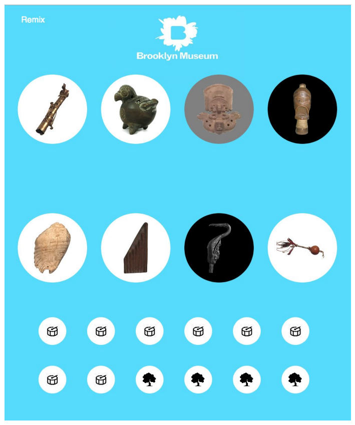
To move between the three states, use the left and right arrow keys.



State 1: Play



State 2: Matching



State 3: Remix

"Play" Instructions:

Each of the instruments has their own dedicated circular button with a picture. Blacked-out buttons are for instruments for which sounds could not be provided. To trigger an instrument's sound, simply click on the specific button to which the instrument corresponds. This will cause a recording of the instrument to start playing. To pause the recording, simply click the button again. Play and pause can be toggled back and forth. When the recording is done playing, it will stop. A instrument's button is gray –instead of white– when it is playing.

"Match" Instructions:

To function as a tool for learning, the "match" state is a challenge to match the instruments' sounds to the image of the actual instrument. This can be done based on memory or trial-and-error. To interact with this state, press each sound icon to play the sound of one of the instruments, which are randomized each load to create a new challenge. To match an instrument with the sound file, hover over an instrument icon and click to drag over to any sound file. If you have made a correct match, the instrument will replace the sound icon and turn green, preventing future dragging. You may still play an instrument's sound once you have matched it by clicking the icon.

"Mix" Instructions:

The "mix" state of the application is an augmentation of the "play" state. All of the instrument buttons function the same way as in the "play" state. Additionally, there is a set of remixable sounds under the instruments. The intention is, after the user is familiar enough with the sounds to match their images to its respective audio in the "match" state, the user can express themselves creatively. As a prototype of the application, basic, un-tempo-synced sound files can be played at the same time as the instrument audio. There are rhythmic remix layers, signified by a drum icon, and soundscape layers, signified by a tree icon. With further iterations, and custom-recorded audio, this application state could become incredibly robust, allowing for tempo locks, multiple beats and phrases, and perhaps even allowing the user to play fully-realized virtual instrument using a sample player.

General Technical Specifications:

This program is written completely in P5 (link), a JavaScript library based off of the Java-based Processing.

The program is separated into three stages that share code. The three stages were written by the three technical team members. Stage one was written by Matthew Tessler, stage two was written by Dylan Colby, and Stage three was written by Christian Owen.

The three stages are kept separate using what was learned in Interactive Computing about states. Each stage is a state, and they each have their own part an `if statement` in the `draw()` function.

Sounds and images are loaded into the program in order to play and display them. All of the functionality of the program relies on the sounds and images, and clicking the buttons triggers functions that manipulate the sounds.

The matching state was integrated into the code as state 1. Since the method of interaction is different in this state, two new classes were created. The first, "ImgS1" is for displaying the instrument images as icons, which listen for the "TouchStarted" and "TouchEnded" functions along with mouse distance to detect when to initiate a click and drag movement. The other class displays the sound files from a multidimensional array containing a randomized X and Y location and the ID of each file. If the "ImgS1" class detects that a specific icon is being dragged and "TouchEnded" is called, the state's function will check the X and Y distance from each "SoundS1" sound and determine which (if any) sound is being interacted with. By changing a universal variable to 'True,' the "SoundS1" class will then check if the sound and image ID's are the same. If they are, a new “ImgS1” will be created in that sound’s location, overlapping it so that the image and the instrument are aligned and can be clicked to play the sound.

The remix state is integrated as its own class, displayed in the “draw” function using its own “display” function for the “StateTwo” class. The “StateTwo” class uses both the “Instrument” class, and a new class, the “RemixButton” class. As with the other sound and image files, the remix sounds and associated images are loaded into arrays and displayed using nested for loops. The start and pause functionality of the remix buttons is implemented in a similar manner to the instrument buttons, calling respective class functions in the “touchStarted” function. Since the “RemixButton” class is a child class of the “StateTwo” class, a function is called on the “StateTwo” class in the “touchStarted” function to allow the remix sounds to be started and paused. This function in the “StateTwo” class respectively calls a function in the “RemixButton” child class.