Josh Lieberman
COMS W4170 – User Interface Design Steven Feiner
Assignment 3
Swiper - A Gesture-Controlled Video Player
Documentation

Note: All images and notebook scans can be found as high-resolution images files in the Images/ directory.

Initial Design Decisions - Sketches

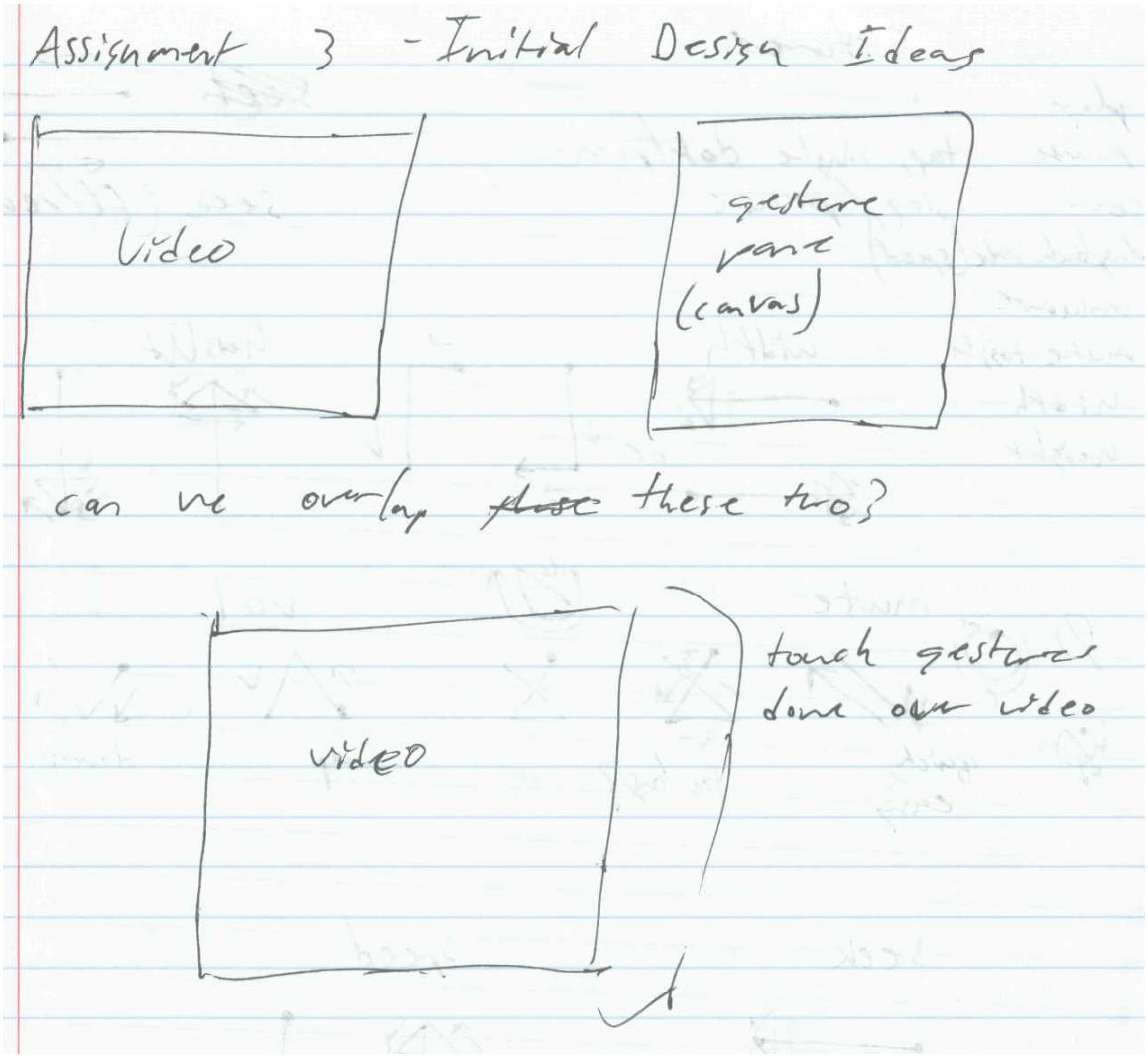

Figure 1: Initial Layout Sketches

In my initial brainstorming session, I had two layouts in mind. One of which had the video player and the gesture canvas separate entities on a web page, while the other had the canvas layered on top of the video player. I ultimately decided to use the

layered design. From a usability perspective, it seemed like having a second object on the page would detract the user from the video, which is the primary experience in our app. Requiring the user to disengage with the video in order to interact with an element that controlled the video seemed counter-productive, and would result in a negative user experience.

The next design decisions I worked on were creating user alerts. I knew there were some situations where the application would need to alert the user of the status of a specific video attribute, such as video volume or playbackRate levels.

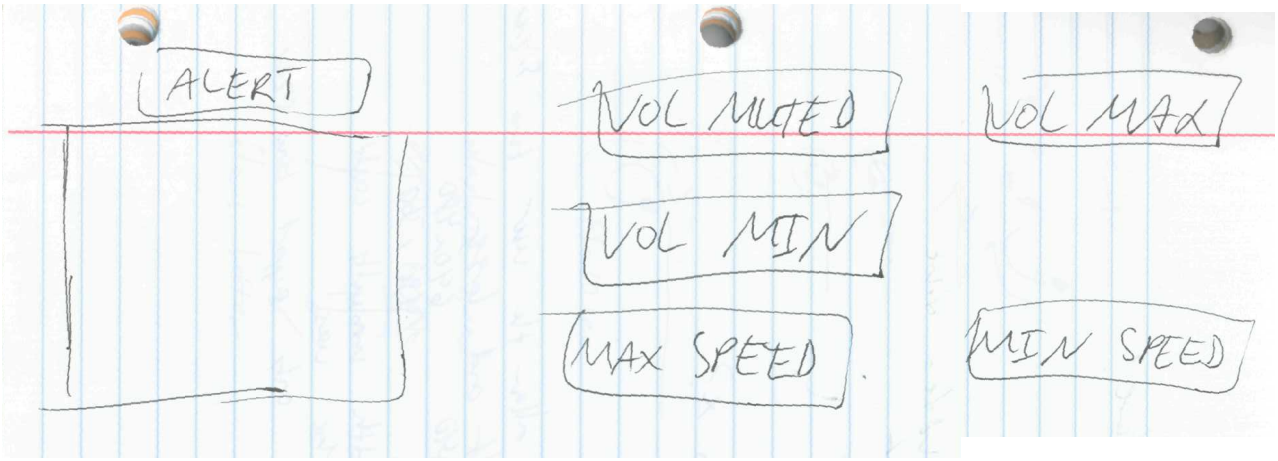

Figure 2: Initial Video Alert Sketches

I noticed that the canvas on the $1 recognizer page had the ability output text to the user, and decided this output device that came would the canvas would be perfect for my needs.

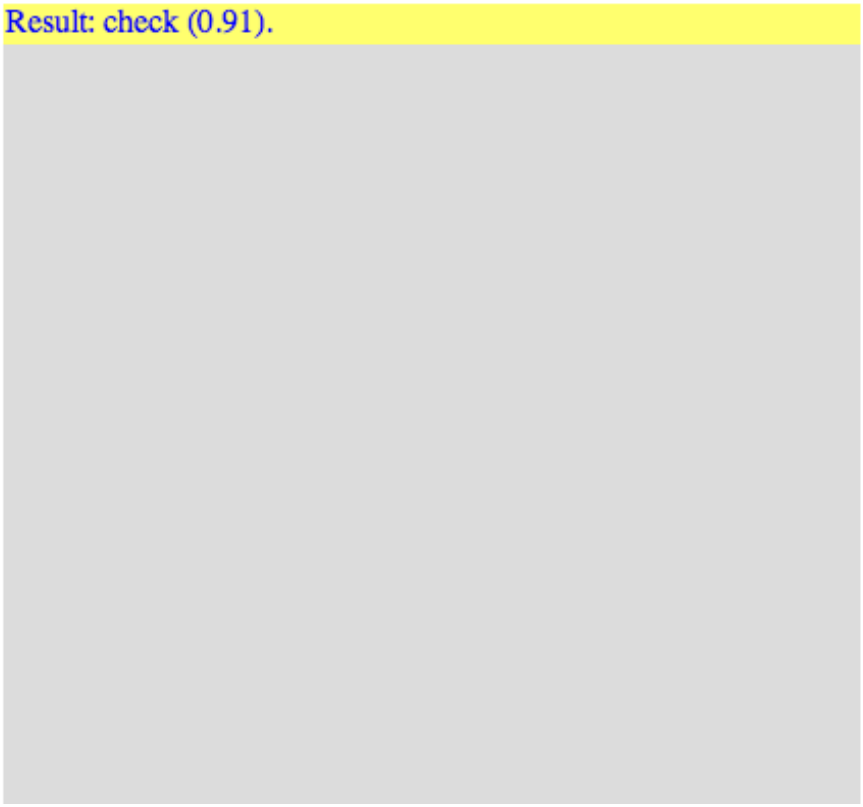

Figure 3: The $1 Recognizer Canvas

Using this built-in text output device had benefits. I could recycle pre-existing code, and not have to waste time writing a different way to alert the user from scratch. The canvas was also customizable, and I could change the colors of the text and backgrounds to fit my design.

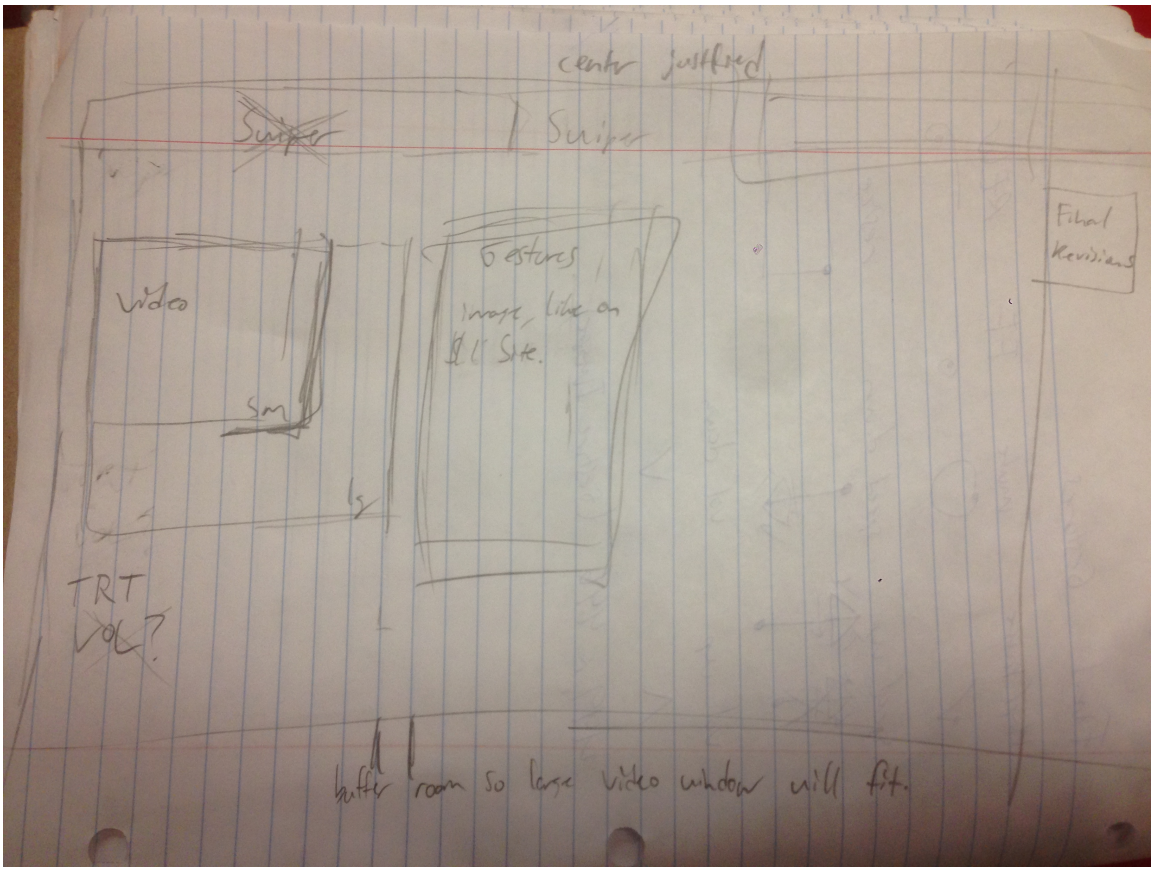

Figure 5: Final Revisions to my Lo-Fi Layout Design

Figure 5 shows my final lo-fi design revisions. I decided I would have the video player left justified on the page, with a gesture/help image next to the player. I left enough room between the player and the gesture help image so that when the video player size was increased, my layout would stay static, and the help image would not be moved further to the right on the page. I also decided I wanted to allow the user to see the total running time as well as the current time of the video. This information is presented in popular video viewing applications such as YouTube, or VLC, and QuickTime. I thought it would be in good UI practice to include it in my design as well.

I decided not to include a volume level indicator. From a selection of 10 random computer science students I polled, 8 preferred to use as computer or device system's internal volume control rather than an addition volume control setting in an application. A specific student said that having multiple controls confused them, and they usually avoid controlling the volume on an application-by-application basis. Note that my application does include proper volume controls for the video, but I didn't feel as if this was an important enough value to include outside of the video player.

Initial Gesture design decisions
Sketches

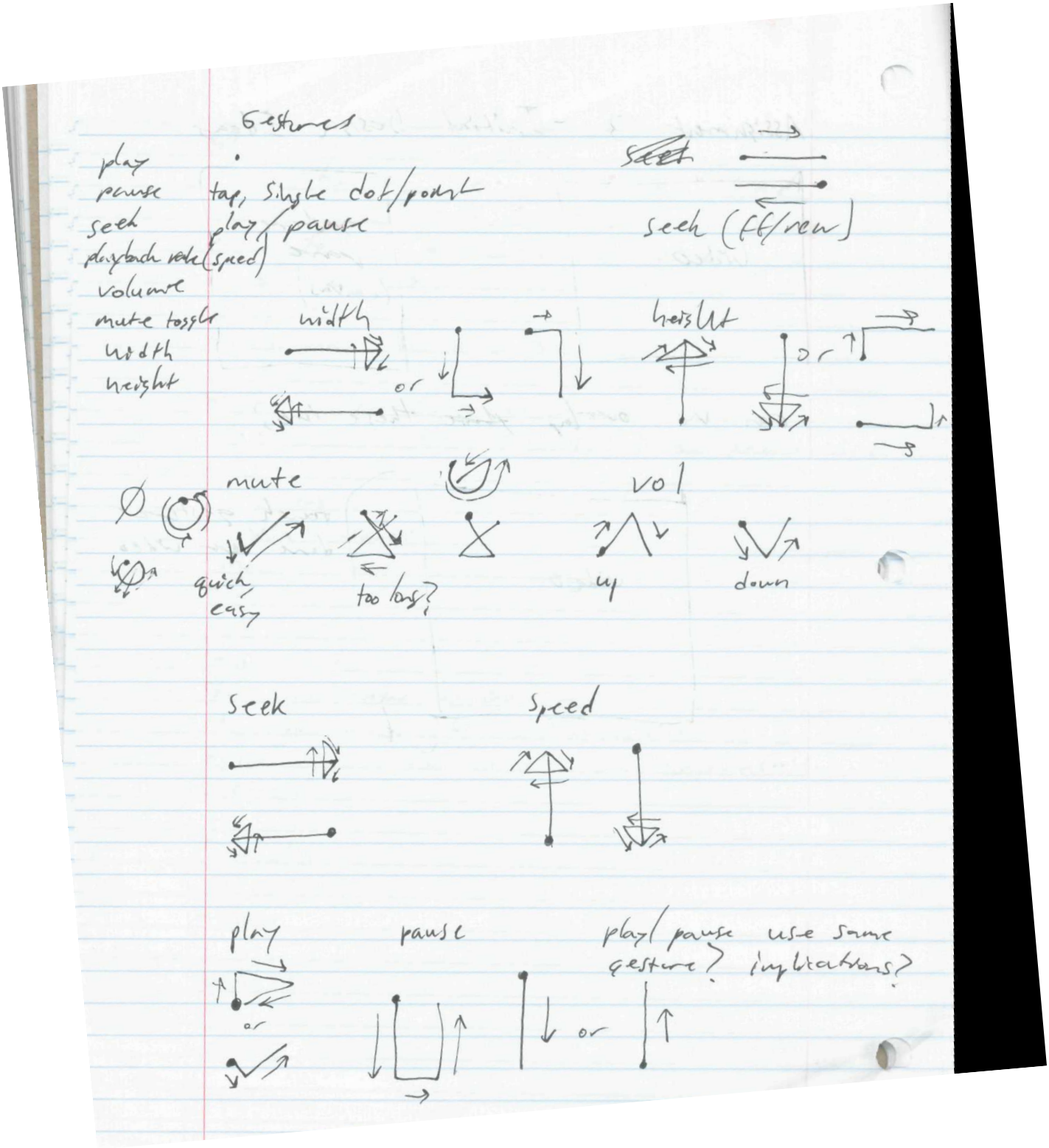

Figure 5: Initial Gesture Sketches

Figure 5 shows the results of my initial gesture brainstorming design session. Initially, for the play/pause gesture, I wanted to use a single point or tap on the canvas. I also considered a horizontal triangle, the universal symbol for play, as seen at the bottom of the page. From a usability standpoint, I wanted play and pause to use the same gesture, so I ended up scrapping the "play button" idea for an improved gesture in my second set of gesture revisions.

For the seek attribute, I knew I wanted to have rewind and fast-forward gestures. My initial thought was to use a single, horizontal line moving either left or right for rewind and fast-forward respectively. These gestures would be further revised in my second and final set of gesture revisions.

For adjusting the width, I considered horizontal arrows pointing left and right, as well as L-bracket-shaped gestures to simulate shrinking or enlarging the sides of a video window. For the height, I chose equal and opposite gestures. The video sizing gestures would be revised and changed further before the final gesture was decided on.

For the set of gestures controlling the speed of the video (the playbackRate), I chose upward and downward pointing arrows. I needed to find a gesture that would directly relate to the increasing and decreasing of some value (in this case, video speed), and thought arrows were a good choice. These arrows were the gestures I ended up using in the final version of my application.

For the volume controls, I decided to choose the pre-defined "v" and "caret" symbols. Because I decided to use upward and downward arrows to control the speed, I needed another set of gestures that were similar to arrows, which had the connotation of increasing or decreasing a value. I felt that the "v" and "caret" were great choices, and ended up being used in the final product.

For mute, I played around with a few different gesture ideas. The first was a check mark. I knew that a check mark was a simple, quick, and easy gesture to recognize and execute, but was unsure if the connotation was appropriate for the mute action. It seemed like there were better choices. I also tried the "delete" gesture, but feared it was too complex and long for an average user to execute. I also had preliminary consideration for a circle gesture, but did not consider that again until the last round of gesture revisions.

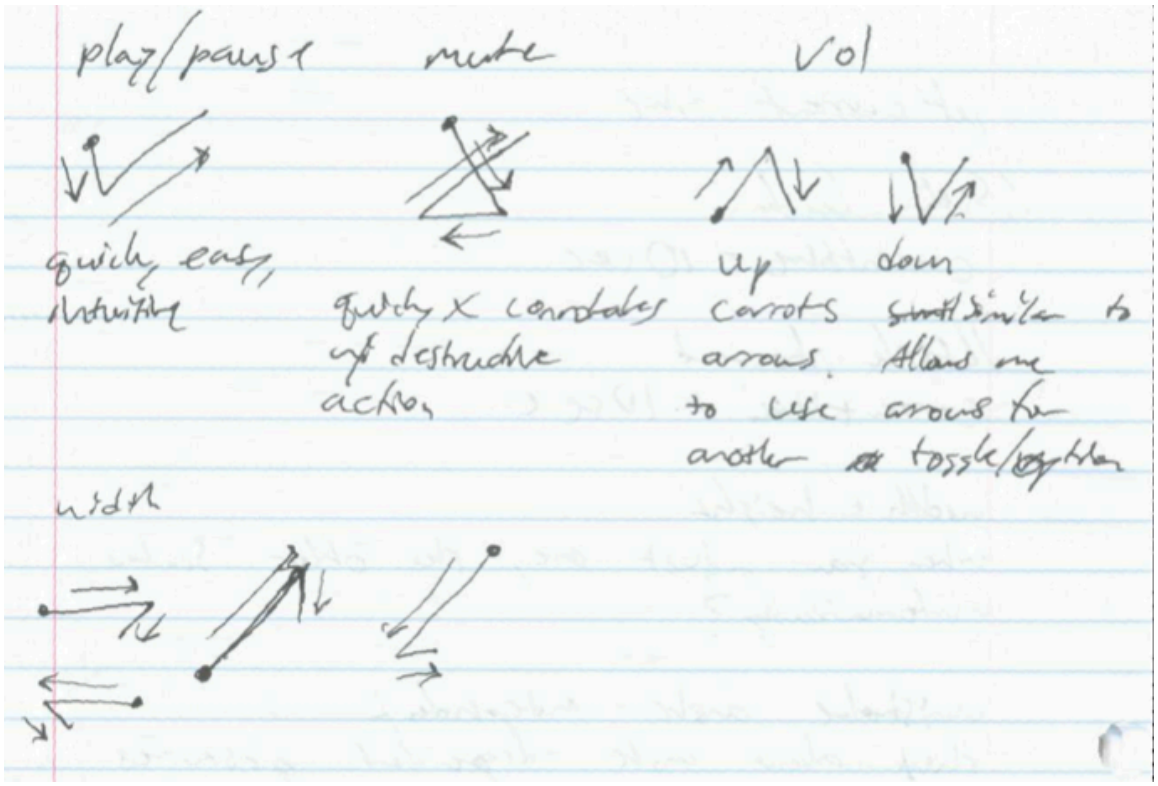

Figure 6: Second Set of Gesture Revisions

In my second set of gesture revisions, I changed the play/pause, mute, and width gestures. For play/pause, I felt that the check mark was a better fit. A check mark is synonymous with some affirmative action, and would be more appropriate for a one of the most basic, fundamental, and important actions of a video player, starting and stopping the video, rather than using it to mute or unmute the video volume.

The mute/unmute gesture was changed to the "delete" gesture. I felt that the connotation of an X (similar to the "delete" gesture) would be most meaningful for a destructive action, such as muting the volume. I would eventually change this gesture again for the final product.

I was still struggling with the width gesture. I didn't know the best way to create a gesture that a user would immediately recognize as one used to expand or enlarge the video player. I would eventually change and limit the way I let the user resize the video player.

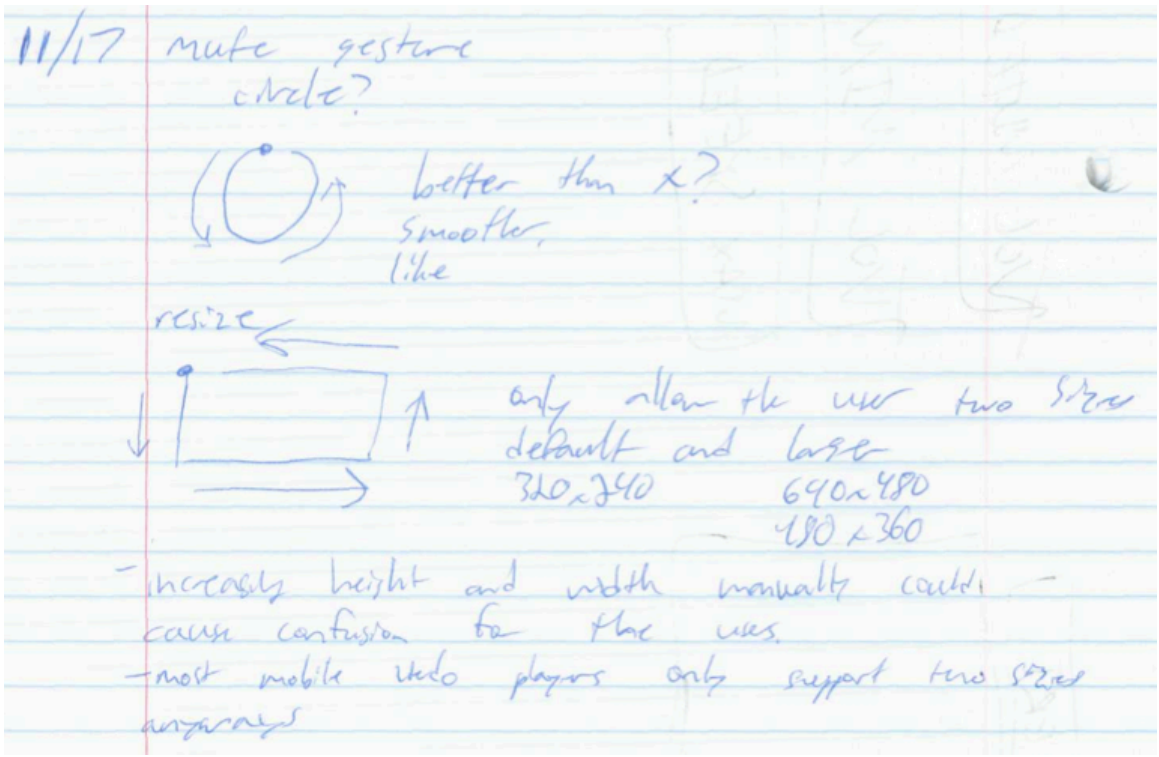

Figure 7: Final Mute and Video Resize Gesture Revisions

Figure 7 shows the final revision for the mute/unmute and resizing gestures. I decided to use the circle gesture for mute and unmute. The idea was that the user would feel like they are turning up or down a volume knob on a stereo. I thought that equating a real-life example to a gesture would make the user more comfortable with it.

For adjusting the width and height of the video, I finally finalized on a decision. Rather than giving the user the ability to control the width and height attributes, I decided my application would offer two video sizes: a small-sized player, and a larger player. Many popular video applications such as YouTube, or Apple's video player on iOS devices restrict the user to two views, a smaller player or a larger player. Requiring the user to adjust the width and height of a video manually might not allow for the best user experience possible. I thought that a rectangle gesture would be the easiest way to communicate to the user that my application had two video views, small or large. In addition, I performed some basic tests on an initial prototype of my application, and my test users agreed that this would be the easiest way to resize a video window in my application.

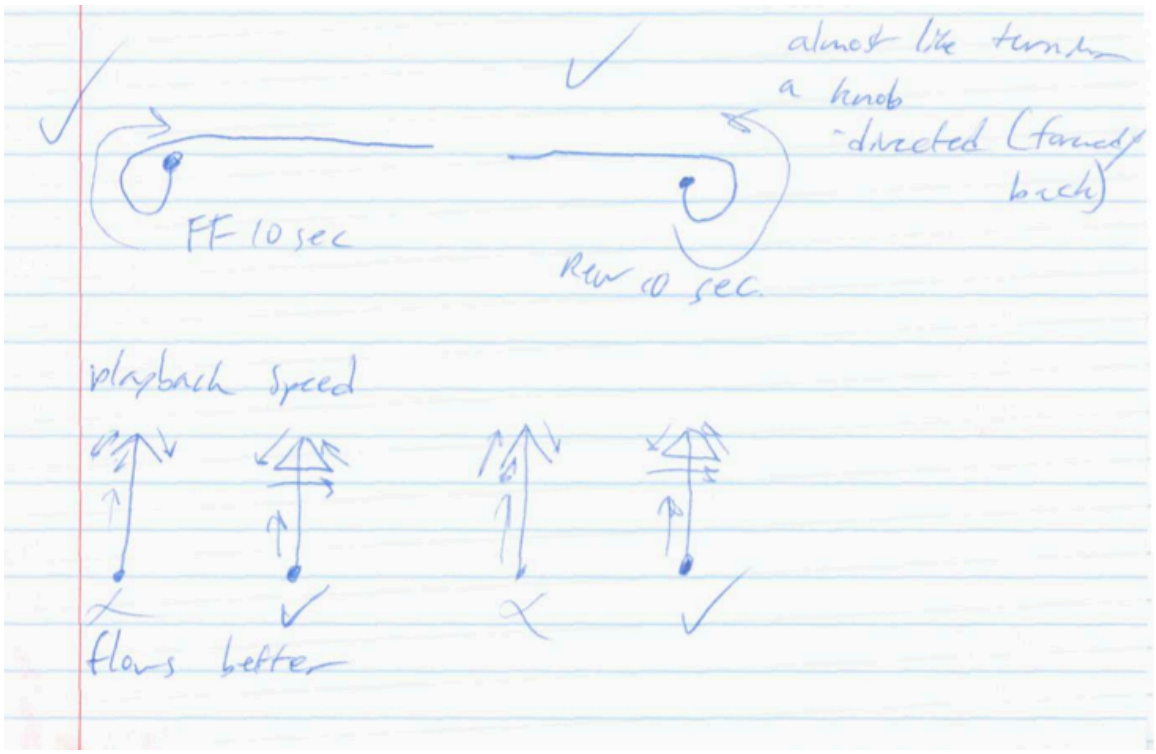

Figure 8: Final Fast-Forward, Rewind, and Video Playback Speed Adjustment Gestures

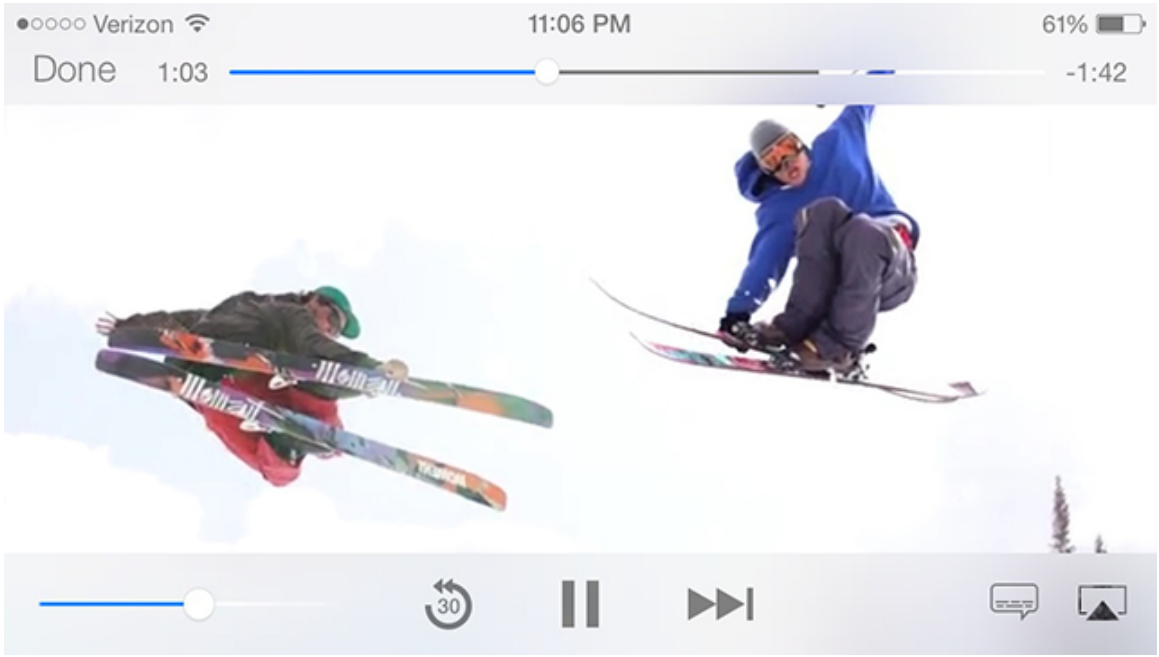

Figure 9: iOS 7 Video Player

Figure 10: Screenshot from RSSDemon v3.1.8 Changelog

For the fast-forward and rewind gestures, I looked to two popular designs for inspiration. Figure 9 shows the iOS7 video player. Note the "rewind 30 seconds" button. Figure 10 is a screenshot from a popular Android applications "RSSDemon" showing the new "swipe to rewind and fast-forward" gesture that can be used while listening to a podcast. I wanted to create a swipe gesture that used the circular aspect of the rewind button in the iOS 7 video player, but also had the connotation of a swipe moving forward or backwards in time. So I created the "circle-to-line" gesture to use for my application.

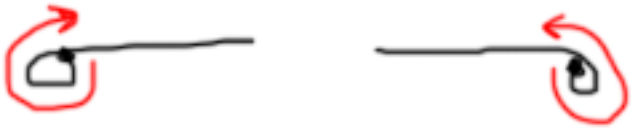

Figure 11: The "Circle-to-Line" Gesture Created for my Application

$1 vs. $P – Why $1?

I chose $1 as my gesture recognizer for a two main reasons.

Usability
A single stroke gesture is easier for a user to input than a multi-stroke gesture, especially with a mouse.

Less Margin of Error
Using $1, I don't have to worry about users accidentally misusing my gesture recognizer. What happens if a user completes the first part of a swipe, but not the other parts? I would have to handle more errors. With $1, there's no need to handle these errors, especially if I decided to try and implement a gesture timeout in case a user inputted half of a gesture, but couldn't figure out how to cancel the gesture in progress and start a new gesture.

Evolution of Design – Development

Every web app needs a good name. I chose Swiper and the tagline "No Typing, Just Swiping."

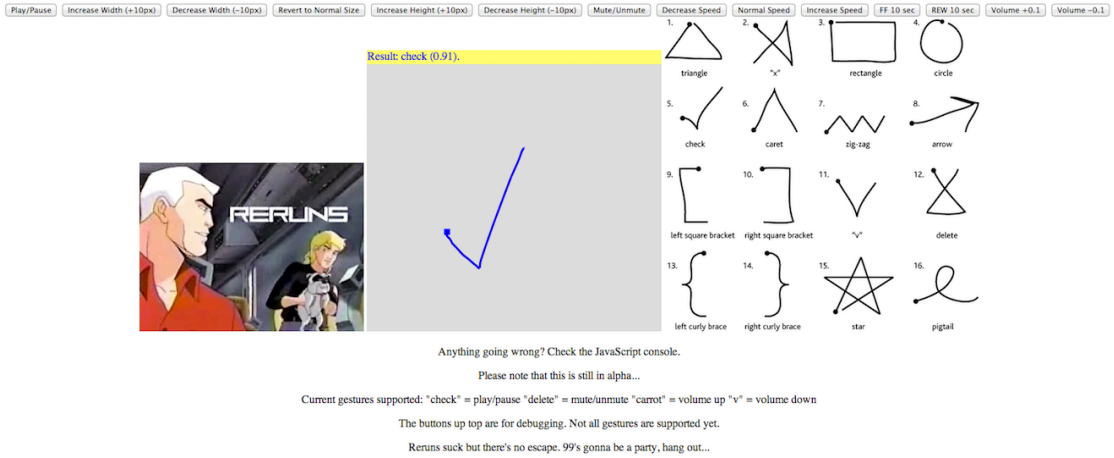

Figure 12: Initial Design and Canvas Test Using Basic HTML

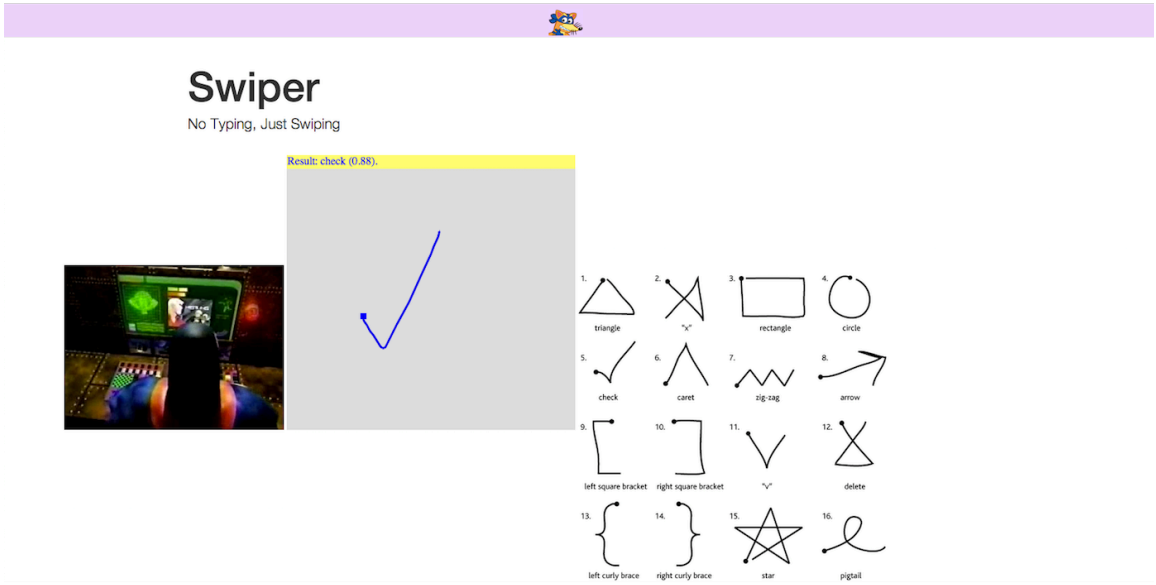

Figure 13: First tests with Bootstrap and proper CSS

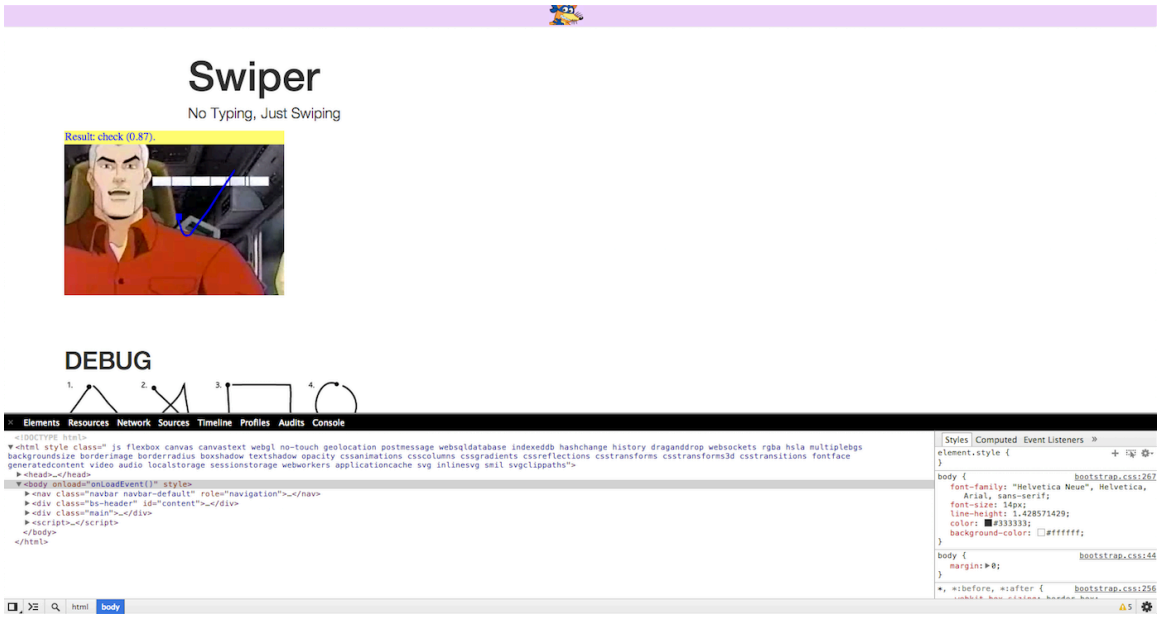

Figure 14: Improving Layout, First Tests with Canvas Overlaying Video Player

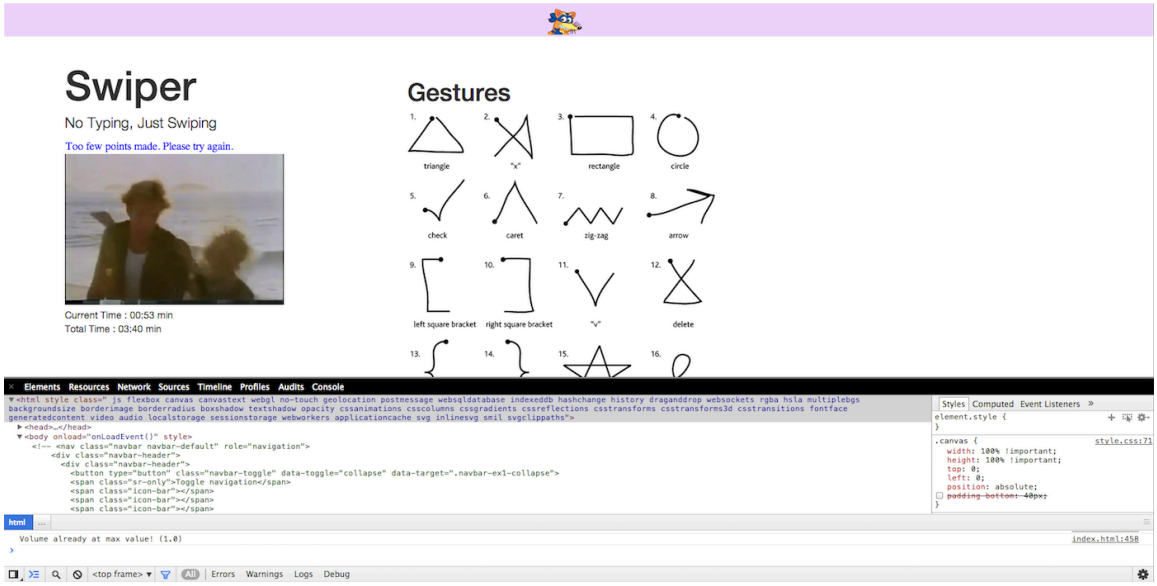

Figure 15: Layout Improvements, Added Video Running Time Views, Canvas Customizations

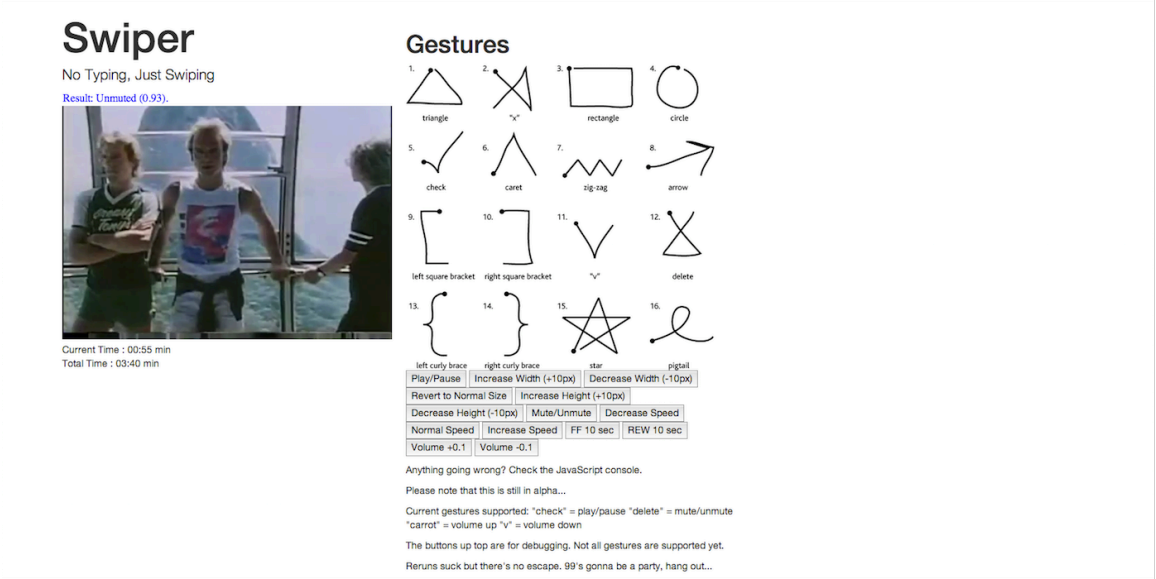

Figure 16: Video Sizing Tests

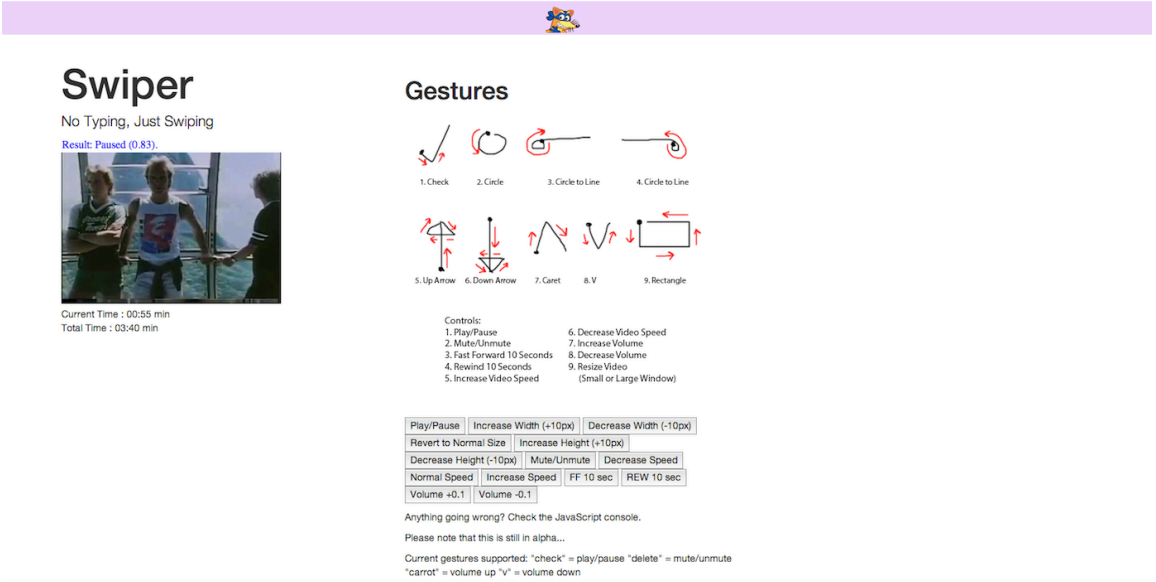

Figure 17: Added Custom Gesture Help Image, Finalizing Layout and Debugging

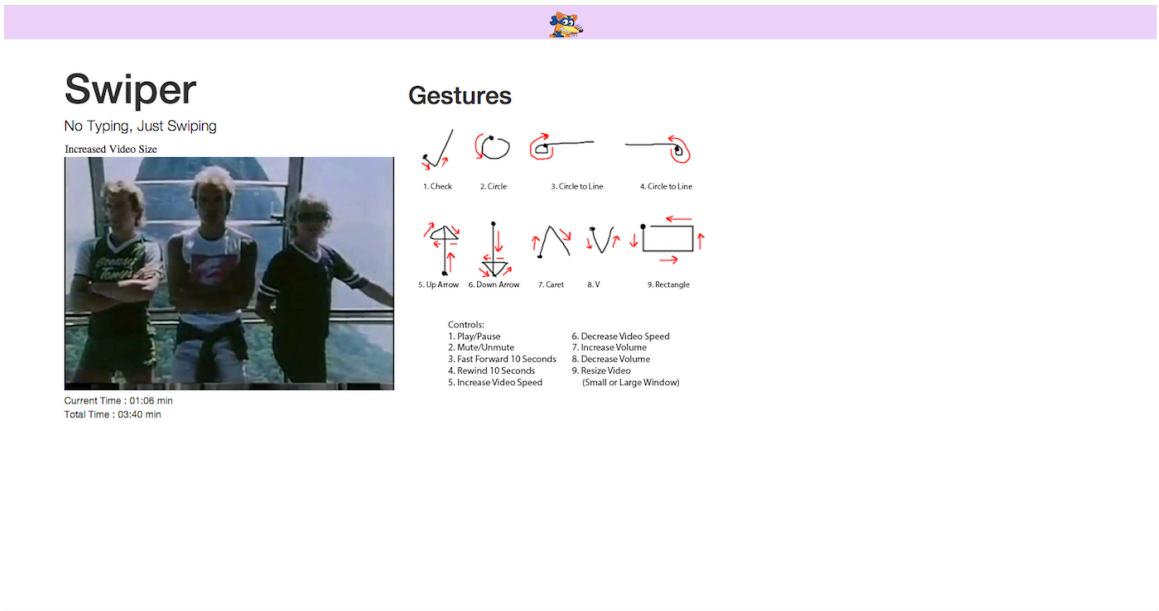

Figure 18: Release Candidate

Using Swiper

Using my gesture-controlled video application is very easy. After loading the page, the user is presented with this view:

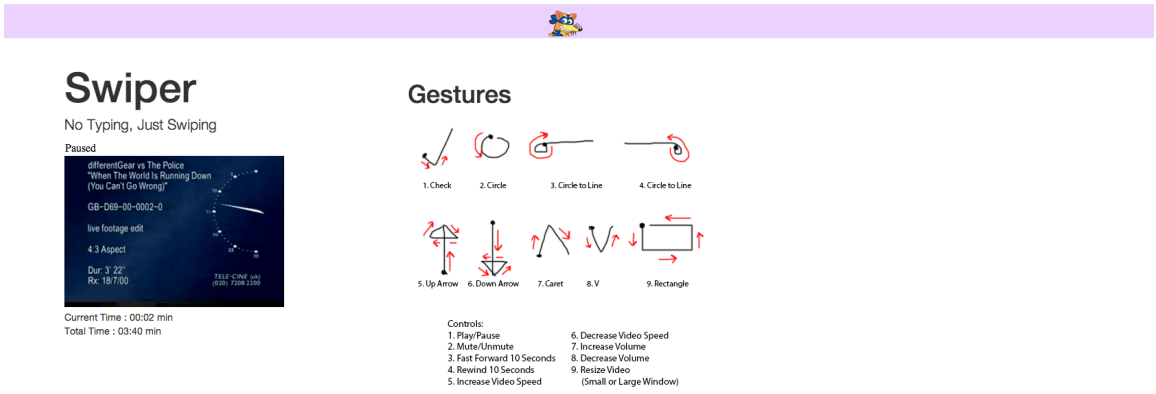

Figure 19: Initial View of Application

To start playing the video, the user can draw a "check" gesture, as seen in the image under the "Gestures" heading:

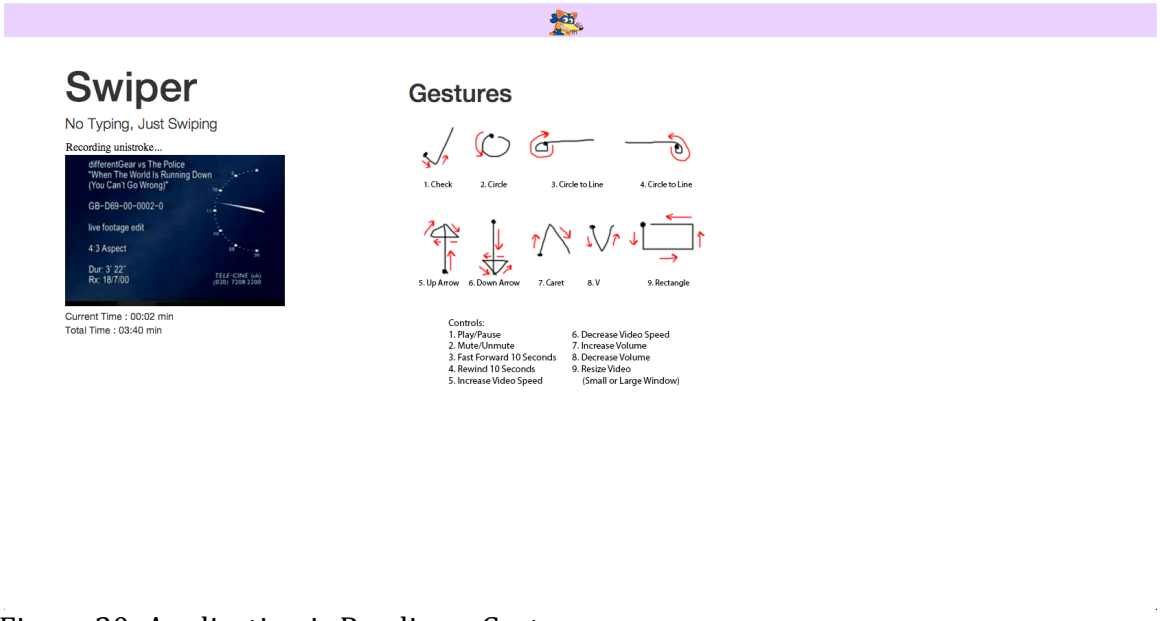

Figure 20: Application is Reading a Gesture

If the user performs an unregistered or unimplemented gesture, the application will respond appropriately:

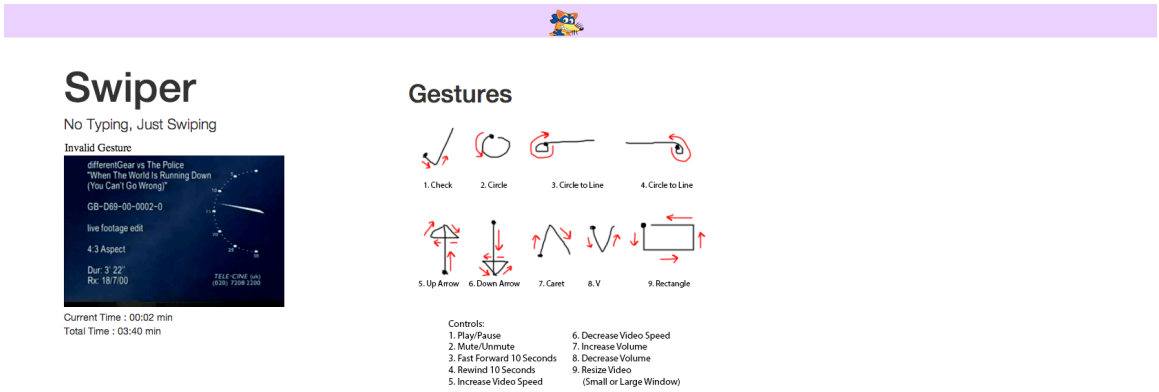

Figure 21: "Invalid Gesture" Error Message

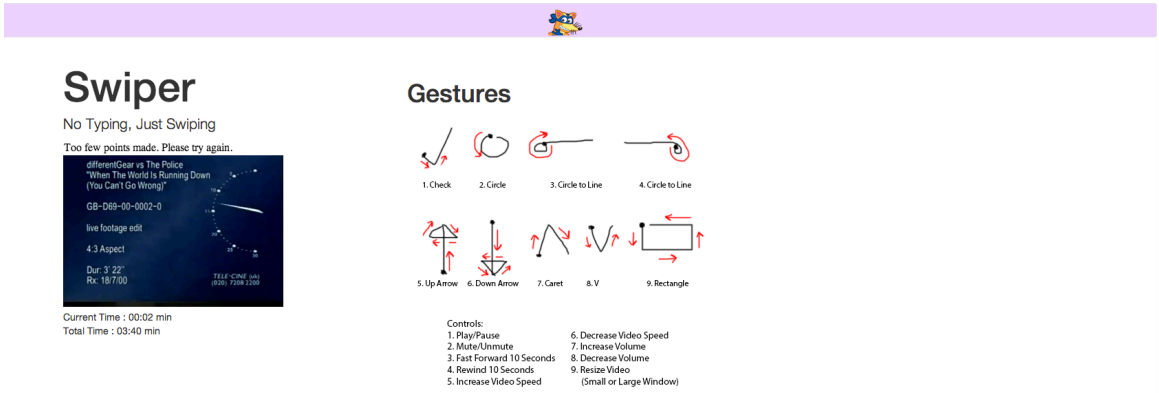

Figure 22: "Too few points made" Error Message

Once the user correctly inputs the "check" gesture, the video will start to play. Notice how the current time attribute updates accordingly below the video player:

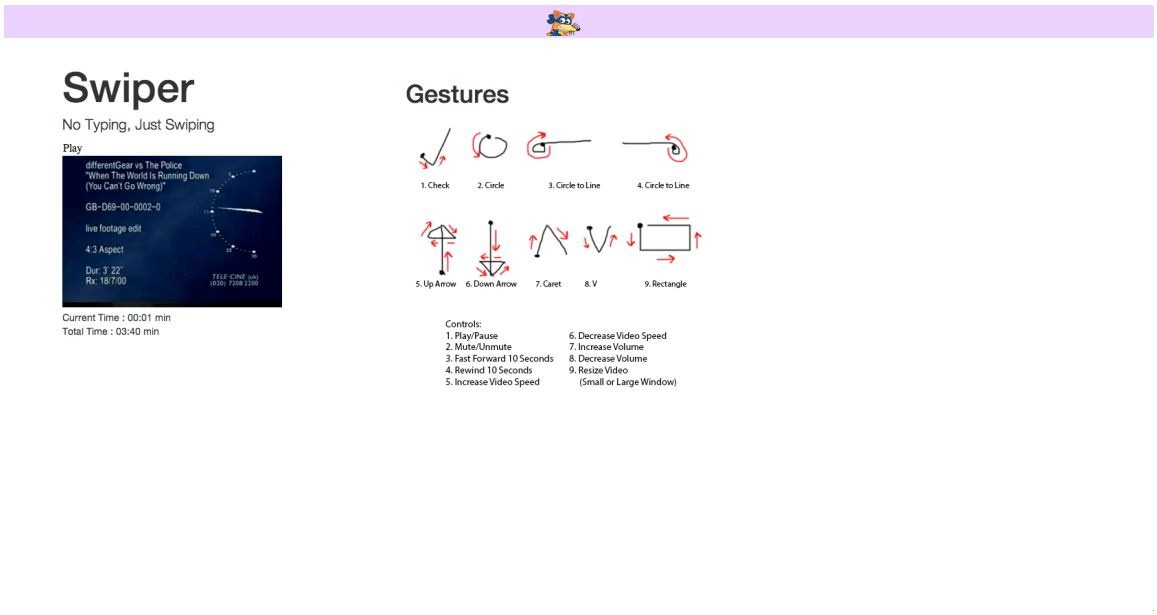

Figure 23: Video Playback

As the user performs gestures, the canvas will respond by displaying the action that the user just inputted, as confirmation, and the video player will respond with the appropriate action

Persona

John is a tech-savvy college student. He has multiple touch-screen devices, from an iPhone 5, to an Android tablet, and a Google Chromebook Pixel. These devices can primarily be controlled by the touch screen, which John likes to use as much as possible. John is looking for a less-conventional video player, which allows him to quickly adjust video attributes and settings without tapping a button. Swiper is the answer to his needs. John can quickly control a video, even without looking for a button, or while looking at another screen.