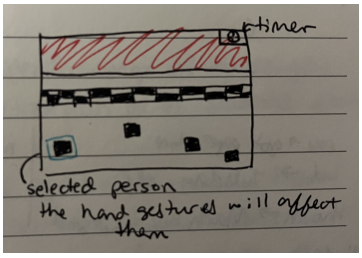
[Link to Slides Presentation]

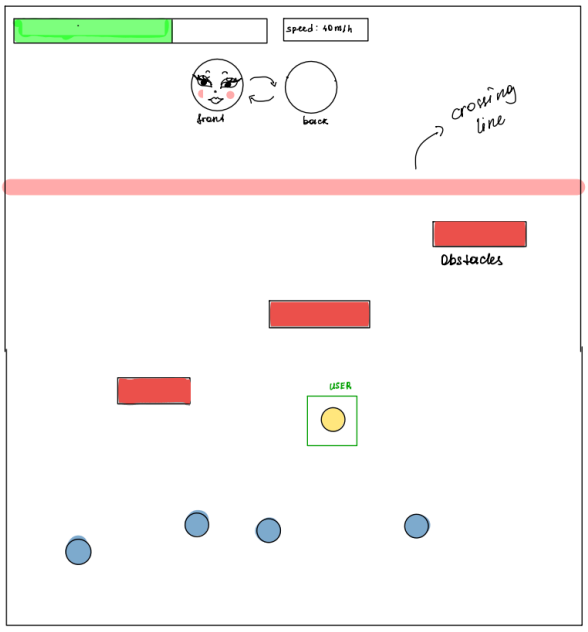
Project Overview:

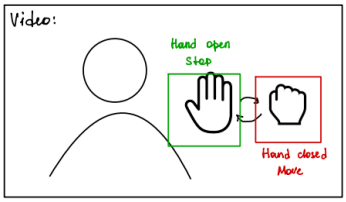
We plan on creating a game modeled off the red light and green light game, as depicted in Squid Game. The simulation includes participants on the screen that are meant to run towards the end line under a specific amount of time. The simulation also includes an evil doll which either faces the participants with either her back or her face and the participants are only allowed to run and move when the doll's back is to them. If a participant moves when the doll is facing them, they are eliminated from the game. The user is able to control their participant using hand gestures. The user selects one participant at a time and controls their movement. The participant runs when the user's hand is closed and stops as soon as the user opens their hand. If the user is caught moving a participant, the participant is eliminated! There may be some obstacles in the path which the participant might need to avoid. The user can make a participant speed up. We chose to display the timer using a bar filled with color and that color reduces (emptying the bar) as time passes. If the participant is not able to make it to the safety line before the bar is fully empty, they are eliminated and you, the user, loses the game.

Sketches:



Original Sketch





Refined Sketch

Technology requirements/technical risks:

- List the technologies you are planning to use for your project
  - Handtrack.js
    - Potential Risk ➔ Possible delay between how long it takes to detect the change in the user's gesture and the selected participant actually stopping. This could mistakenly eliminate the participant even though the user gave the gesture at the right time. We will account for this delay while we check to see if a participant is moving and have a minimal grace period.
  - Javascript, HTML, CSS

Technology Feasibility Test:

We decided to test whether we could control a player on the screen through the open and closed hand gestures, since this is a crucial component of our game. We were able to model this successfully and below is the webpage showing the same.

Feasibility Testing Link

Problems We Overcame:

- Face was detected as a closed hand when we made it so that only the hand open and hand closed gestures would have a bounding box around them
  - FIX => we had all the gestures be detected so that the face could be correctly identified as a face
- In the original code, the canvas was being redrawn regardless of whether there were any predictions (causing blank frame or flashing effect)

Code

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Red-Light Green-Light</title>
   <script
src="https://cdn.jsdelivr.net/npm/handtrackjs@latest/dist/handtrack.min.js"></script>
   <style>
       #video {
           display: none;
       }
       #canvas {
           width: 640px;
           height: 480px;
       }
   </style>
   <script>
       class Person {
           constructor(color, number, speed, x, y) {
               this._color = color;
               this._number = number;
               this._speed = speed;
               this._x = x;
               this._y = y;
               this._status = true;
               this._chosen = false;
           }

           move() {
               if(this._chosen) {
                   this._y--;
                   if (this._y < 0)
                       this._y = 480;
               }
           }

           draw(gc) {

               gc.beginPath();
               gc.strokeStyle = "#00ff00";
               gc.lineWidth = "3";
               gc.rect(this._x, this._y, 30, 30);
               gc.stroke();
           }

           setChosen(val) {
               this._chosen = val;
           }

       };
```

```
class Model {
    constructor(canvasId) {
        this.id = document.getElementById(canvasId);
        this.gc = this.id.getContext("2d");
        this.people = [];
        this.people.push(new Person("#0000ff", 1, 1, 100, 400));
        this.animationId = null;
        this.counter = 0;
        this._motion = false;

        this.counterInterval = setInterval(() => {
            this.counter++;
        }, 1000);

        this.start();

    }

    draw() {
        // Clear the canvas before drawing
        this.gc.clearRect(0, 0, this.id.width, this.id.height);

        this.people.forEach(person => person.draw(this.gc));
    }

    move() {
        this.people.forEach(person => person.move());
        this.draw();
    }

    animate() {
        this.move();
        this.draw();
        // console.log("requesting animation frame");

        this.animationId = requestAnimationFrame(() => this.animate());

    }

    start() {
        // Begin the animation loop
        this.animate();
    }

    stop() {
        // Stop the animation loop
        if (this.animationId) {
            cancelAnimationFrame(this.animationId);
            this.animationId = null;
        }
```

```
            }

            setMotion(val) {
                this._motion = val;
            }

        };

        class Main {
            constructor() {
                this._myModel = new Model("bgCanvas");
            }

            getModel() {return this._myModel}
        };

        window.onload = function() {
            main = new Main();
        }
    </script>


</head>
<body>
    <canvas id="canvas" width="640" height="480"></canvas>
    <canvas id="bgCanvas" width="640" height="480"></canvas>
    <video id="video" autoplay></video>
    <script>
        const video = document.getElementById('video');
        const canvas = document.getElementById('canvas');
        const context = canvas.getContext('2d');

        const modelParams = {
            flipHorizontal: false,
            outputStride: 16,
            imageScaleFactor: 1,
            maxNumBoxes: 20,
            iouThreshold: 0.2,
            scoreThreshold: .55,
            modelType: "ssd320fpnlite",
            modelSize: "small",
            bboxLineWidth: "2",
            fontSize: 17,
        };

        handTrack.startVideo(video).then(status => {
            if (status) {
                navigator.mediaDevices.getUserMedia({
                    video: {
                        width: { ideal: 640 },
                        height: { ideal: 480 }
                    }
```

```
                }).then(stream => {
                    video.srcObject = stream;
                    setInterval(runDetection, 10);
                }).catch(err => console.log(err));
            }
        });

        let model;
        handTrack.load(modelParams).then(lmodel => {
            model = lmodel;
        });

    function runDetection() {
        model.detect(video).then(predictions => {
            // Always draw the video feed on the canvas
            context.drawImage(video, 0, 0, video.width, video.height);
            const person = main.getModel().people[0];

            predictions.forEach(prediction => {
                const bbox = prediction.bbox;
                const label = prediction.label;

                // Draw bounding box
                context.strokeStyle = label === 'open' ? '#8B0000' : '#006400';
                context.lineWidth = 2;
                context.strokeRect(bbox[0], bbox[1], bbox[2], bbox[3]);

                // Add label
                context.font = '18px Arial';
                context.fillStyle = label === 'open' ? '#8B0000' : '#006400';
                context.fillText(label, bbox[0], bbox[1] - 10);

                // Change background color based on hand state
                if (label === 'open' || label === 'closed') {
                    document.getElementById("bgCanvas").style.backgroundColor = label ===
'open' ? '#8B0000' : '#006400';
                    person._chosen = label == "closed" ? true : false;
                }
            });
        });
    }

    </script>
</body>
</html>
```