PROJECT DEVELOPMENT-SPRINT 4 HTML CODING:
Loading

MediaPipe

Click here for more info

```
CSS: @keyframes spin { 0% { transform: rotate(0deg); } 100% { transform: rotate(360deg); } }
.abs { position: absolute; } a { color: white; text-decoration: none; &:hover { color: lightblue; } }
body { bottom: 0; font-family: 'Titillium Web', sans-serif; color: white; left: 0; margin: 0;
position: absolute; right: 0; top: 0; transform-origin: 0px 0px; overflow: hidden; } .container {
position: absolute; background-color: #596e73; width: 100%; max-height: 100%; } .input_video
{ display: none; position: absolute; top: 0; left: 0; right: 0; bottom: 0; &.selfie { transform:
scale(-1, 1); } } .input_image { position: absolute; } .canvas-container { display:flex; height:
100%; width: 100%; justify-content: center; align-items:center; } .output_canvas { width: 100%;
display: block; position: relative; left: 0; top: 0; } .logo { bottom: 10px; right: 20px; .title { color:
white; font-size: 28px; } .subtitle { position: relative; color: white; font-size: 10px; left: -30px;
top: 20px; } } .control-panel { position: absolute; left: 10px; top: 10px; } .loading { display: flex;
position: absolute; top: 0; right: 0; bottom: 0; left: 0; align-items: center; backface-visibility:
hidden; justify-content: center; opacity: 1; transition: opacity 1s; .message { font-size: x-large; }
.spinner { position: absolute; width: 120px; height: 120px; animation: spin 1s linear infinite;
border: 32px solid #bebebe; border-top: 32px solid #3498db; border-radius: 50%; } } .loaded
.loading { opacity: 0; } .shoutout { left: 0; right: 0; bottom: 40px; text-align: center; font-size:
24px; position: absolute; } JS: import DeviceDetector from "https://cdn.skypack.dev/device-
detector-js@2.2.10"; const mpHands = window; const drawingUtils = window; const controls =
window; const controls3d = window; // Usage: testSupport({client?: string, os?: string}[]) //
Client and os are regular expressions. // See: https://cdn.jsdelivr.net/npm/device-detector-
js@2.2.10/README.md for // legal values for client and os testSupport([ {client: 'Chrome'}, ]);
function testSupport(supportedDevices:{client?: string; os?: string;}[]) { const deviceDetector =
new DeviceDetector(); const detectedDevice = deviceDetector.parse(navigator.userAgent); let
isSupported = false; for (const device of supportedDevices) { if (device.client !== undefined) {
const re = new RegExp(`^${device.client}$`); if (!re.test(detectedDevice.client.name)) {
continue; } } if (device.os !== undefined) { const re = new RegExp(`^${device.os}$`); if
(!re.test(detectedDevice.os.name)) { continue; } } isSupported = true; break; } if (!isSupported) {
alert(`This demo, running on ${detectedDevice.client.name}/${detectedDevice.os.name}, ` + `is
not well supported at this time, continue at your own risk.`); } } // Our input frames will come
from here. const videoElement = document.getElementsByClassName('input_video')[0] as
HTMLVideoElement; const canvasElement =
document.getElementsByClassName('output_canvas')[0] as HTMLCanvasElement; const
controlsElement = document.getElementsByClassName('control-panel')[0] as
HTMLDivElement; const canvasCtx = canvasElement.getContext('2d')!; const config =
{locateFile: (file: string) => { return
`https://cdn.jsdelivr.net/npm/@mediapipe/hands@${mpHands.VERSION}/${file}`; }}; // We'll
add this to our control panel later, but we'll save it here so we can // call tick() each time the
graph runs. const fpsControl = new controls.FPS(); // Optimization: Turn off animated spinner
after its hiding animation is done. const spinner = document.querySelector('.loading')! as
HTMLDivElement; spinner.ontransitionend = () => { spinner.style.display = 'none'; }; const
```

```
landmarkContainer = document.getElementsByClassName( 'landmark-grid-container')[0] as
HTMLDivElement; const grid = new controls3d.LandmarkGrid(landmarkContainer, {
connectionColor: 0xCCCCCC, definedColors: [{name: 'Left', value: 0xffa500}, {name: 'Right',
value: 0x00ffff}], range: 0.2, fitToGrid: false, labelSuffix: 'm', landmarkSize: 2,
numCellsPerAxis: 4, showHidden: false, centered: false, }); function onResults(results:
mpHands.Results): void { // Hide the spinner. document.body.classList.add('loaded'); // Update
the frame rate. fpsControl.tick(); // Draw the overlays. canvasCtx.save(); canvasCtx.clearRect(0,
0, canvasElement.width, canvasElement.height); canvasCtx.drawImage( results.image, 0, 0,
canvasElement.width, canvasElement.height); if (results.multiHandLandmarks &&
results.multiHandedness) { for (let index = 0; index < results.multiHandLandmarks.length;
index++) { const classification = results.multiHandedness[index]; const isRightHand =
classification.label === 'Right'; const landmarks = results.multiHandLandmarks[index];
drawingUtils.drawConnectors( canvasCtx, landmarks, mpHands.HAND_CONNECTIONS,
{color: isRightHand ? '#00FF00' : '#FF0000'}); drawingUtils.drawLandmarks(canvasCtx,
landmarks, { color: isRightHand ? '#00FF00' : '#FF0000', fillColor: isRightHand ? '#FF0000' :
'#00FF00', radius: (data: drawingUtils.Data) => { return drawingUtils.lerp(data.from!.z!, -0.15,
.1, 10, 1); } }); } } canvasCtx.restore(); if (results.multiHandWorldLandmarks) { // We only get
to call updateLandmarks once, so we need to cook the data to // fit. The landmarks just merge,
but the connections need to be offset. const landmarks =
results.multiHandWorldLandmarks.reduce( (prev, current) => [...prev, ...current], []); const
colors = []; let connections: mpHands.LandmarkConnectionArray = []; for (let loop = 0; loop <
results.multiHandWorldLandmarks.length; ++loop) { const offset = loop *
mpHands.HAND_CONNECTIONS.length; const offsetConnections =
mpHands.HAND_CONNECTIONS.map( (connection) => [connection[0] + offset,
connection[1] + offset]) as mpHands.LandmarkConnectionArray; connections =
connections.concat(offsetConnections); const classification = results.multiHandedness[loop];
colors.push({ list: offsetConnections.map((unused, i) => i + offset), color: classification.label, });
} grid.updateLandmarks(landmarks, connections, colors); } else { grid.updateLandmarks([]); } }
const hands = new mpHands.Hands(config); hands.onResults(onResults); // Present a control
panel through which the user can manipulate the solution // options. new controls
.ControlPanel(controlsElement, { selfieMode: true, maxNumHands: 2, modelComplexity: 1,
minDetectionConfidence: 0.5, minTrackingConfidence: 0.5 }) .add([ new
controls.StaticText({title: 'MediaPipe Hands'}), fpsControl, new controls.Toggle({title: 'Selfie
Mode', field: 'selfieMode'}), new controls.SourcePicker({ onFrame: async (input:
controls.InputImage, size: controls.Rectangle) => { const aspect = size.height / size.width; let
width: number, height: number; if (window.innerWidth > window.innerHeight) { height =
window.innerHeight; width = height / aspect; } else { width = window.innerWidth; height =
width * aspect; } canvasElement.width = width; canvasElement.height = height; await
hands.send({image: input}); }, }), new controls.Slider({ title: 'Max Number of Hands', field:
'maxNumHands', range: [1, 4], step: 1 }), new controls.Slider({ title: 'Model Complexity', field:
'modelComplexity', discrete: ['Lite', 'Full'], }), new controls.Slider({ title: 'Min Detection
Confidence', field: 'minDetectionConfidence', range: [0, 1], step: 0.01 }), new controls.Slider({
title: 'Min Tracking Confidence', field: 'minTrackingConfidence', range: [0, 1], step: 0.01 }), ])
.on(x => { const options = x as mpHands.Options; videoElement.classList.toggle('selfie',
options.selfieMode); hands.setOptions(options); }); OUTPUT:
```