```
/**
 * Pose Detection Application
 * Using TensorFlow.js and Teachable Machine
 * Created: January 2024
 */

// Model URL from Teachable Machine

/****************************************
* Paste your teachable machine link below
*
*/
const URL = "your teachable machine link goes here";









let model, webcam, ctx, labelContainer, maxPredictions;

// State variables for pose detection
let explosionActive = false;
let pose3ExplosionActive = false;
let explosionSound = new Audio('explsn.mp3');
let pose1Triggered = false;
let pose2Triggered = false;
let pose3FirstWindowTriggered = false;
let pose3SecondWindowTriggered = false;
let pose4Triggered = false;
let pose5Triggered = false;

/**
 * Initialize the application
 */
async function init() {
    const modelURL = URL + "model.json";
    const metadataURL = URL + "metadata.json";

    const video = document.getElementById('instructionVideo');
    video.volume = 0.4;
    try {
        model = await tmPose.load(modelURL, metadataURL);
        maxPredictions = model.getTotalClasses();

        const width = 600;
        const height = 600;
        const flip = true;
        webcam = new tmPose.Webcam(width, height, flip);
        await webcam.setup();
        await webcam.play();
        window.requestAnimationFrame(loop);

        const canvas = document.getElementById("canvas");
        canvas.width = width;
        canvas.height = height;
        ctx = canvas.getContext("2d");
        labelContainer = document.getElementById("label-container");
        for (let i = 0; i < maxPredictions; i++) {
            labelContainer.appendChild(document.createElement("div"));
        }
    } catch (error) {
        console.error("Error initializing model:", error);
    }
}

async function loop(timestamp) {
    webcam.update();
    await predict();
    window.requestAnimationFrame(loop);
}

function playExplosionSound() {
    const newSound = new Audio('explsn.mp3');
    newSound.volume = 1.0;
    newSound.play();
}

async function predict() {
    try {
        const { pose, posenetOutput } = await model.estimatePose(webcam.canvas);
        const prediction = await model.predict(posenetOutput);
        const video = document.getElementById('instructionVideo');
        for (let i = 0; i < maxPredictions; i++) {
            const classPrediction =
                prediction[i].className + ": " + prediction[i].probability.toFixed(2);
            labelContainer.childNodes[i].innerHTML = classPrediction;

            // Check for different poses
            checkPose1(prediction[0], video);
            checkPose2(prediction[i], video);
            checkPose3(prediction[i], video);
            checkPose4(prediction[i], video);
            checkPose5(prediction[i], video);
        }

        drawPose(pose, explosionActive);
    } catch (error) {
        console.error("Error in predict:", error);
    }
}


/****************************
*
* You can edit your video "times" in the spaces below.
* Note that if you are doubling up on a particular pose,
* see pose three code.  You will have to most likely modify pose three
* code otherwise.
*
*/

function checkPose1(prediction, video) {
    if (prediction.className === "pose 1" &&
        prediction.probability > 0.8 &&
        video.currentTime >= 0.9 &&
        video.currentTime <= 3 &&
        !pose1Triggered &&
        !explosionActive) {
        explosionActive = true;
        pose1Triggered = true;
        playExplosionSound();
        setTimeout(() => {
            explosionActive = false;
        }, 300);
    }
}
```

```
function checkPose2(prediction, video) {
    if (prediction.className === "pose 2" &&
        prediction.probability > 0.8 &&
        video.currentTime >= 5.5 &&
        video.currentTime <= 7.5 &&
        !pose2Triggered &&
        !explosionActive) {
        explosionActive = true;
        pose2Triggered = true;
        playExplosionSound();
        setTimeout(() => {
            explosionActive = false;
        }, 300);
    }
}

function checkPose3(prediction, video) {
    if (prediction.className === "pose 3" &&
        prediction.probability > 0.8) {
        if (video.currentTime >= 11.5 && video.currentTime <= 13 &&
            !pose3FirstWindowTriggered && !pose3ExplosionActive) {
            pose3ExplosionActive = true;
            pose3FirstWindowTriggered = true;
            playExplosionSound();
            setTimeout(() => {
                pose3ExplosionActive = false;
            }, 300);
        } else if (video.currentTime >= 17.5 && video.currentTime <= 19.5 &&
            !pose3SecondWindowTriggered && !pose3ExplosionActive) {
            pose3ExplosionActive = true;
            pose3SecondWindowTriggered = true;
            playExplosionSound();
            setTimeout(() => {
                pose3ExplosionActive = false;
            }, 300);
        }
    }
}

function checkPose4(prediction, video) {
    if (prediction.className === "pose 4" &&
        prediction.probability > 0.8 &&
        video.currentTime >= 15.5 &&
        video.currentTime <= 16.6 &&
        !pose4Triggered &&
        !explosionActive) {
        explosionActive = true;
        pose4Triggered = true;
        playExplosionSound();
        setTimeout(() => {
            explosionActive = false;
        }, 300);
    }
}

function checkPose5(prediction, video) {
    if (prediction.className === "pose 5" &&
        prediction.probability > 0.8 &&
        video.currentTime >= 19.5 &&
        !pose5Triggered &&
        !explosionActive) {
        explosionActive = true;
        pose5Triggered = true;
        playExplosionSound();
        setTimeout(() => {
            explosionActive = false;
        }, 300);
    }
}

function drawPose(pose, explode) {
    const shouldExplode = explode || pose3ExplosionActive;
    if (webcam.canvas) {
        ctx.drawImage(webcam.canvas, 0, 0);
        if (pose) {
            const minPartConfidence = 0.5;
            if (shouldExplode) {
                pose.keypoints.forEach(keypoint => {
                    if (keypoint.score > minPartConfidence) {
                        const scale = 3;
                        ctx.beginPath();
                        ctx.arc(keypoint.position.x, keypoint.position.y, 10 * scale, 0, 2 * Math.PI);
                        ctx.fillStyle = '#FF0000';
                        ctx.fill();
                    }
                });
            } else {
                tmPose.drawKeypoints(pose.keypoints, minPartConfidence, ctx);
                tmPose.drawSkeleton(pose.keypoints, minPartConfidence, ctx);
            }
        }
    }
}

async function playInstructionVideo() {
    const video = document.getElementById('instructionVideo');
    const videoContainer = video.parentElement;

    video.addEventListener('timeupdate', () => {
        const minutes = Math.floor(video.currentTime / 60);
        const seconds = Math.floor(video.currentTime % 60);
        document.getElementById('videoTime').textContent =
            `Time: ${minutes}:${seconds.toString().padStart(2, '0')}`;
    });

    const videoCanvas = document.createElement('canvas');
    videoCanvas.id = 'poseCanvas';
    videoCanvas.style.position = 'absolute';
    videoCanvas.style.left = '0';
    videoCanvas.style.top = '0';
    videoCanvas.width = 600;
    videoCanvas.height = 450;

    videoContainer.style.position = 'relative';
    videoContainer.appendChild(videoCanvas);
    const videoCtx = videoCanvas.getContext('2d');

    video.play();

    async function processFrame() {
        if (!video.paused && !video.ended) {
            try {
                const { pose, posenetOutput } = await model.estimatePose(video);
                videoCtx.clearRect(0, 0, videoCanvas.width, videoCanvas.height);

                if (pose) {
                    tmPose.drawKeypoints(pose.keypoints, 0.6, videoCtx);
                    tmPose.drawSkeleton(pose.keypoints, 0.6, videoCtx);
                }
            } catch (error) {
                console.error('Pose detection error:', error);
            }
            requestAnimationFrame(processFrame);
        }
    }

    if (model) {
        processFrame();
    } else {
        console.log("Please start webcam first to load the model");
    }
}

function stopInstructionVideo() {
    const video = document.getElementById('instructionVideo');
    video.pause();
    video.currentTime = 0;
    const canvas = video.parentElement.querySelector('canvas');
    if (canvas) {
        canvas.remove();
    }
    pose1Triggered = false;
    pose2Triggered = false;
    pose3FirstWindowTriggered = false;
    pose3SecondWindowTriggered = false;
    pose4Triggered = false;
    pose5Triggered = false;
}

function stopWebcam() {
    if (webcam) {
        webcam.stop();
        const canvas = document.getElementById("canvas");
        const ctx = canvas.getContext("2d");
        ctx.clearRect(0, 0, canvas.width, canvas.height);
    }
}
```