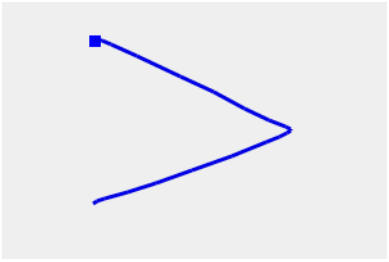
Part I. Gestures

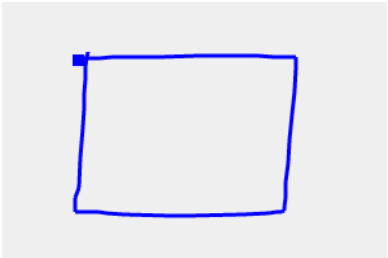

Play/Pause

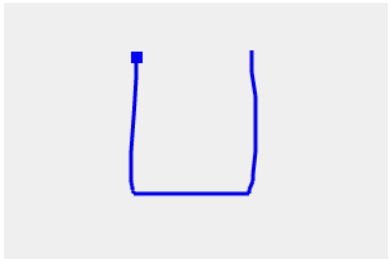

Seek Backward

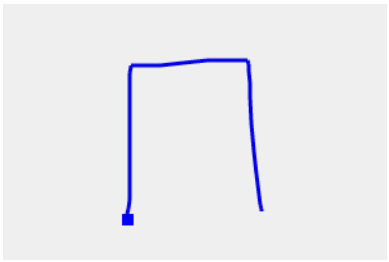

Seek Forward

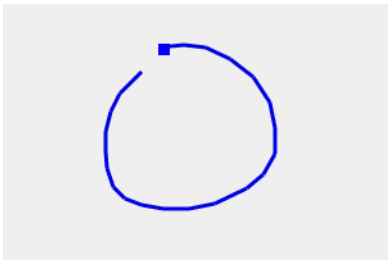

Speed Decrease

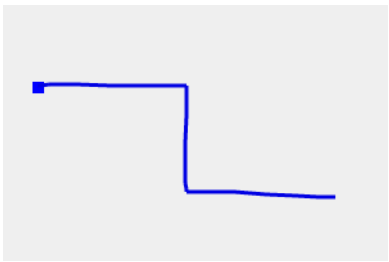

Speed Increase

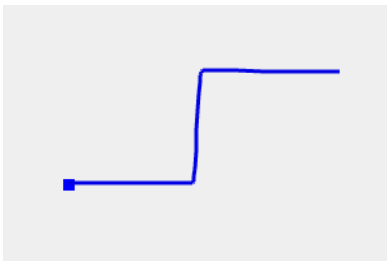

Speed Reset

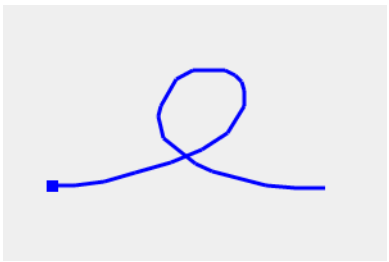

Volume Decrease


Volume Increase


Mute/Unmute


Size Decrease


Size Increase

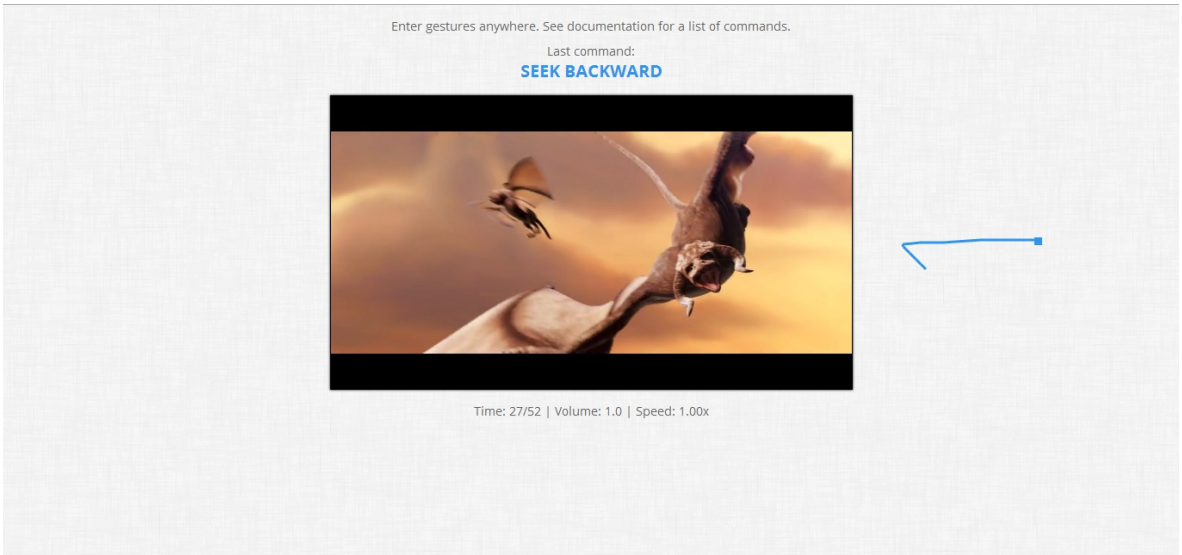

Seek to Start

Part II. Introduction

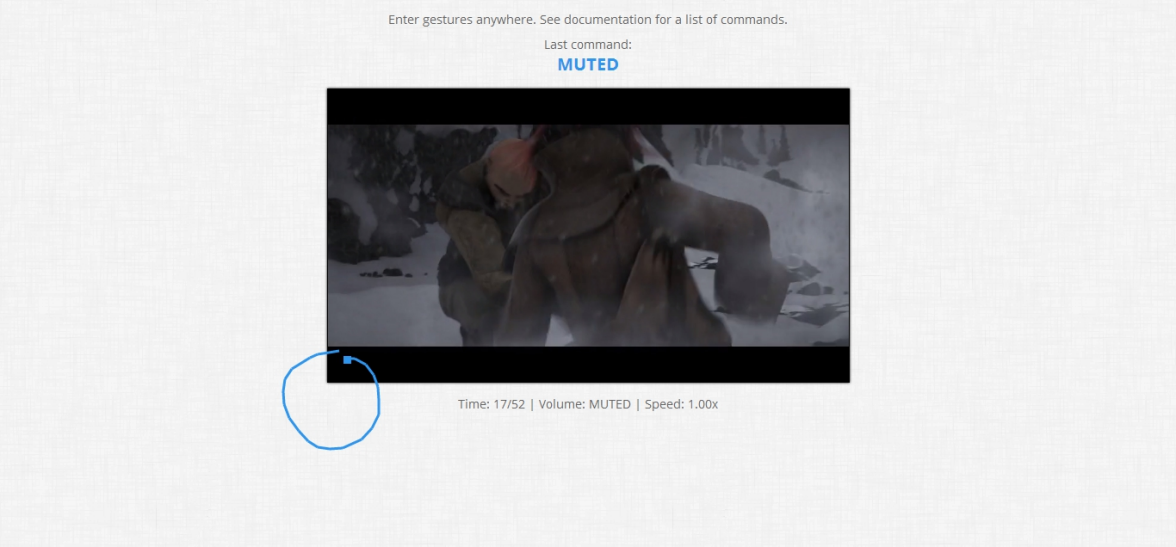
This application is a gestural video player built using JavaScript and HTML5 (<video> and <canvas> elements). The standard video player controls have been disabled. The user controls the video with only the gestures above. These gestures can be input anywhere in the window.

Using the application is very simple – make a gesture by pressing and holding down the left mouse button and releasing when the gesture is complete. Using the unistroke recognizer

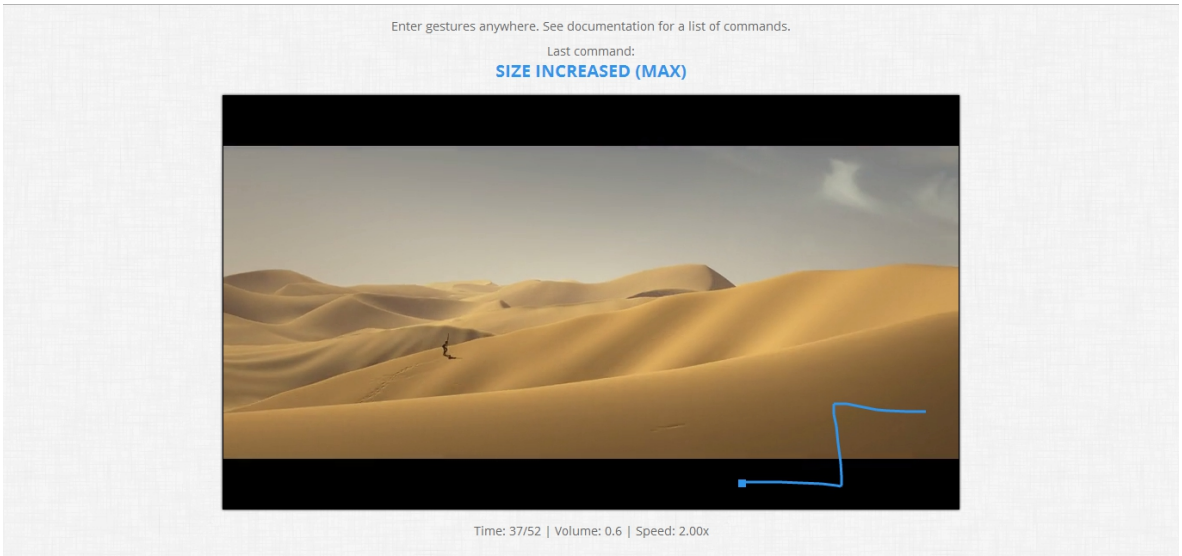
algorithm provided, the application will then determine which gesture was entered and execute the corresponding video command. Several screenshots of the application have been provided, but do not demonstrate every available feature.



Here the user has entered the “seek backward” gesture on the right side of the video, which sets the current time of the video back 10 seconds. Note that the last command entered is shown at the top so that the user knows if the gesture was successfully recognized.



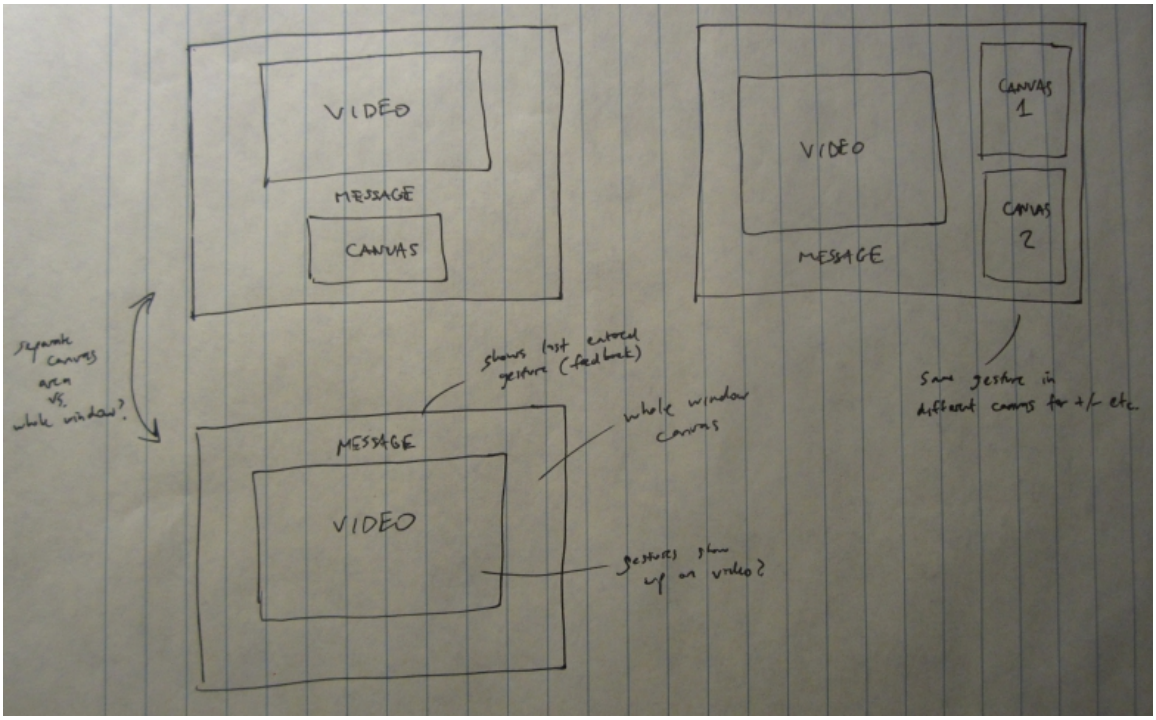
In this screenshot, the user has entered the mute/unmute gesture. Because the video was unmuted, this gesture mutes the video. Note that “MUTED” appears next to 'Volume' at the bottom. The application includes a rudimentary display of information regarding elapsed time, volume, and playback speed. Also notice that the gesture was entered partially inside of the video element. Gestures can be entered anywhere in the window – outside of the video, inside of the video, or overlapping both areas.



Here, the user has entered the gesture to increase the size of the video. Playing around with the application will give a better idea of how it works and responds to gesture input. Note that some commands require the video to be playing. To play the video initially, use the “play/pause” gesture.

Part III. Development

In the first stage of development, sketches were made to determine the layout of the application:

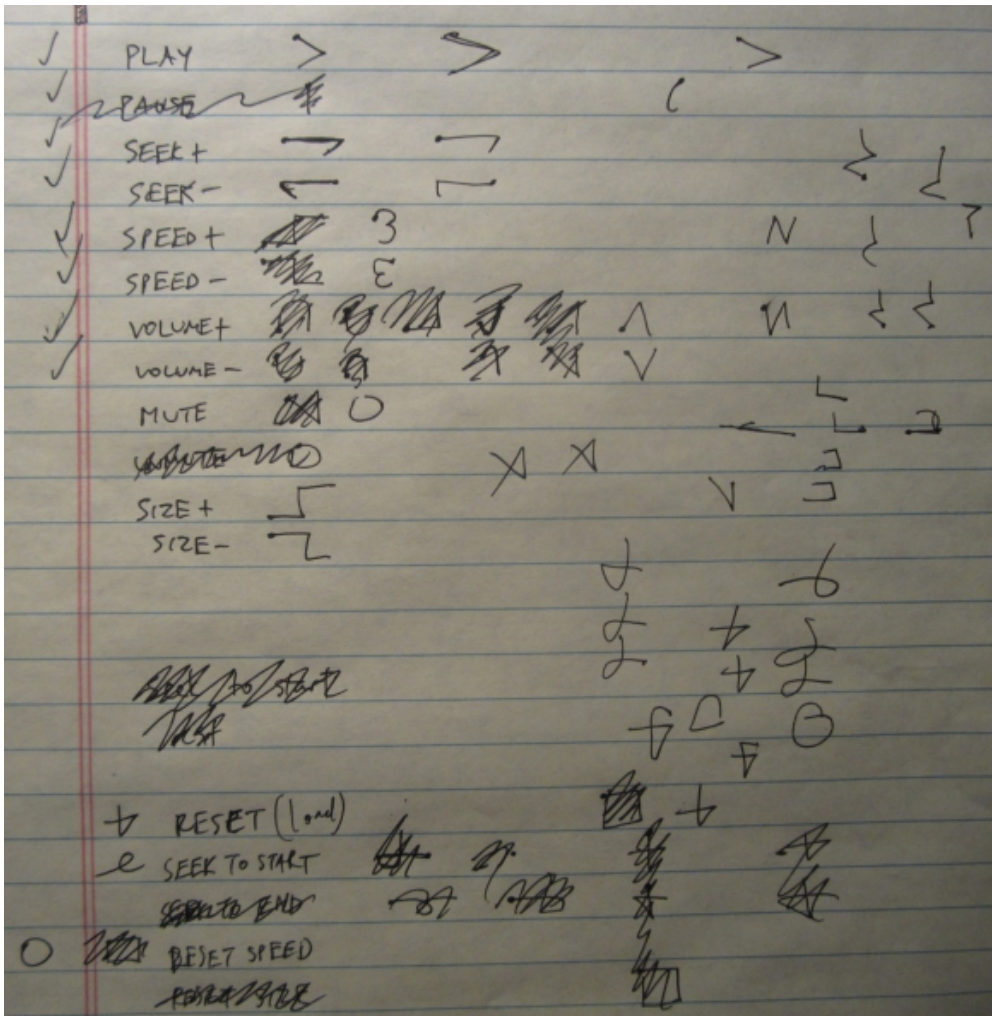


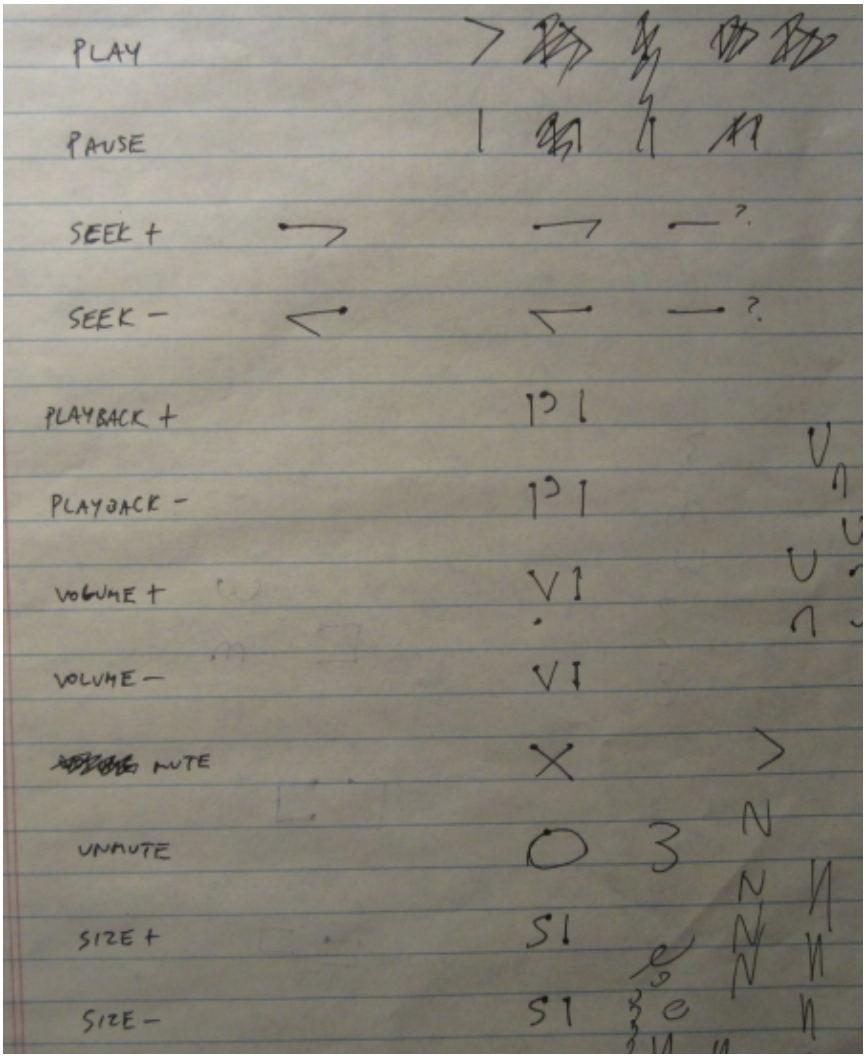
The above sketches show several possible layouts that were considered. The most important part

of designing the application layout was the <canvas> element – does the user have to enter gestures in (1) a designated area, (2) within the video element, or (3) anywhere on the screen? In the sketch, the design in the top left and the design at the bottom are similar except that the former has a designated canvas area where the user must enter gestures. The top right design included two canvas areas to distinguish between commands such as "volume increase" vs. "volume decrease." In this design, one gesture could correspond to different commands depending on which canvas it was entered in. This idea was scrapped because it was too confusing and inconsistent with the concept of only using gestures (in some sense, each canvas could be considered a button).

The bottom design is very close to the final design that was implemented – it contains a message to the user (including feedback about the last entered gesture) and the video itself. Without the need for buttons or other user interface features, the design is naturally very minimalist. The information below the video (current time, volume, and playback speed) was not included in the initial sketches, but was added later (for reasons that will be discussed in Part IV).

After the layout was decided, the next major part of the development process was creating the gestures themselves:

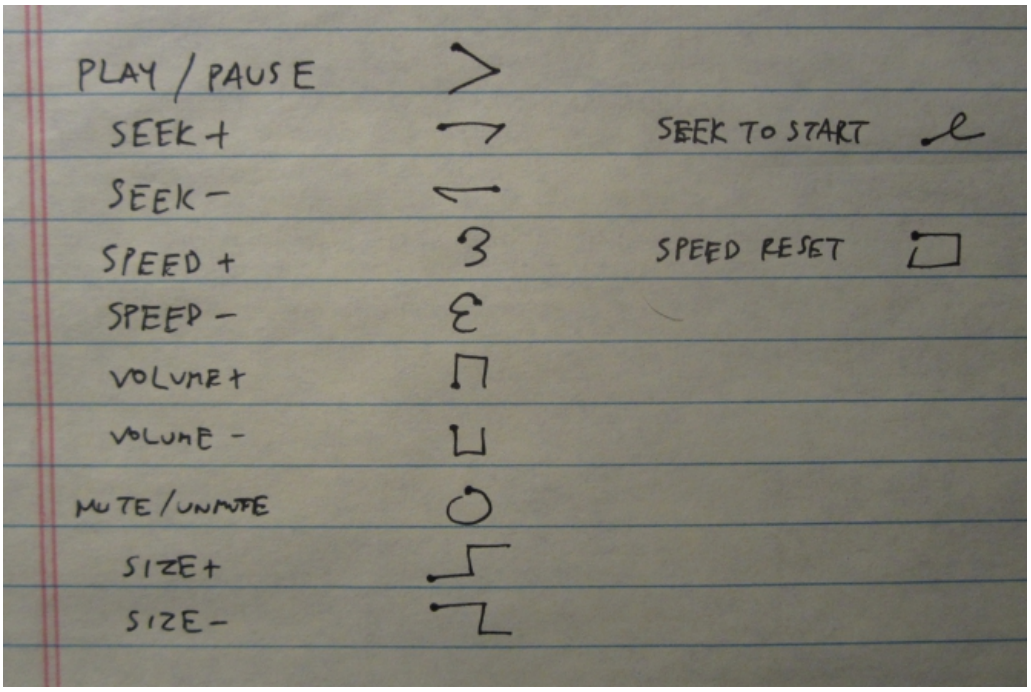




As you can see, much of the development stage consisted of designing gestures that were intuitive and easy to remember, but also distinguishable from other gestures using the given recognizer algorithm. Many of the original gestures had to be redesigned and made more complex because the application would incorrectly recognize a gesture that was too similar to another one (especially if gestures were entered quickly and without much care). Some gestures were awkward to recreate or simply did not “feel right” in terms of usability and convenience.

The second image shows a set of gestures designed for the multistroke recognizer. The concept in this design was to enter one gesture for the command and then another to indicate increasing or decreasing. For example, a user would enter an S-shaped gesture followed by an up or a down gesture for increasing or decreasing size. However, this idea was discarded because the multistroke recognizer does not distinguish the direction that a gesture was made; an upwards line and a downwards line would be treated as identical. Additionally, the multistroke recognizer requires the user to enter a special input when they are finished with the gesture (e.g. right click).

This felt awkward when compared to the simple gestures designed for the unistroke recognizer. Ultimately, the following set of gestures was chosen (these are the same gestures shown in Part I). The decisions made in regards to the design of these gestures will be discussed in the next part.



Part IV. Design Decisions

Layout – As mentioned previously, the layout was designed to be minimalist. This occurred naturally because the objective was to create a video player without standard controls (buttons, sliders, etc.). This minimalist design influenced the decision to allow gestures to be entered anywhere in the window instead of inside a designated area or the video element. There is no clutter on the application page – it is essentially just the video itself. While entering gestures on the video itself can obstruct the user's view, there may exist some situations where the user wants to be able to enter gestures anywhere, for convenience. Because a user can enter gestures outside of the video anyway, this is a negligible flaw. There is plenty of space to make gestures on either side of the video as well as below it (if it is not set to maximum size).

Feedback – The application displays the command corresponding to the last entered gesture at the top of the screen. If the last entered gesture was invalid (not recognized as one of the available gestures or too few points entered), “NONE” is displayed. The reasoning for this feedback is to let the user know whether the intended gesture was entered. If the wrong gesture was entered, the message at the top of the screen would show which gesture, if any, was recognized. This allows the user to easily undo a mistake (i.e. entering the “speed decrease” gesture to nullify an accidental “speed increase”) or notice that he or she is entering gestures incorrectly.

This feedback message is easily noticeable; it is important for the user to receive feedback for any input, whether it is a valid gesture or not. Additionally, this message will tell the user if a maximum or minimum has been reached. For example, if a user has increased the volume to the maximum value, the message displayed will be "VOLUME INCREASED (MAX)." Any subsequent attempts to increase the volume will display the same message without increasing the volume further. Lastly, for the "toggle" gestures ("play/pause" and "mute/unmute"), the message will display the correct command depending on the video state. If the video is paused and the "play/pause" gesture is entered, the message displayed will be "PLAY."

Video information – Below the video, the application shows the current time of the video, the volume, and the playback rate. Because these values change, it is important to let the user know what the current values are, especially since the user has control of these aspects. When the user changes the volume or speed, it is helpful to know the current value as well as how much the value changes after the command is executed. Knowing the current time of the video is essential for using the seeking commands. The current time display is very basic (just text). Ideally there would be a more traditional progress bar for elapsed video time as well as a bar for volume and icons for muted/unmuted states. However, these user interface features are not important to demonstrate the gesture-oriented nature of this video player application.

Gestures – Overall, the gestures were designed to be intuitive, easy to learn, and easy to replicate. Initially, the gestures were supposed to "make sense" with respect to the corresponding command. For example, the "play/pause" gesture is very similar to the widely used symbol for a play button on any common media player. The "seek backward" and "seek forward" gestures are meant to resemble arrows facing backwards and forwards respectively. Unfortunately, while developing the gestures, it became difficult to include meaningful gestures that would be easily distinguishable by the recognizer algorithm. The "volume increase" and "volume decrease" gestures were originally meant to be triangular to represent and up or down arrow (like the "play/pause" gesture but rotated). The unistroke recognizer had difficulty distinguishing between the "play/pause" gesture and the volume control gestures, so the volume control gestures were changed to rectangular shapes (they somewhat retained their upward/downward directions).

A similar process occurred for a number of the other gestures. This explains why certain gestures do not seem to have meaningful shapes. The playback speed control gestures have an unrelated "3" or backwards "3" shape due to the the shapes being unique and easily recognized by the unistroke recognizer. Still, most of the gestures make intuitive sense – the size control gestures clearly represent an increase/decrease and the "seek to start" gesture consists of a loop shape, which suggests starting over.

On the topic of precision, the user has no control over the amount of increase or decrease when a command is executed. This decision was made because it did make sense to have absolute precision when creating a gesture-based application. Entering exact values would be extremely awkward using gestures instead of keyboard input. Likewise, seeking to an absolute point in the

video would be difficult without a slider. As such, the increase/decrease and seek commands only modify the video relative to current values. This is acceptable for attributes such as volume, but for speed and current time, there are some values which the user should be able to get to immediately. Thus, the “speed reset” and “seek to start” gestures were created to allow the user to quickly set the playback speed to the default or navigate to the start of the video. Without these commands, it would be inconvenient for the user to do these actions (many speed change or seek inputs would be necessary). While it is true that these “shortcuts” are not necessary, they are included for convenience.

Another important aspect in the design of the gestures was consistency. Since many of the commands have an increase/decrease or forward/backward duality, the two corresponding gestures were designed to be similar shapes but with opposite orientation (see “seek backward” vs. “seek forward” and “volume/size/speed increase” vs. “volume/size/speed decrease”). The gestures that do not follow this pattern are the ones that have no opposite counterpart. Originally, there were separate gestures for play and pause and for mute and unmute. However, since play is only useful when the video is paused and vice versa (same for mute/unmute), it made more sense to combine them into one “toggle” gesture. This is very intuitive since in standard media players, usually the same button is clicked to play and pause or mute and unmute, even if the button changes depending on the state.

One last decision that was considered in regards to designing the gestures was simplicity. All of the gestures included in the application are easy to replicate and do not contain any complex shapes or large amount of necessary points. The more commonly used gestures are simpler than the ones that are rarely used. The “play/pause” and seek gestures are literally two lines and take almost no time to enter, while resetting the speed (not a common action) requires four lines. Even so, the most complicated gesture is still a simple shape. The gestures were designed this way to (1) allow new users to learn the gestures quickly and (2) let expert users (who already know the gestures) execute commands quickly, especially ones they use a lot.