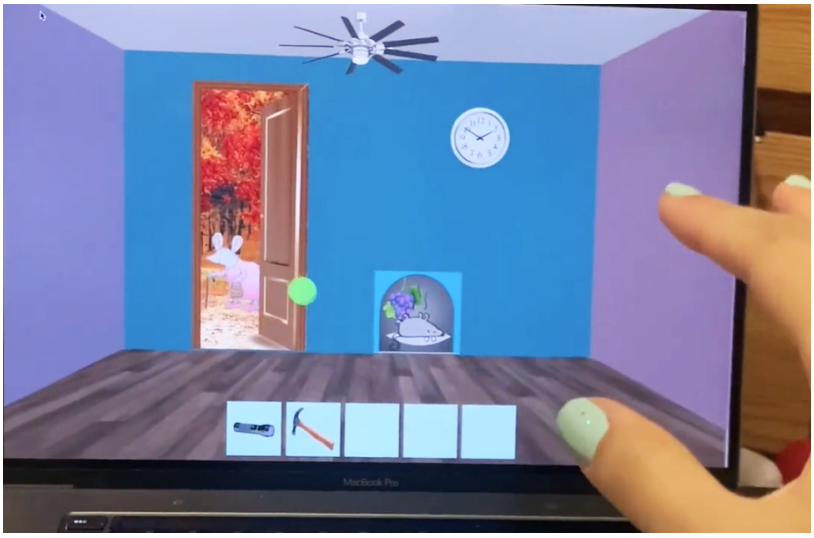
IMMERSCAPE

Interactive Multi-Modal Escape Room
Simulator Combining Audio and Physical Expression

Team Members: Karen Gao and Hizami Anuar
May 10th, 2022



Abstract: We built an interactive virtual escape room by combining LeapMotion's ability to track hand poses and motion, Web Speech API to recognize speech, and graphics rendered by the Famo.us library. The system checks for gestures/speech commands (open, close, turn on, turn off, press, turn right, grab, use an object from inventory, etc.) from the user through the leap and laptop's microphone, processes it in the game logic and gives both visual and audio feedback such as the drawer opening, the light turning on in all of the room, and sound effects and descriptions through Google's Speech-to-Text for the audio feedback such as ("you just obtained the cheese", "can't use hammer on door", etc.). It knows 7 hand gestures and around 16 voice commands, where some actions can be done with either speech or hand gestures depending on the user's personal preference. The system's accuracy rate in hand detection was around 91% on average over all of the hand gestures with 10 trials each. Its accuracy for speech detection was nearly 100% averaged over 8 trials for 4 of the random commands we picked out to test. It worked pretty well for speech detection and certain gestures like turning, grabbing, opening, and zooming in , but struggled with more complicated gestures like pressing a button, partly due to difficulty in classifying a motion as one of the moves and also the finnickiness of the LeapMotion detector.

1 Introduction and Overview **2**
1.1 Motivation 2
1.2 Solution 2

2 System Description **3**
2.1 Tutorial 3
2.2 Talking to the Capybara 4
2.3 Opening the Safe 4
2.4 Unlocking the fridge and obtaining the capybara 4
2.5 Escaping the Room 5

3 Implementation **6**
3.1 Key Functionalities 7
3.1.1 Hand Gesture Recognition 7
3.1.2 Speech Recognition 7
3.1.3 Speech Generation 8
3.2 Difficulties and Failures 8
3.2.1 Difficulty: Gesture for Panning Left and Right 8
3.2.2 Difficulty: Over-Chatty Program 8
3.2.3 Failure: Gesture For Closing 9
3.2.4 Failure: Gesture for Pressing 9

4 Roadblocks and Pivots **10**
4.1 Revamping Audio Commands 10
4.2 Speech Generation Interfering with Speech Recognition 10
4.3 Clearing Up Confusion 11

5 User Study **12**
5.1 Qualitative Results 12

6 Performance **13**
6.1 Gesture Detection Accuracy 13
6.2 Speech Detection Accuracy 14
6.3 Room for Improvement 15

7 Tools, packages, and libraries used **17**

8 Collaboration **18**
8.1 Hizami 18
8.2 Karen 18

1 Introduction and Overview

1.1 Motivation

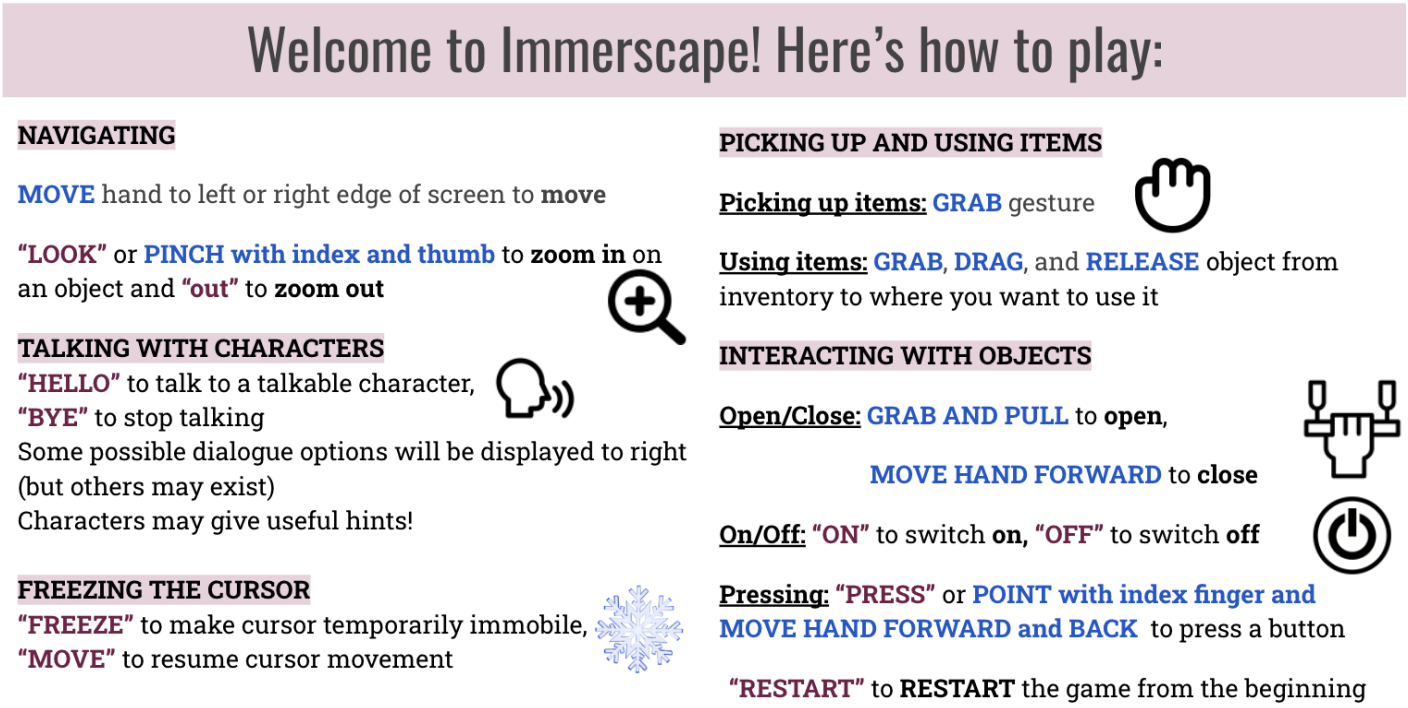
Escape rooms are a recent trend where users are placed into a room filled with puzzles they must solve in order to escape a room. Escape rooms can be great team-building exercises as they typically require great amounts of cooperation and brainstorming to be completed. However, escape rooms can only be effectively enjoyed by individuals in physical proximity to each other. In our more modern globally connected world, especially with the recent pandemic, it has become increasingly necessary for individuals to be able to interact and bond over long distances. While some virtual options exist such as the 3D game Escape Simulator, or point-and-click adventure games, they are effectively just that - your main mode of interaction is pointing your mouse and clicking (or using the keyboard for input). They lack one of the fundamental elements that make escape rooms so fun, which is a direct immersion into the scenario of the escape room. In these virtual games, there is a clear disconnect between the player and the game environment.

1.2 Solution

This issue is the motivation behind our project, Immerscape, a multimodal virtual escape room that allows users to use gestures and speech to interact with objects and characters in the virtual environment. By replacing the boring keyboard and mouse input of existing virtual escape room games with more engaging input of gestures and speech, we hope to provide a more immersive experience to the user, creating a virtual environment that the user can interact with in ways that more closely mirror a real physical environment of an escape room and even improves upon the qualities of an escape room utilizing the capabilities of a virtual world.

2 System Description

Here's a summary of all of the gestures and voice commands the system recognizes that we show the user before playing our game in the help screen:



As for the tasks we gave the user to perform, we barely have any except telling them to explore around the room to find things to unlock the door and escape the room. Below is a real example of user input and the system's response:

2.1 Tutorial

The user starts by going through a tutorial teaching them the gestures for turning left/right, opening drawers, turning things on/off, freezing the cursor, talking to animals, grabbing items, and using items from the inventory on other objects such as the door. If the user turns the wrong way, our system tells that the user. Finally, when the user unlocks the door with the key obtained from the drawer, the system congratulates them, and the user points through the open door to enter the actual escape room.

2.2 Talking to the Capybara

The first thing the user sees when they enter the actual escape room is the wall with the talking capybara. The user tries to interact with it by saying "hello", which pulls up a

dialogue screen with the capybara with a set of questions for the user to ask the capybara that changes depending on what you ask/respond to the capybara, where keywords that our system responds to are capitalized. The user asks the capybara “What is the SAFE CODE”, and the capybara responds and says that it doesn’t remember but that it starts with the number 2. The user then asks “why is the mouse so sad”, to which the capybara responds that Steve is sad because he wants some cheese. The user exits the dialogue by saying “bye” to the capybara, which takes them back to the zoomed-out view of the wall.

2.3 Opening the Safe

The user explores the rooms by waving their hand left and right. He opened the dresser and obtained a UV flashlight as well as grabbing the cheese that was sitting on top of the fridge in the wall with the watermelon painting. The user notices some very faint letters on some of the walls that are barely visible and decides to turn off the lamp to get a better look by hovering over it and saying “off”. The user tries to use the flashlight on the letters on the walls by grabbing and dragging the flashlight from the inventory and releasing it on the wall sections that turn the cursor to green which means that it is interactable. Once the flashlight’s been released, the user can see the hint on the wall that says “my favorite number is _45”, which the user assumes is the last 2 digits of the safe code. The user then tries to find the safe by opening the watermelon painting and a safe is revealed behind it. The user tries zooming in on the safe by pinching their index and thumb. The user then puts in the safe code one by one by hovering over each of the buttons and moving the hand slightly forward to “press” the button and then hitting the enter, which unlocked the safe zoomed out and gave the user a hammer.

2.4 Unlocking the fridge and obtaining the capybara

With the cheese, the user tries to feed it to the mouse, which gives him a small black key. The user tried using it on the main door and it didn’t work. The user then tries to use it on the fridge door and the door is unlocked. The user opens the fridge door and obtains a piece of watermelon. The user first feeds it to the mouse and the mouse doesn’t want it so he then tries to feed it to the capybara, which the capybara ate and then became “grabbable” so the user grabbed it and put it in his inventory.

2.5 Escaping the Room

The user tries to use the hammer on the door, fails, and then uses it on the window instead, which shatters but the user can’t fit through since the hole is too small. The user then tries to throw the capybara out the window, which works and the capybara escapes

the room and unlocks the front door from the outside, reuniting the mouse with the grandma and allowing the user to finally escape the room.

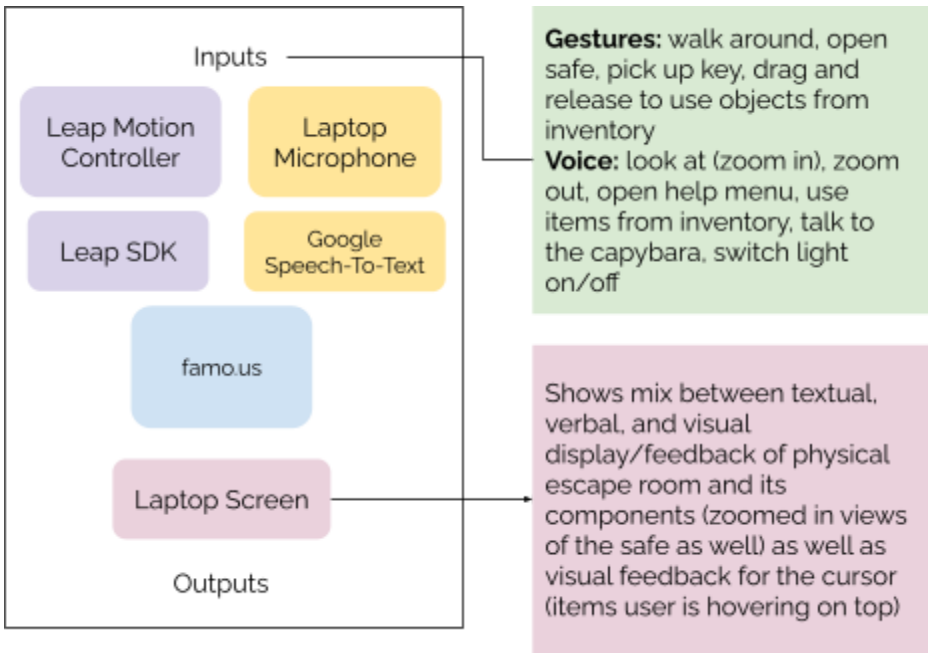
3 Implementation

The system uses the Leap motion sensor which passes its data to the LeapSDK in order to process it and detect key features of the hand including position and orientation of hand, as well as other features such as velocity of hand and which fingers are outstretched. The system takes audio input through the laptop microphone, which is processed by the Speech Recognition API Webkit and processed into actual words/phrases.

Our system then processes the data provided by the LeapSDK and the speech recognition software in order to determine what action the user is attempting to make through mathematical computations and logic. The game logic processes the determined action and its effect on the game.

It then passes on information about the game state to the famo.us rendering engine allowing it to re-render the game with the appropriate visual changes. It may also play audio clips or send speech lines to be spoken by the Speech Synthesis API Webkit.

The basic architecture of our implementation is portrayed below:



3.1 Key Functionalities

3.1.1 Hand Gesture Recognition

For our system, we chose gestures that had distinguishable features allowing them to be easily recognizable, such as the fist-like grab gesture or the pointing gesture. This allowed us to recognize gestures by doing computations on the features of the hand detected by the Leap motion sensor rather than having to create a large dataset to train a machine learning model (and retrain everytime we added new gestures).

The Leap motion sensor is quite powerful at detecting feature-level details such as when the user is grabbing or when specific fingers are outstretched. So detecting certain gestures such as the grab gesture or the pointing gesture was quite easy, and only required determining acceptable threshold values for gesture recognition. However, sometimes movement-based gestures were a bit tricky.

3.1.2 Speech Recognition

How does it work?

For speech recognition, we use the Speech Recognition API Webkit built into the browser, which provides an ongoing transcript of what the user is saying. Our speech processor then splits this string into its separate words and detects if any of these words is a keyword triggering an action or event.

How well does it work?

Originally, our speech recognition was quite poor. Many words were repeatedly misrecognized as other similar-sounding words, such as “freeze” recognized as “breath” or “close” as “clothes”. While we would have liked to provide keywords to help improve the speech recognition, it appears that it did not have such functionality that is typical of speech recognition software. So instead, we decided to create a list of similar-sounding words for each keyword, allowing the speech recognition to be more robust. We generated these lists by having various users repeat certain voice commands and reviewing the transcripts to see what the speech recognition recognized them as saying.

Speech recognition tends to perform a lot better when there is context surrounding the word spoken, i.e. it is spoken into a full sentence. So for speaking to characters in our game, speech recognition actually performs decently well.

We also noticed that sometimes the Google-Text-to-Speech would make the word detected to be capitalized, like “Goodbye” instead of “goodbye”, and sometimes even adds a

period at the end of the word for no reason, like “goodbye.”. To combat this, we added those alternatives detected to the list of words to detect for many of our speech commands that were commonly misclassified as something else to improve performance. Although, a better, more robust solution would be to standardize the transcript by converting all letters to lowercase and removing punctuation.

3.1.3 Speech Generation

We use the browser’s built-in text-to-speech API in order to convert text strings to speech. One small issue with this implementation is that there is large amounts of variance between operating systems and browsers used, so the speech generated with certain config options on one computer may sound completely different from the speech generated on another computer.

3.2 Difficulties and Failures

3.2.1 Difficulty: Gesture for Panning Left and Right

Our gesture for turning left or right was simply the user moving their hand to the left or right edge of the screen (causing the user’s view to change by 90 degrees in the desired direction). However, when the user moves their hand to the edge of the screen, it typically stays there for several frames. If we were to just trigger the turning action every frame the hand is detected offscreen, several turns would be made when the user likely only intends to make one. We needed a way to ensure that turns are only recognized once. So we came up with the following solution: once a user’s hand is offscreen and the turn action occurs, turning is disabled until the user’s hand re-enters the screen (for a small duration of time). An alternative solution could have been implementing a timer that disables turning for a set duration of time, allowing the user to turn multiple times by keeping their hand offscreen for longer. However, this implementation felt a bit strange and unintuitive and we didn’t want to force the user to move their hand back in frame immediately.

3.2.2 Difficulty: Over-Chatty Program

Our game had an audio response to almost every action the user could make. Sometimes, there was even a large queue of generated speech built up as some of the speech lines were exceptionally long, resulting in the program saying things in a very delayed manner long after the relevant context. We solved this by canceling any on-going speech generation before attempting to generate a new line of speech. This results in some lines being cut off, but allows the user to hear the most recent relevant speech generated.

3.2.3 Failure: Gesture For Closing

Our initial gesture for closing simply involved detecting if the hand was located forward past some threshold. However, this led to the user closing objects accidentally simply because their hand was a little too far forward on the Leap sensor when hovering over it.

We decided to modify the gesture by changing it to detect when the user's hand moved forward. However, this gesture still had a critical flaw. Recall that our opening gesture involves the user grabbing and pulling back (as if they were pulling on a handle). When the user moves their hand backwards, it is natural for them to move their hand back into frame by moving their hand forwards. So immediately after they open a drawer, they might accidentally close it. Though, this didn't significantly impact the playability of the game as objects typically didn't need to remain open.

3.2.4 Failure: Gesture for Pressing

Another gesture that our game has difficulty recognizing is the press gesture, which involves the user pointing with their finger and moving it forward (as if they were pressing a small button). Our recognition for this gesture has two components: recognizing that the user is pointing, and recognition that the user's hand is moving forward. The latter component is where the issues arise from.

Our current gesture recognition uses the z coordinate of the palm velocity from the leap to detect forward motion. However, we noticed that the quick side-to-side motions used to navigate to different button positions on the safe somehow triggered the threshold, which led to accidental button presses. It is likely because these motions had slight forward motion and passed the threshold which was based on just the z coordinate, despite the low relative magnitude in the z-direction compared to the x or y directions. To combat this, we tried simply just increasing the threshold for the forward z velocity but this made pressing much more difficult, if not impossible even if we froze the cursor.

To improve this, we could try using a normalized vector to detect that the primary direction of movement is forward and combine that with our z-direction threshold for less false negative button presses and fewer false positive accidental presses.

4 Roadblocks and Pivots

4.1 Revamping Audio Commands

Originally, our project gave the user the option of a gesture or an audio command for each action. For example, in order to grab an item, the user could hover over the object they wish to grab and either say “grab” or make a grabbing motion with their hand. However, our audio recognition was quite poor and it would often take multiple tries for the system to recognize an audio command, resulting in a quite frustrating experience for the user.

The purpose of the multimodal system is to provide a more fluid and natural experience to the user. However, in many cases even keyboard input was preferable to audio input. Furthermore, we realized, while an interesting mode of interaction, it is not as natural to interact with the world with speech rather than physical manipulation. So we redesigned our system to de-emphasize usage of audio commands.

For commands with two options, we removed the audio command and emphasized using the gesture to perform the action. (The audio commands still exist in the game, but the user is not informed about this fact).

4.2 Speech Generation Interfering with Speech Recognition

During our first stages of implementing the ability of the user to speak with characters such as the Capybara to gain hints for the escape room, we realized a critical issue: the Capybara would react to hearing his own speech! Our conversation relied on detecting certain keywords spoken. So if any of these keywords were present in the capybara's response, he would trigger further dialogue from himself! We could perhaps try to design his dialogue to avoid these keywords, but this was very limiting to the dialogue design and not a surefire solution.

Instead, we made a modification to the speech recognition and speech generation system. Whenever our program generated speech, we would pause the speech recognition system until the speech finished. This successfully prevented the generated speech from triggering keywords in the speech recognition system. However, this came at the small cost of preventing the user from being able to interact with the system using audio commands, which can be slightly annoying in the cases when the Capybara is especially chatty. Luckily, the only reason the user should want to speak is to advance the conversation or to say “Goodbye” and leave the conversation. So we can defend this

drawback with the excuse that it is rude to interrupt or cut off another person while engaging in natural conversation with them.

This was also part of the reason we had to revamp the use of audio commands. The program had a verbal response to almost every action the user attempted, even if the action failed. So the user would have to wait for the program's speech to end before attempting to take any action that required speech input.

4.3 Clearing Up Confusion

For earlier iterations of the game, there was quite a bit of a difficulty curve to understanding the game. Users were constantly forgetting how to perform certain functionalities in the game and having to re-consult the instructions page. We partially alleviated this issue by providing an in-game instructions page the user could consult by saying "help", but this did not fix the critical issue of the steep learning curve.

However, we realized that users with prior experience with the game's commands typically had less difficulty interacting with the game. We realized that experience was a useful tool in learning how to perform commands. We decided that it was not sufficient to simply explain all the rules to the user beforehand on the quick instructions page. So we added a tutorial to the game which would gradually teach the user the various actions they can take with gestures and audio commands, and give them context for when these actions can be taken. The tutorial proved successful in teaching the user all of the relevant basic actions of our game through a fun little sub-game.

Another issue was that sometimes users took illogical actions, trying to open unopenable objects or grab ungrabbable objects. While we did have a coloring system that helped to indicate when certain actions were allowed or prohibited, the coloring system proved to be quite unintuitive. So we decided to revamp the cursor display. Instead of changing color, the cursor would change its image icon. If an object is grabbable, the cursor changes to a hand grabbing. If an object is openable, the cursor changes to a hand pulling on a handle. This also helps to reinforce the necessary gesture needed to take actions.

5 User Study

For our user study, we asked four participants to test and rate our system.

5.1 Qualitative Results

Qualitative results are the feedback that our participants gave during and after experiencing the virtual escape room. The key feedback of our system is summarized below:

- **Progress bar:** Users expressed that it would be nice to have some sort of score or metric to show the user how close they are to escaping the room to give them the motivation to keep playing if they're completely lost or don't know what to do next.
- **Game design/logic:** Users have complained that some of the hints are a little too clear, such as the safe code. They've suggested we make it more cryptic like an actual escape room that takes up to an hour to solve. They also think it'll be nice to add a time limit to give the user more motivation to finish faster and beat their friends, as well as add timed subgoals to the game for extra game credit. Finally, some users complained that it was annoying to have to redo everything if they're at the end and messed up there so an ability to undo an action that lets them go back to the last point they were correct instead of fully restarting would be nice.
- **Visual/Audio feedback:** Some users have indicated that it'll be interesting to add suspenseful music, as well as have a log of the messages since the AI's voice moves pretty fast. As for visual feedback, some users suggested we add a smoothing function to make the interactions less jittery.
- **Gesture recognition:** Users suggested we play around with thresholds a bit more to reduce false positives and look at the position data not just velocity at a point in time to recognize gestures. As for specific gestures to improve on, users have suggested we change the zoom-in gesture to something more intuitive than just a pinch from the index and thumb.
- **Future expansions**: Users have expressed that it'll be interesting to have different rooms as well as separate difficulty modes for each room to make it game even more exciting and versatile, that maybe having slightly different escape rooms every time so the same users can replay the game over and over again. Some even suggested extending our game to one of those "chart your own stories" kinds of games where there is no "right path" to solve the escape room, and you can choose which way you want to escape.

6 Performance

Overall, the system does fairly well in detecting speech and gestures. The accuracy of speech detection is within 1% error for the ones we experimented with and hand gestures 8.6% error.

6.1 Gesture Detection Accuracy

Number of trials per gesture: 10

Gesture	Detection Accuracy	Notes
Button Press	80%	Sometimes I press two adjacent buttons in a row if I just wanted to press one or accidentally press a button I'm hovering even though I'm not moving my hand forward.
Open	90%	Cursor jitters around quite a bit, easier if you freeze it. Sometimes it flies off if you don't freeze.
Zoom in	80%	Sometimes I pinch lightly and it doesn't detect it. I have to make sure other fingers are extended and not making a "grab" gesture. Rarely zooms in when I'm not pinching though.
Closing	100%	Used to only depend on z coordinate threshold, now cares about z forward velocity to detect forward motion.
Grab	90%	Sometimes it's hard to grab a small object if we don't freeze the cursor, which leads to failure in grabbing the item. However, if you freeze the cursor, the grab gesture has 100% accuracy.
Grab and drag (use an item on another item)	100%	Once the object is grabbed, the object is never dropped as you're dragging it unless the fist becomes unclenched. To release, as long as you extend all your fingers fully, the object will be released 100%.
Turn left/right	100%	As long as the leap is not smudged and in place (not glitching), turning left and right works 100% when the cursor goes off screen. Sometimes the leap just refuses to work properly and it's hard to get the cursor to go off screen, which is necessary for turning.
Average Accuracy:	91.4%	The accuracy could probably have been more if we had even more trials or were more careful for some of the

		gestures since some of the errors were human error and not the system such as grabbing when the cursor is no longer hovering on the object due to not freezing the cursor.

We've also greatly improved the performance for certain actions like pressing a button and opening smaller objects like the night table drawer by implementing the ability to freeze/unfreeze the cursor by saying the word "freeze" since more complicated gestures like pointing and pressing or grabbing and pull is much easier to do it the cursor is not moving side to side since the cursor can easily slip off the hovering object and make the gesture no longer detectable.

For the button pressing, if you don't freeze the cursor, it's very easy to press two buttons in a row accidentally that are adjacent to each other since when as your hand is moving forward to press a button also tends to drift side to side by accident as you're making the gesture, which gets pretty annoying since then you have to either press delete or enter and start over.

6.2 Speech Detection Accuracy

After testing out our system multiple times, we realized that certain words such as "on" or "open" are just harder to recognize than other words like "restart" or "skip" since those words are more unique/less common in terms of the phonemes. Also, we found out that it helped to give some of the commands some context to improve the recognition, such as saying "open the fridge", or "turn right" instead of just "open" or "right". Also, a lot of words just sound really similar so we incorporated alternative speech recognition words for each command to increase the accuracy even more, such as adding the word "grav" for "grab", and "scat" or "skit" to "skip". These were all words that were incorrectly recognized by Google Text-to-Speech when we tried to say the actual verbal commands.

We did some experiments on a couple of the voice commands to test the accuracy with and without context, with 8 trials for each of the categories for the command. The results are shown below:

Number of trials each (per command): 8

Voice command	Accuracy	Notes
"on"/"off"	100%	After adding words like "aunt" for "on", and "Off" or "IHOP" for "off", the accuracy became 100%

“left” / “right”	100%	After adding a bunch of words like “love”, “live”, “lap”, and “laugh” to the word “left”, the system’s accuracy for detecting increased to 100%.
“hello”	100%	We also added the alternatives of “hey” or “hi” if the user prefers saying those instead.
“open”	100%	As long as you wait enough in between voice commands, the open command is detected correctly almost 100% of the time, which we did by incorporating other words similar to “open” in the list of keywords we got by printing out the transcript.
Average Accuracy:	100%	As long as the user is physically close enough to the mic and enunciates the words clearly and semi-slowly, the accuracy is almost 100%.

6.3 Room for Improvement

There is great potential for combining the modalities of speech and gesture. One interesting potential addition could be the following scenario: The user hovers over an object on their inventory and says “Use this”. The object is highlighted in their inventory. They hover over an object in the scene and say “On that.” The game tells them they cannot use the item that way. So they hover over another object in the scene and say “On that.” They do not have to perform a command to reselect the object on the failure, and can continue to use it by saying “On that”, even if they move to another scene. The user’s hand no longer has to make frequent rapid movements back and forth in order to try to use a single object on all possibilities.

Another potential for combining multimodal capabilities is when the user is talking to the Capybara. They are confused about how to use an item in the inventory and would like a hint on how to use it. So they hover over the watermelon in the inventory and ask “What’s this?” or “How do I use this?” The Capybara responds with a hint like “That is a watermelon! It is my favorite food! If you were to feed it to me, I would allow you to pick me up and will help you to escape!” They do not need to say a clunky phrase like “What is the watermelon used for?”

Ultimately, there is a lot of potential to combine the multiple modalities to provide a more natural, fluid, and immersive experience to the user.

There are a few ways the system current capabilities can be improved. For one, the Webkit Speech Recognition leaves much to be desired. It can be somewhat slow at times, and is not very flexible. One major change we could make is shifting to a more reliable speech recognition software (implementing one may be too complex) which allows us more flexibility to improve speech recognition such as providing a list of expected keywords. As mentioned in previous sections, there are also various ways we could improve the recognition for certain gestures which can be a bit finicky such as pressing and closing, by performing more computation on the hand features provided by the Leap SDK.

7 Tools, packages, and libraries used

package/tool /library	Where is it available?	What does it do?	How well did it work?	Did it work out of the box? If not, what did you have to do to use it?
famo.us	http://deprecated.famous.org/	Javascript graphic/UI rendering library to help us render images and text surfaces to display elements in our escape room.	It is somewhat difficult to learn and is poorly documented, but it got the job done.	Yes
Web API Speech Recognition	https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API	Parses the user's speech/verbal commands from the laptop's built-in microphone into written transcripts that we process for speech recognition.	The speech to text was pretty inaccurate if the user just says one word and gives the command no context, such as saying "open" instead of "open the door", so we had to include a bunch of similar words for each command to increase accuracy.	Yes
Web API Speech Synthesis	https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API	Allows the system to generate verbal/speech feedback by turning sentences that we write into voice, synthesizing speech from the text we hard coded in our game in response to user actions.	The generated voices were fairly realistic and there were many options for us to assign to our multiple speaking characters. There was sufficient functionality to stop the speech or add callback functions.	Yes
LeapSDK	https://developer.leapmotion.com/	Detect palm positions and velocities and features of the hand such as outstretched fingers or grabbing	It worked great and provided great amounts of data making the gesture recognition process easier.	Yes

8 Collaboration

For building and testing the system, we were both more or less involved in almost everything. The major responsibilities for parts of the project are broken down roughly as follows, even though we both helped each other with each other's responsibilities:

8.1 Hizami

- Refactored code such as verbal/gesture commands in the processSpeech function and gesture detection function since there are multiple ways of performing certain actions such as saying "open" or by doing the grab and drag gesture.
- Logic and functions for transitioning between the walls of the room for turning or transitioning into zoomed in views of things like the safe or conversation with the capybara.
- Setting up skeleton code and backbone for the tutorial in Tutorial.js.
- Detecting forward and backward motion, grabbing, and pointing for pressing buttons.
- The files in the models folder: created backbone models and functions for View (walls), Item.js (items like cheese, dresser, or mousehole), Room.js (consists of 4 walls and additional views like zoomed safe), Inventory.js (manages items in inventory as well as getting rid of or adding items), and Capybara Conversation that displays the capybara, a large speech bubble, and smaller ones on the side as options to say for the user to the capybara.
- Prevent capybara and quokka from responding/talking to themselves.

8.2 Karen

- Game logic for using items on other items such as key on door, or cheese on mouse.
- Graphics + rendering items in the room such as the position and size of each object in each of the walls as well as downloading and adding sound effects to actions like door unlocking, or capybara eating watermelon, for example.
- Constructed the conversation/dialogue between user and quokka and capybara as well as updating the list of speech options for the user at each step.
- Added more things to the tutorial such as conversation with the quokka for learning actions like pressing buttons and zooming in, freezing/unfreezing, and switching light on/off.
- Added restart and skip tutorial command options + functions, as well as eliminating speech backlog by stopping the first phrase before playing the second one if the user does two consecutive actions very quickly one after the other.

- Arranged and created the zoomed in view of safe and programming buttons on the safe to register button presses.
- Generated the verbal feedback from the computer system/narrator in response to certain actions like feeding the mouse mashed potatoes after it already died, or giving the mouse cheese and the computer responding by saying that “the mouse gives you a small black key”, and updating the inventory/game state variables accordingly in response to the actions.