# LiveArt

Saadiyah Husnoo & Alice Wu

LiveArt tackles the problem of disengagement in art by building an interactive and educational piece of technologically-integrated modern art. We combined the LeapMotion device's ability to track hand motions with a built-in computer microphone to make a system that allows users to create ecologically realistic and responsive art with voice and gesture commands. All animals and sounds are biologically accurate for their environment and species, which makes LiveArt a nice educational tool for children, in addition to being an interactive piece of art.

LiveArt currently provides users with a deciduous forest landscape and four native North American species that users can manipulate with 10 basic commands, each of which has several variations to allow for differences in natural human speech. Users can place animals on the screen, make animals speak/sing, have animals follow their finger, and remove animals at will.

The system works quite well for gestures, although we would have preferred to animate the individual animals, both when they are static on the canvas and upon user interaction.

Deployment: [https://alice_wu.scripts.mit.edu/LiveArt](https://alice_wu.scripts.mit.edu/LiveArt)
Code: [https://github.mit.edu/saadiyah/LiveArt](https://github.mit.edu/saadiyah/LiveArt)
Video demo: [https://www.youtube.com/watch?v=mRWDTj0sxaQ](https://www.youtube.com/watch?v=mRWDTj0sxaQ)

# 1 INTRODUCTION AND OVERVIEW

## 1.1 Our Motivation

Art has become sterile, inaccessible, and easily ignored by the viewer. It is not uncommon to see people walking by art exhibits in museums with barely a passing glance, or staring at pieces intently, gesturing and talking about or even to it, moving closer and closer, but then “BEEP!”

“Sir, please step away from the painting.”

It is clear that people want to engage with art, but art has become something distant and inaccessible, guarded behind velvet ropes and bulletproof glass, far away from its audience. Art is meant to deeply connect with its audience, to induce an emotional or physical response from its viewers, to provoke thought and incite conversation, to immerse people in a new experience... how is it supposed to do that in its current form?

## 1.2 Our Solution: LiveArt

Our solution to this problem is to bring more technology into the art experience by allowing viewers to interact with a piece of art and have the piece respond. In doing so, we hope LiveArt can increase engagement with art, invite the audience to spend more time interacting with art and allow the user to explore art in a safe, nonjudgmental, small-scale way.

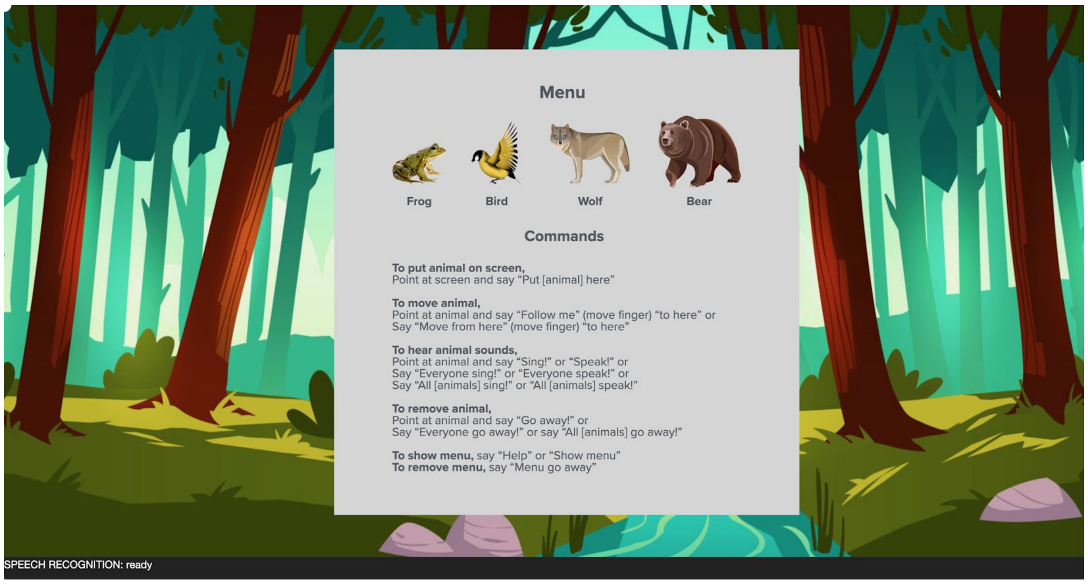
To make the application fun, especially for children, we focused on keeping the interactions intuitive while giving visual and auditory feedback to user commands and/or actions. We also tried to keep the aesthetic colorful and somewhat whimsical to attract smaller children, since art appreciation can be taught from a young age.

We also decided to put an educational twist on LiveArt by making our background environment, animal species, and animal vocalizations ecologically and biologically accurate. This way, both children and adults alike can learn while exercising their creative license through LiveArt.

# 2 System Description

LiveArt allows the viewer to make their own decisions in creating and interacting with the art. To start, LiveArt presents the viewer with a canvas showing a fun background, and a menu of options for various commands and gestures (see Figure 1).

LiveArt runs in the browser, takes in users’ hand motions with the LeapMotion controller, and captures speech with the computer’s microphone. We included a transcript of spoken speech at the bottom of the screen for accessibility and clarity reasons so the user can self-correct when the system does not respond as expected.



Menu

Commands

**To put animal on screen,**
Point at screen and say “Put [animal] here”

**To move animal,**
Point at animal and say “Follow me” (move finger) “to here” or
Say “Move from here” (move finger) “to here”

**To hear animal sounds,**
Point at animal and say “Sing!” or “Speak!” or
Say “Everyone sing!” or “Everyone speak!” or
Say “All [animals] sing!” or “All [animals] speak!”

**To remove animal,**
Point at animal and say “Go away!” or
Say “Everyone go away!” or say “All [animals] go away!”

**To show menu,** say “Help” or “Show menu”
**To remove menu,** say “Menu go away”

SPEECH RECOGNITION: ready

Figure 1: Welcome screen. The user first says “Menu go away” to begin interacting with the art.

## 2.1 The Interactions

The various interactions we programmed into the interface include:

1) Putting an new animal on the screen
   To trigger this functionality, the user uses the LeapMotion controller to point to a location on the screen, and issues the voice command “Put a [frog|bird|wolf|bear] here”, and if the location is valid, an animal will appear there.
   The animal’s graphic is randomly chosen among multiple images we ship with LiveArt.
   The validity of the placement hinges on whether the animal will overflow out of the frame, or if it will overlap with other existing animals.

2) Moving an animal on the screen
   To trigger this functionality the user points at an animal and issues a two-part voice command: “Move from here” | “Follow me” to pick up an animal, after which the user can move their hand to a valid location as described above, and then say “To here” | “Stop” to release the animal.

3) Making a specific animal sing or speak
   To trigger this functionality, the user points at an animal and says “Speak” | “Talk”. LiveArt will then play that animal’s sound.

4) Making groups of animals or all animals sing or speak
   For this functionality, only voice commands are needed. The user can say “Everyone [sing | speak]” to make all animals sing or “All [birds|frogs|bears|wolves] sing | speak” to trigger only animals of a certain species to emit sounds.

The sounds are themselves randomized within each species and started at different intervals so they won't completely drown each other out.

5) Deleting a specific animal from the screen
To trigger this functionality, the user points at an animal and says "Go away". The animal will then be removed from the art and not be able to trigger sounds as well.

6) Deleting animals by group
For this functionality, only voice commands are needed. The user can say "All [birds|frogs|bears|wolves] go away" to make all animals of a species disappear, or say "Everyone go away" to clear all animals from the canvas.

7) Summoning and removing the menu
LiveArt features a pop-up menu with all available commands and animal species. To show the menu the user can say "Help" or "Show menu", and to hide it, they can say "Menu go away".

Most of these commands contain some form of error checking. Some cases include:

- If the user attempts to issue a command that needs a target animal without pointing at it, the system will say "Please point at an animal" and display a corresponding transcript at the bottom of the screen
- If the user attempts to place or move an animal to a location off screen, the system will say "Please pick a spot on the screen" and again display a corresponding transcript

## 2.2 Feedback For Errors

In the above cases, we did not like the idea of silent failures, so we opted to give the user feedback with instructions on how to correct their errors. We considered two options - a visual pop-up and auditory feedback. We ended up combining the two by adding both auditory feedback and a visual transcript of all spoken speech. This gives us several benefits: (1) the user does not necessarily have to interrupt their flow to look away to read text and can instead listen to error messages, (2) if users are confused, they can double-check what the system said, or what the system thinks the user said, and (3) this makes our system accessible for hard-of-hearing users.

This decision was made in the context of our local setups and current intended use cases which are in relatively quiet environments. Art galleries, after all, tend to be very quiet, and artists generally exercise great control over their displays; LiveArt would likely require a separate soundproof room. However, in a noisy or open space, we would probably opt for text-only feedback instead.

## 3 System Design

Below is a diagram detailing the various components of LiveArt:

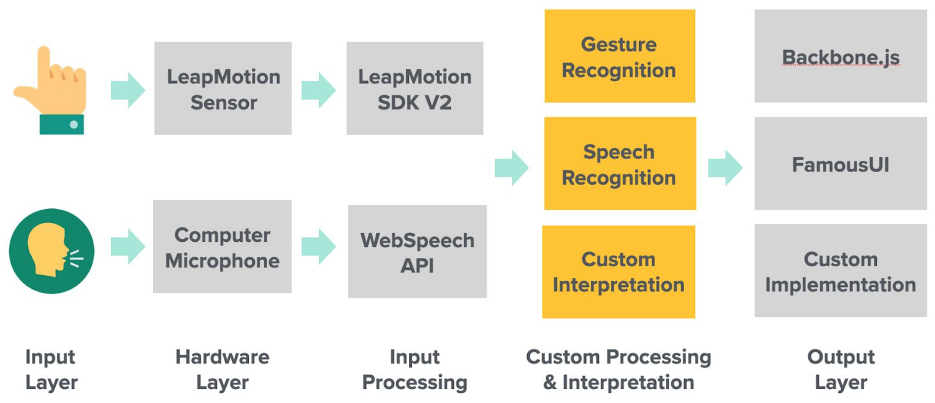


Figure 2: An overview of LiveArt architecture

LiveArt runs in the user's browser. When the user loads the page, our system grabs their screen dimensions and scales the canvas to make use of the most real estate. The user can then begin interacting with LiveArt. To do so, they move their hands over their LeapMotion Sensor, and the system translates their motions to a coordinate in the canvas. They are also able to speak through their computer microphone and the system uses WebSpeech API to process it to determine if the user issued a recognizable command.

Once recognized, the speech and hand movements are coupled with our custom logic to determine the appropriate action/ feedback to be produced on the screen. Each animal is a Backbone.js model and has its own custom implementations for commands that are specific to each animal species: how they look, what sounds they produce, their dimensions etc. We used a signal and slots mechanism to trigger sounds and appearance/movement/removal features such that a "signal" for the action is emitted. The intended signal recipients (which in this case are animal models), upon receiving the signal, can determine if they need to respond, and if they do, a "slot" function is called to do so. One example of such a response is playing a sound.

The images and vocalizations for an animal of a particular species is randomly determined upon creation. These are accounted for by a supervisor model that tracks all animals on the screen. This design came in handy especially when implementing more advanced interactive features such as determining overlap, highlighting on hover, deleting animals, etc.

### 3.1 Experimentation

Currently LiveArt is able to handle about 30 animals. To experiment with the performance of our system, we added 10 animals per category and triggered the various commands. We did not observe a noticeable delay in the processing speeds and the system was still quite available, though this gave us the idea of staggering the sounds start times and variety of sounds. We did however notice this was taxing on Chrome and so would not recommend more than 20 animals at a time. We also noticed hardware differences in LiveArt's performance in this regard - users who had older computer models with many applications running simultaneously sometimes experienced extreme system slowdowns well before the 30 animal mark.

The system's features all work well without bugs, but it suffers from occasional accuracy issues with gesture and speech recognition. The word "bear" in particular can be mistaken for a variety of other words - they're, their, there, Bayer, bare, etc. We experienced occasional finickiness when working with the LeapMotion, particularly if we were working under natural light, which makes sense since the Leap works with infrared.

We had close friends test out the system and noticed that the speech recognition has trouble with varying accents. We had two options: (1) rebuild the speech portions using a more suitable speech recognition package, or (2) add more flexibility to our existing speech recognition. We picked the latter due to time constraints, since we wanted to focus more on adding more interactions and addressing user feedback.

## 3.2 An Interesting Failure

An interesting failure cropped up when we added speech generation for feedback to users. One of our error messages was causing an infinite loop because it contained the keyword that triggered the failure in the first place. As a result, our system kept repeating retriggering the same failure and repeating the same failure message. This did not come to light until a bit late in the development process because the failure disappears quickly when the user corrects themselves, which we as developers did while testing the feature.

To be more specific, the feature involved the "speak" command, and the error message used to read "point at which animal you want to speak". As soon as the user pointed at a valid animal, the error message would stop, and the system would instead emit the appropriate animal noise. While developing this feature, we would immediately truncate the error by correcting ourselves as we assumed a user would, and thus this bug went unnoticed for some time.

To solve this, we first attempted to disable speech recognition while the system was speaking, but this got very dicey with multiple animal sounds, since they could vary considerably in length. Therefore, we instead edited the voice feedback to not include trigger words, but given more time we would have liked to create a better fix for this issue.

## 3.4 What Worked Easily

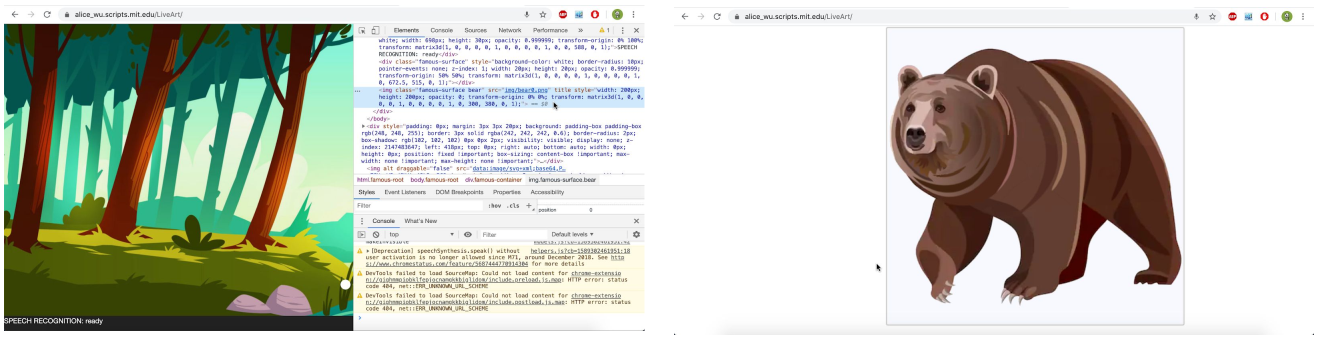
We were fortunate in that our development process went pretty smoothly. Once we had the animal models set up, implementing the signals and slots was seamless and we were thankful for having made a design choice early on that allowed us to add new features like sounds fairly easily.

## 3.3 What Was Difficult To Implement

We initially planned to animate our animals with gifs and sprites, but this turned out much harder than we originally anticipated. This was mostly because the package we picked to handle the images and interactions within the webpage, famo.us, does not support the use of gifs. Given that support for this library was discontinued and good documentation was hard to find, we eventually decided to code up an alternative in CSS instead, playing with the image sizes instead of adding true animations, which we thought was a somewhat acceptable alternative.

The “go away” feature was also a bit tricky to implement. While we could find ways to add items to the HTML, it was significantly harder to truly remove them. After attempting various things such as trying to find the HTML element and deleting it manually, we settled on moving the animal off the screen and disconnecting its slots so that in effect, the user cannot see or interact with the animal anymore.

We decided on doing so instead of hiding the animal in place under the background or changing the animal’s opacity to 0, since these could create unpleasant and confusing interactions for users who have image-related browser extensions such as Imagus, which will search for and pop out images on a web page when the user mouses over them, regardless of the opacity or relative position of the image, as illustrated in Figure 3.


alice_wu.scripts.mit.edu/LiveArt/
Elements Console Sources Network Performance
SPEECH RECOGNITION: ready
html.famous-root body.famous-root div.famous-container img.famous-surface.bear
Styles Event Listeners DOM Breakpoints Properties Accessibility
Filter :hov .cls
position 0
Console What's New
top
Filter
Default levels
alice_wu.scripts.mit.edu/LiveArt/

```
white; width: 698px; height: 30px; opacity: 0.999999; transform-origin: 0% 100%;
transform: matrix3d(1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 588, 0, 1);">SPEECH
RECOGNITION: ready</div>
<div class="famous-surface" style="background-color: white; border-radius: 10px;
pointer-events: none; z-index: 1; width: 20px; height: 20px; opacity: 0.999999;
transform-origin: 50% 50%; transform: matrix3d(1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
0, 672.5, 515, 0, 1);"></div>
<img class="famous-surface bear" src="img/bear0.png" title style="width: 200px;
height: 200px; opacity: 0; transform-origin: 0% 0%; transform: matrix3d(1, 0, 0,
0, 0, 1, 0, 0, 0, 0, 1, 0, 300, 380, 0, 1);"> == $0
</div>
</body>
<div style="padding: 0px; margin: 3px 3px 20px; background: padding-box padding-box
rgb(248, 248, 255); border: 3px solid rgba(242, 242, 242, 0.6); border-radius: 2px;
box-shadow: rgb(102, 102, 102) 0px 0px 2px; visibility: visible; display: none; z-
index: 2147483647; left: 418px; top: 0px; right: auto; bottom: auto; width: 0px;
height: 0px; position: fixed !important; box-sizing: content-box !important; max-
width: none !important; max-height: none !important;">…</div>
<img alt draggable="false" src="data:image/svg+xml;base64,P…
```

```
[Deprecation] speechSynthesis.speak() without    helpers.js?cb=1589302461951:18
user activation is no longer allowed since M71, around December 2018. See http
s://www.chromestatus.com/feature/5687444770914304 for more details
DevTools failed to load SourceMap: Could not load content for chrome-extensio
n://gighmmpiobklfepjocnamgkkbiglidom/include.preload.js.map: HTTP error: status
code 404, net::ERR_UNKNOWN_URL_SCHEME
DevTools failed to load SourceMap: Could not load content for chrome-extensio
n://gighmmpiobklfepjocnamgkkbiglidom/include.postload.js.map: HTTP error: status
code 404, net::ERR_UNKNOWN_URL_SCHEME
```

Figure 3: (Left) This empty-looking landscape contains a bear with opacity 0. (Right) When moused over, it still pops out a bear image due to the user’s Imagus browser extension.

# 4 Design & Implementation Tweaks

LiveArt was built in multiple phases while we slowly added more functionality, the ordering was roughly as follows:

- Create UI and add basic features to create animals via pointing and speech
- Add movement interactions (“follow me” command)
- Add sound playing capability
- “Animating” animal images
- Add error feedback
- Improve UI, add variety in animal species, images and sounds

That being said, the process was not very linear, and new features downstream made us question and rework features we had already built, especially the logic behind them, though those changes might not have been obvious to our users. While we had a pretty clear idea of what we wanted LiveArt to look like and be able to do, we had to make some tweaks along the way. These were in response to feedback we received from our users and during the various checkpoints, as well as obstacles we ran into along the way.

## 4.1 Animation

One of the features we had in mind was to animate the various animals in LiveArt, showing them flapping their wings, hopping up and down, licking their chops, etc. We originally intended to handle the animation using sprite sheets and CSS, but this turned out to be very hard to implement using famo.us and Backbone.js since we could only display static images. After an unsuccessful attempt at modifying our libraries to support gifs and receiving feedback saying that animations might feel out of place and unnecessary, we pivoted and settled on a compromise; we now “animate” the static images with pulsating size changes so the animals stand out against the static background and feel somewhat alive.

## 4.2 Visual Feedback on Hover

The LeapMotion controls a fairly small pointer for selecting animals. Unfortunately, this yielded a precision problem - users could get quite close to an animal, but still end up missing and getting the error message “please point at an animal”. This frustrated them as they tried to figure out what the issue was. This gave us the idea of highlighting the target animals to provide better feedback to the user. We decided on a yellow border since it popped against the background as shown here with the bear.

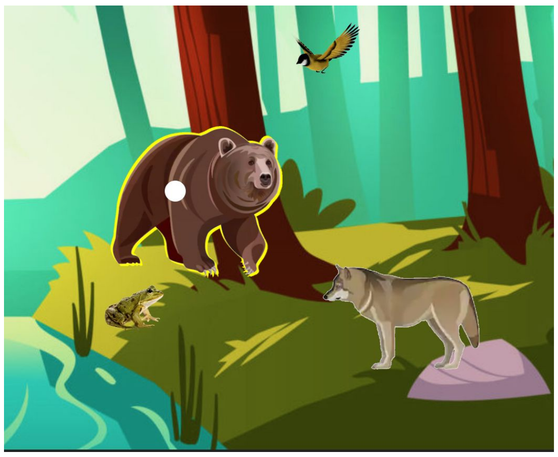


Figure 4: Yellow border on hover

With this visual cue, our test users were better able to calibrate and correct their interactions, and they later expressed great appreciation for this feature.

### 4.3 Error Messaging

At first, we didn't have any error reporting, which meant that the only feedback users received was their commands not working immediately. We received feedback during user tests that this was extremely frustrating and confusing, which caused us to add error messages for users. We chose to add only auditory messages at first because we thought this would disrupt users' workflow less, as they wouldn't have to look away from to read an error message, but we later received a request for written transcripts for accessibility reasons. Separately, another user suggested that we expose the system's interpretation of user speech to users, so users could see if there were any speech recognition glitches. We ended up combining these two requests and displaying the system's speech recognition results as a transcript at the bottom of the screen, which served both purposes.

## 5 User Study

We had close friends try out the system and gained the following insights:

- Users wanted an instruction menu, which we implemented.
- Users wanted to use all the available screen space, so we got rid of the dead space.
- Users seemed frustrated when they could not figure out why the system was not responding as they wished. To address this we added error messaging and the yellow hover border.

We learned that users often do not use the system as intended, so we as developers must give them the tools to do so, here via a menu. We also learned that small tweaks can have a large impact on the usability of a system, for example the hover border and error feedback.

## 6 Performance

We learned through our iterative design process that having a solid initial design really dictates the overall direction of the project. It made adding incremental features much easier. We were both pretty new to JavaScript and CSS before taking this class, and through this project, we learned some interesting ways to program asynchronously. We also learned to ask what truly is important about particular interactions, and how to expose features to users in a way that would be useful and intuitive. We resisted the urge to clutter our implementation with too many features, which in itself was a humbling lesson in simplicity, usability, and accessibility when developing for all ages.

In terms of features we wish we could have added; we would have liked to allow users to pick their own backgrounds and interact with animals in more ways, perhaps tickling/poking the animals and watching them squirm or jump away, but this feature is currently outside our grasp.

We also would like to improve upon our deletion hack and implement true animations. Given more time, we would like to pivot away from using famo.us and explore alternative packages with more animation features. We believe that having nice smooth animations would help increase engagement and give the user more ways to express their artistic ability.

## 7 Tools, Packages & Libraries

The packages we used are as follow:

- Leap motion SDK V2 in JavaScript (found here): This allowed us to use the LeapMotion sensor, so we did not have much choice here, though we did make the conscious decision to use the JavaScript offering since our app runs in the browser.
- Google WebSpeech API (found here): This package allowed us to parse and generate speech. The speech parsing is decent but would occasionally confuse similar sounding words. Therefore, we had to add flexibility for those words based on observed failures.
- Backbone.js (found here): This allowed us to define the animals as instances of different models with their own custom sounds and images, giving us more structure than plain JavaScript. We also leveraged the events system to trigger asynchronous sounds.
- Famo.us (found here): We used this API to drive the mechanism for adding and moving images. We realised a bit too late that this API was no longer being updated or used. Documentation was virtually nonexistent. We did not find a way to animate images with it, and could not integrate sprites etc. Instead we used CSS properties to change the sizes of images to make them pulsate. Given more time, we would have used an alternative; for example, we could have used CSS sprites directly or packages such as javascript.info (found here).

## 8 Logistics

LiveArt is a two-person team. Saadiyah got a headstart in designing and implementing the project structure with underlying models, interactions (add, follow me, sing), hover border, animations and randomized animal  sounds. Alice added the menu, deletion feature, variety of animal species, randomized images, and created the final images and sounds. We were both involved in the initial design process. The various videos were drafted together and Alice recorded and edited them. The final paper was also written collaboratively.

When it came to testing, we took turns testing what the other had built and improving upon them. Where possible, we involved close friends and family members as testers. In short, we evenly distributed all work.

We really enjoyed working on LiveArt and learning more about multimodal user interactions while implementing this project.