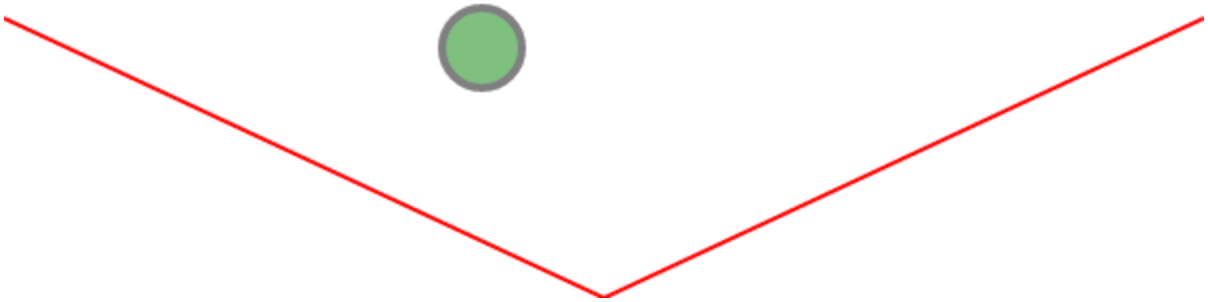
Challenges

Ship Deployment

Pointing at screen--normalize for leap motion

We noticed that the Leap Motion's area of detection is not a rectangle but rather an upside-down triangle.



There were several adjustments to make as a result of this. First, we needed to set the bottom of the battleship application to be enough above the Leap Motion sensor so that we can move the hand along the x-axis (side to side) while keeping the hand within the bounds of the Leap Motion sensor.

Second, we had to map the coordinates from the Leap Motion to the coordinates in the battleship application. Ignoring the z-coordinate because battleship is only two-dimensional, we mapped the x- and y- coordinates from the Leap Motion to the application as follows[1]:

$$x_{app} = (x_{leap} - Leap_{start})\frac{App_{range}}{Leap_{range}} + App_{start}$$

where

$$Leap_{range} = Leap_{end} - Leap_{start}$$

$$App_{range} = App_{end} - App_{start}$$

and correspondingly for the y-values.

We set the app_x range to be 0 to double the size of the board (550 * 2) to account for the horizontal space including the ships at the left, and the app_y range to be 0 to 1.5 times the size of the board (550 * 1.5) to allow the users hand to point slightly above and below the board.

[1] https://developer.leapmotion.com/documentation/csharp/devguide/Leap_Coordinate_Mapping.html

Smooth grabbing

We had trouble moving the ships sometimes due to our grabbing code. We set a variable "isGrabbing" to true if the grab strength was above a certain threshold. However, occasionally, the grab strength would be reported as below the threshold even when the user was still grabbing. Thus, we thought we could implement an average that would take into account the last 5 calls, and if this average was above the threshold, we'd set "isGrabbing" to true. This would be a way to smooth out this grabbing problem. We experimented additionally with several different window sizes greater than 5, but found that the smoothing did not produce a consistently better grabbing mechanism than using it without.

We incorporated smoothing as we described above into our final submission but it didn't noticeably improve the user experience. The fluctuating values seemed to come more from user error (accidentally releasing, or a sleeve being detected as part of the hand). The errors we wanted to account for, such as recording a small grab strength when the hand is rotated, were not solved for well by smoothing.

Determining roll threshold

We thought that we'd try to fix the behaviour when rotating the ship. We implemented two functions to do this.

First, we smoothed the roll values as well, to reduce the jumps from PI to -PI. We used the same window and smoothing algorithm as described above for grabbing.

Second, we clamped the rotation values of the ship to be between 0 and PI/2 radians clockwise. We did this because we encountered many problems when the hand when it jumped between the values PI/2 and -PI/2. Moreover, because the ship can only snap either horizontally or vertically, clamping these values allows for both of these cases while simplifying the motion. In addition, we scaled the hand.roll() values so that the user doesn't have to turn his or her hand uncomfortably to get the ship in a vertical orientation.

Further improvements would be to make the roll sensitive to righthandedness or lefthandedness (our current implementation only supports right hand, so if you're left handed I'm sorry), however these properties weren't included in the Leap Motion JS library we were provided (although it is available on Leap Motion!).

Snapping ships into place

When placing the ships, the ships would sometimes snap to a different location or snap to the side. We tried to implement the fix found on Piazza and found the the problem persisted. Thus, we decided not to try fixing it, since it seemed like a problem with the snapping mechanism code and not a bug that we introduced into the system.

Player's Turn

We had some issues detecting commands here. For some of our group, the speech recognition wouldn't recognize the word "fire", instead substituting "I are" or "iron". We also added "star" as a recognized command to start the game (in addition to "start" for better usability).

CPU's Turn

During the CPU's turn, we had a few issues recognizing commands. The speech recognition sometimes registered the commands with capital letters at the beginning of the word, so we had to force the transcript to lowercase in order to avoid any possibility of error. Other than that, sometimes the infrastructure of the game froze when the hand was no longer detected by the Leap Motion so game functionality was halted. Unfortunately, there is no good way to account for this user error because it is related to the functionality of the Leap Motion and not our implementation of the game mechanisms.

Extensions

Trash Talk

We decided to give our CPU a little bit of personality during the game, so we had him react to strings of hits and misses with some witty banter. The CPU can detect when either player has gone on strings of hits or strings of misses, and changes his response based on how many hits or misses have occurred in a row. In addition, the dialogue changes when a streak ends, based on how long the streak was. This makes the game more interesting to play because the CPU has things to say. It interesting to see how real we can make the CPU seem to people who don't know the inner workings of the code.

Detect Lying

In addition to trash talking, we gave our CPU the ability to detect if the player is lying. The player is given two opportunities to lie about the outcome of the the shot before the CPU stops listening to the player and starts checking directly if the CPU shot was a hit, miss, sunk ship, or game ending shot. We added additional dialogue for the CPU in cases of lying.

If the player gives a false game over, the CPU automatically marks them as untrustworthy. This makes for more interesting game play, so that the player can't win just by saying "Game Over" at the start of the game. However, one consequence of this is that if the player successfully lies about specific CPU hits, the CPU will never be able to win. This is because of the way the CPU shots are generated in the starter code. Because the CPU does not shoot a tile that it has already shot at, it is possible for the CPU to shoot all the tiles but not win the game.

A further improvement would be more ways to detect if the player is lying, related to the geometry of the board and previous and hits and misses, i.e. if the CPU was able to keep it's own "mental" model of the board so far and detect if the player-reported shot results so far are possible configurations or not.

Note: We came across a lot of problems when deploying to Athena (specifically that our voice recognition didn't work well or didn't work at all). We were successfully able to run it and make changes more easily by spinning up a simple HTTP Python server, which we recommend for future students who do this pset.