```
//global for the controls and input
var controls = null;
//store visualisations in a container
var vis = null;
//variable for the p5 sound object
var sound = null;
//variable for change track and previous track function
var currentSongIndex = 0;
//variable for p5 fast fourier transform
var fourier;
// backgrounds array
var backgroundImages = [];
//variable for amplitude
var amplitude;
// array of song nams to load for changing tracks
var songs = [
  'TokyoBossa.mp3',
  'SwimmingBird.mp3',
  'CanYouHoldMe.mp3',
        '90sFreestyle.mp3'
];

function preload(){
        sound = loadSound('assets/TokyoBossa.mp3');
        for (let i = 0; i < songs.length; i++) {
    loadSound('assets/' + songs[i]);
  }
        backgroundImages.push(loadImage('Images/Window.JPG'));
  backgroundImages.push(loadImage('Images/headphones.jpg'));
  backgroundImages.push(loadImage('Images/Kakashi.png'));
  backgroundImages.push(loadImage('Images/Sky.jpg'));
```

```
    backgroundImages.push(loadImage('Images/lofiGirl.png'));
        backgroundImages.push(loadImage('Images/lofitrain.jpeg'));
        backgroundImages.push(loadImage('Images/Skyforeground.png'));
}

function setup(){
        var canvas = createCanvas(windowWidth,windowHeight);
        canvas.parent('music__vis');
        controls = new ControlsAndInput();
         //instantiate the fft object
        fourier = new p5.FFT();
        amplitude = new p5.Amplitude();
         //create a new visualisation container and add visualisations
         vis = new Visualisations();
         vis.add(new Spectrum());
         vis.add(new WavePattern());
         vis.add(new Needles());
         vis.add(new WaveTriangle());
         vis.add(new ParticlePattern());
         vis.add(new Cubes());
}

function draw(){
        background(0);
        //draw the selected visualisation
        vis.selectedVisual.draw();
        //draw the controls on top.
        controls.draw();
        // Draw circles that disapear after media button has been pressed
        controls.playbackButton.update();
}
```

```
function mouseClicked(){
	controls.mousePressed();
}

function keyPressed(){
	controls.keyPressed(keyCode);
}

//when the window has been resized. Resize canvas to fit
//if the visualisation needs to be resized call its onResize method
function windowResized(){
	resizeCanvas(windowWidth, windowHeight);
	if(vis.selectedVisual.hasOwnProperty('onResize')){
		vis.selectedVisual.onResize();
	}
}


//container function for the visualisations
function Visualisations(){
	//array to store visualisations
	this.visuals = [];
	//currently selected vis. set to null until vis loaded in
	this.selectedVisual = null;

	//add a new visualisation to the array
	//@param vis: a visualisation object
	this.add = function(vis){
		this.visuals.push(vis);
		//if selectedVisual is null set the new visual as the
```

```
		//current visualiation
		if(this.selectedVisual == null){
			this.selectVisual(vis.name);
		}
	};

	//select a visualisation using it name property
	//@param visName: name property of the visualisation
	this.selectVisual = function(visName){
		for(var i = 0; i < this.visuals.length; i++){
			if(visName == this.visuals[i].name){
				this.selectedVisual = this.visuals[i];
			}
		}
	};
}



//displays and handles clicks on the playback button.
function PlaybackButton(){
  // these are properties to the play button
  this.x = 60;
  this.y = 20;
  this.width = 20;
  this.height = 20;
  this.playing = false;

  // these are properties for the circles that apear when a media button is pressed.
	this.circleTimer = 0;
  this.circleDuration = 10; // Duration in frames
```

```
this.circleTimer2 = 0;
this.circleDuration2 = 10;
this.circleTimer3 = 0;
this.circleDuration3 = 10;

// These properties for track change button
this.trackChangeButtonWidth = 25;
this.trackChangeButtonHeight = 25;

// properties for the previous button
this.previousButtonWidth = 25;
this.previousButtonHeight = 25;

// properties for the track duration slider
this.sliderWidth = 120;
this.sliderHeight = 20;

// created a volume slider in the dom
this.volume = createSlider(0,1,0.5, 0.01);
this.volume.parent('sliders');
let currentSongName = '';
let playButtonPressed = false;

this.draw = function() {
  //draws media button pill container
        noStroke();
  fill(176,224,230,100);
  arc(this.x + 235, this.y + 33, 78, 90, -HALF_PI, HALF_PI, OPEN);
  arc(this.x - 20, this.y + 33, 78, 90, HALF_PI, -HALF_PI, OPEN);
  rect(this.x - 20,this.y -12,255,90);
```

```
// draws circle animations around media buttons when pressed
            if (this.circleTimer > 0) {
                fill(0,139,138,100);
  ellipse(this.x + this.width / 2, this.y + this.height / 2, 35, 35);
}
if (this.circleTimer2 > 0) {
  fill(0,139,138,100);
  ellipse(this.x + 50, this.y + 9, this.trackChangeButtonWidth, this.trackChangeButtonHeight);
}
if (this.circleTimer3 > 0) {
  fill(0,139,138,100);
  ellipse(this.x - 35, this.y + 9, this.previousButtonWidth, this.previousButtonHeight);
}

// Draw playback button
if (this.playing) {
                fill(255,235,205);
  rect(this.x, this.y, this.width/2 - 2, this.height);
  rect(this.x + (this.width/2 + 2), this.y, this.width/2 - 2, this.height);
} else {
                fill(255,235,205);
  triangle(this.x, this.y, this.x + this.width, this.y + this.height/2, this.x, this.y+this.height);
}

// draw track change button
            fill(255,235,205);
triangle(this.x + 43, this.y + 3, this.x + 55,
                    this.y + 10 , this.x + 43, this.y + 17);
            rect(this.x + 55, this.y + 4, 3, 12);

            //draw previus track button
```

```
            triangle(this.x -28, this.y +3 , this.x -40,
                        this.y + 10, this.x - 28, this.y + 17);
    rect(this.x - 43, this.y +4, 3, 12);

    //draw track time slider
            fill(0,139,138);
            rect(this.x + 100, this.y, this.sliderWidth, this.sliderHeight);
            ellipse(this.x + 100, this.y +10, 20, this.sliderHeight);
            ellipse(this.x +220, this.y+10, this.sliderHeight);
            fill(255,235,205);
    //map the song duration to the slider
            let sliderPosition = map(sound.currentTime(), 0, sound.duration(), this.x +100, (this.x
+100) + this.sliderWidth);
            ellipse(sliderPosition, this.y + this.sliderHeight / 2, 15, 15);
            // draw time remaining to the right of the slider
            let timeRemaining = sound.duration() - sound.currentTime();
            let minutes = floor(timeRemaining / 60);
            let seconds = nf(floor(timeRemaining % 60), 2);
            textAlign(LEFT);
            textSize(12);
            fill(255);
            text(minutes + ':' + seconds, (this.x +100) + this.sliderWidth + 15, (this.y +
this.sliderHeight / 2) +4);

    //Volume is added to slider and style
    sound.setVolume(this.volume.value());
    this.volume.style('width', '150px'); // Set width
    this.volume.style('height', '20px'); // Set height

    //draw song names
    fill(255);
```

```
    textSize(13);
    stroke(20);
    textAlign(LEFT);
    text(currentSongName, this.x + 50, this.y + 70);
  };

  //track slider hitcheck function , decided to write it in its own function to is if it was an easier way to be called
  // into to biger hitcheck function. I might be a better way to compartmentalize if this was done to all of the media buttons.
        this.sliderHitCheck = function() {
    return (mouseX > this.x + 100 && mouseX < (this.x +100) + this.sliderWidth &&
        mouseY > this.y && mouseY < this.y + this.sliderHeight);
  };

  //hitcheck functions for media buttons
        this.hitCheck = function() {
    //hitcheck for playback button
    if (mouseX > this.x && mouseX < this.x + this.width && mouseY > this.y && mouseY < this.y + this.height) {
      if (!playButtonPressed) {
        currentSongName = songs[currentSongIndex];
        playButtonPressed = true;
      }

      if (sound.isPlaying()) {
        sound.pause();
        // add animation if clicked
                                this.circleTimer = this.circleDuration;
      } else {
        sound.loop();
        //animation for everytime it is clicked
```

```
                this.circleTimer = this.circleDuration;
        }
        this.playing = !this.playing;
        return true;
        //track change button hitcheck
    } else if (mouseX > (this.x +50)-10 && mouseX < (this.x +50) + this.trackChangeButtonWidth -10&&
        mouseY > (this.y + 9) -10 && mouseY < (this.y + 9) + this.trackChangeButtonHeight -10) {
        // if button is ckicked then call changeTrack
        changeTrack();
        // again add animation
                this.circleTimer2 = this.circleDuration2;
        return true;
        //previous track hitcheck
    }else if (mouseX > (this.x -35) && mouseX -10 < (this.x -35) + this.previousButtonWidth-10 &&
        mouseY > (this.y + 9) -10 && mouseY < (this.y + 9) + this.previousButtonHeight-10){
        // if button is clicked then call previousTrack
                previousTrack();
        // animation
                this.circleTimer3 = this.circleDuration3;
                return true;
            } else if (this.sliderHitCheck()) {
        // Update the playback track position based on the slider position
        let newPosition = map(mouseX, this.x + 100, (this.x +100) + this.sliderWidth, 0,
sound.duration());
        // I used sound.jump to jump to new track position
        sound.jump(newPosition);
        return true;
        }
    return false;
  };
```

```
//Function to change track
function changeTrack() {
  currentSongIndex = (currentSongIndex + 1) % songs.length;
  sound.stop();
  sound = loadSound('assets/' + songs[currentSongIndex], function() {
    sound.loop();
    currentSongName = songs[currentSongIndex];
  });
}

//Function to go back to previous track
      function previousTrack(){
  currentSongIndex = (currentSongIndex - 1 + songs.length) % songs.length;
  sound.stop();
  sound = loadSound('assets/' + songs[currentSongIndex], function() {
    sound.loop();
    currentSongName = songs[currentSongIndex];
      });
}

// update media button animation timer
      this.update = function() {
            if (this.circleTimer > 0) {
                  this.circleTimer--;
            }
            if (this.circleTimer2 > 0) {
            this.circleTimer2--;
        }
            if (this.circleTimer3 > 0) {
            this.circleTimer3--;
        }
```

```
        };
}


//Functions for controls
function ControlsAndInput() {

  this.menuDisplayed = false;
  this.playbackButton = new PlaybackButton(); // Create a new PlaybackButton instance

  //global variable properties for new UI menu. I drew circles with the number of the visualizer inside.
  var circleDiameter = 25;
  var circleSpacing = 40; // for even spacing between circles
  var startX = 45; //starting x for the first circle

  // mouse pressed function calls playback button hitcheck and menu hitcheck
  this.mousePressed = function() {
    if (this.playbackButton.hitCheck()) {
    }
    //menu
    if(this.menuHitcheck()){
    }
  };

  // key press function for number keys and spacebar
  this.keyPressed = function(keycode) {
    //use the space bar to play or pause music
    if (keycode == 32) {
      if (sound.isPlaying()) {
        sound.pause();
      } else {
```

```
        sound.loop();
      }
      this.playbackButton.playing = !this.playbackButton.playing; // to update the playback button
    }
    // assigning number keys to visualizations remains the same
    if (keycode > 48 && keycode < 58) {
      var visNumber = keycode - 49;
      vis.selectVisual(vis.visuals[visNumber].name);
    }
};

//draws new media buttons including slider, also draws new menu.
this.draw = function() {
    push();
    // media controls
    this.playbackButton.draw();
    // menu
    this.menu();
 pop();
};

this.menu = function() {
    // drawing properties for new menu
    noStroke();
    textSize(15);
    for (let i = 0; i < vis.visuals.length; i++) {
      let xPos = startX + i * circleSpacing;
      let yPos = 60;

      // if statement to check which music visual is selected and indicate it with a change in color.
      if (vis.selectedVisual.name === vis.visuals[i].name) {
```

```
        fill(0,139,138,);; // will change the fill color of the button
      } else {
        fill(255,235,205); // Otherwise, keep it this color
      }

      //circles
      ellipse(xPos, yPos, circleDiameter, circleDiameter);
      //to display the number inside the circle
      fill(0);
      textAlign(CENTER, CENTER);
      text(i + 1, xPos, yPos);
    }
  };

  // menu hitcheck properties
  this.menuHitcheck = function(){
      // for loop and if statement to check if the menu elements have been clicked.
      for (let i = 0; i < vis.visuals.length; i++) {
        let xPos = startX + i * circleSpacing;
        let yPos = 60;

        if (dist(mouseX, mouseY, xPos, yPos) < circleDiameter / 2) {
          // change the visualization to the one selected and break to get out of the loop.
          vis.selectVisual(vis.visuals[i].name);
          break;
        }
      }
  }
}

function Spectrum(){
```

```
        this.name = "spectrum";
// tweaked and changed this code to add a different color spectrum.
        this.draw = function(){
                push();
                image(backgroundImages[1], 0, 0, width, height);
                //experimenting with HSB color
                colorMode(HSB, 360, 100, 100);
                var spectrum = fourier.analyze();
                noStroke();
                var barHeight = height / spectrum.length;

        for (var i = 0; i < spectrum.length; i++) {
        var h = map(spectrum[i], 0, 255, 0, height);
        var y = map(i, 0, spectrum.length, 0, height - barHeight);

    // To map the colors to the amplitude
          var hueValue = map(spectrum[i], 0, 255, 280, 200);
    var saturationValue = 100;
    var brightnessValue = 100;

    // Set the fill color based on HSB values
    fill(hueValue, saturationValue, brightnessValue);
          //drawbars
    rect(0, y, h, barHeight);
                }
                pop();
        };
}


// I experimented and changed this code to make the wave rotate in a circle. Also added another wave.
```

```
function WavePattern() {
    // Visualization name
    this.name = "wavepattern";

    // This section was coded using a youtube video that teaches about cos, sin and radius.
    // most of it came about from experementing different code and posiblilites
    // Variables for wave rotation
    let angle1 = 0;
    let angle2 = 0;
    let rotationSpeed = 0.0043; // This controls rotations speed. I initially wanted a slow rotation. Then I realized a quick rotation makes it look like a whole circle of waves.

    // Draw the wave form within a circle, reffrence to youtube video then applied to original wave from template.
    this.draw = function() {
      push();
      image(backgroundImages[2], 0, 0, width, height);
      colorMode(HSB, 360, 250, 300);
      noFill();

      translate(width / 2, height / 2); // moved the origin to the center of the canvas so it will spin around it.
    //initially started with one wave but added two rotating in counter direction for a better look.
    // First wave (main wave)
      stroke(180, 200, 300);
      strokeWeight(4);
      beginShape();
      var wave = fourier.waveform();
      for (var i = 0; i < wave.length; i += 5) {
            var radius = height * 0.3;
            var x = cos(angle1) * (radius + map(wave[i], -1, 1, 0, radius));
            var y = sin(angle1) * (radius + map(wave[i], -1, 1, 0, radius));
```

```
            vertex(x, y);
            angle1 += rotationSpeed;
        }
        endShape();

        // Second wave (opposite direction)
        stroke(200, 10, 300); // Different color for the second wave
        beginShape();
        for (var j = 0; j < wave.length; j += 5) {
            var radius2 = height * 0.25; // Different radius for the second wave, made this smaller to allow better visibility of waves.
            var x2 = cos(angle2) * (radius2 + map(wave[j], -1, 1, 0, radius2));
            var y2 = sin(angle2) * (radius2 + map(wave[j], -1, 1, 0, radius2));
            vertex(x2, y2);
            angle2 -= rotationSpeed; // Opposite rotation direction for the second wave. not noticible for a faster rotation speed, but noticible for slower rotation speed.
        }
        endShape();
        pop();
      };
}


//Function for waveTriagle pattern.
// This uses the activity video from the lecture on creating horizontal lines. I added to the code and change the colors.
function WaveTriangle() {
  this.name = "wavetriangle";
  this.output = [];
  this.speed = 0.7;

  // Function to update dimensions based on window size
```

```
this.updateDimensions = function() {
  this.startX = width / 5;
  this.endY = height / 5;
  this.startY = height - this.endY;
  this.spectrumWidth = (width / 5) * 3;
};

// Call updateDimensions when the window is resized
this.windowResized = function() {
  this.updateDimensions();
};

this.addWave = function() {
  let w = fourier.waveform();
  let outputWave = [];
  let smallScale = 3;
  let bigScale = 40;

  // x cordinate for the base of the triangle , to match the horisontal lines to the starting point.
  let baseX = this.startX + this.spectrumWidth / 2;
  // width of the triangles base, so the lines are not wider than the triangle base
  let baseWidth = this.spectrumWidth;

  for (let i = 0; i < w.length; i++) {
    if (i % 20 == 0) {
      // keep the lines in the right place after window resize
      let x = map(i, 0, 1024, baseX - baseWidth / 2, baseX + baseWidth / 2);
      // calculate y cordinate
      let y;
      if (i < 1024 * 0.25 || i > 1024 * 0.75) {
        y = map(w[i], -1, 1, -smallScale, smallScale);
```

```
        } else {
            y = map(w[i], -1, 1, -bigScale, bigScale);
        }
        //adjust y-coordinate to start at the base of the triangle
        y += this.startY;
        outputWave.push({ x: x, y: y });
      }
    }
    this.output.push(outputWave);
};

  // Draw the pattern
  this.draw = function() {

    push();
    this.updateDimensions();
    image(backgroundImages[0], 0, 0, width, height);
    colorMode(HSB, 360, 250, 300);
    stroke(147, 112, 219);
    strokeWeight(2);
    fill(170, 224, 208, 0.5);

    // Draw triangle
    beginShape();
    vertex(this.startX + this.spectrumWidth / 2, this.endY);
    vertex(this.startX + this.spectrumWidth, this.startY);
    vertex(this.startX, this.startY);
    endShape(CLOSE);

    // Draw horizontal line waves
    if (frameCount % 20 == 0) {
```

```
      this.addWave();
    }
    for (let i = 0; i < this.output.length; i++) {
      let o = this.output[i];
      beginShape();
      for (let j = 0; j < o.length; j++) {
        o[j].y -= this.speed;
        vertex(o[j].x, o[j].y);
      }
      endShape();
      if (o[0].y < this.endY) {
        this.output.splice(i, 1);
      }
    }

    // Map amplitude to colors, refrence to p5.js website on lerpColor
    let level = amplitude.getLevel();
    let mappedLevel = map(level, 0, 1, 0, 50);
    let colorLevel = map(level, 0, 1, 0, 1);
    let colorStart = color(221, 200, 300);
    let colorEnd = color(330, 200, 300);
    let changeColor = lerpColor(colorStart, colorEnd, colorLevel);
    stroke(changeColor);

    // draw the diamond shape that moves with amplitude.
    noFill();
    beginShape();
    vertex(this.startX + this.spectrumWidth / 2 + random(-mappedLevel, mappedLevel), this.endY - 75
+ random(-mappedLevel, mappedLevel));
    vertex(this.startX + this.spectrumWidth + 50 + random(-mappedLevel, mappedLevel), this.startY -
200 + random(-mappedLevel, mappedLevel));
```

```
    vertex(this.startX + this.spectrumWidth / 2 + random(-mappedLevel, mappedLevel), height - this.endY + 75 + random(-mappedLevel, mappedLevel));
    vertex(this.startX - 50 + random(-mappedLevel, mappedLevel), this.startY - 200 + random(-mappedLevel, mappedLevel));
    endShape(CLOSE);
    pop();
  };
}



//Function to draw particles.
function ParticlePattern() {
  this.name = "particlepattern";
  this.burstSize = 50;
  this.minSize= 2;
  this.maxSize = 10;
  this.minSpeed = 1;
  this.maxSpeed = 3;

  let maxRadius;
  // how many layers of particles there are . I wanted to ceate depth. also maxradius defined, and make sure to call maxRadius
  //again after window resize.
  const layers = 3;
  function calculateMaxRadius() {
    maxRadius = Math.min(windowWidth, windowHeight);
  }
  calculateMaxRadius();
  windowResized(() => {
    calculateMaxRadius();
  });
```

```
// This code is written from online video and web sources and has been tweaked or changed to fit my idea.
function Particle(x, y, size, speed) {
  this.x = x;
  this.y = y;
  this.size = size;
  this.speed = speed;
  this.angle = Math.random() * 2 * Math.PI;
}
// this code and for loop is to create the diffeent size and speed of each layer and how many particles each layer will have.
// it was written in combination with video tutorial and information from the course and music template.
let particles = [];
let burstParticles = [];

for (let layer = 0; layer < layers; layer++) {
  particles[layer] = [];
  const numParticles = 40 * (layer + 1);

  for (let i = 0; i < numParticles; i++) {
    const angle = Math.random() * 2 * Math.PI;
    const x = width /2+ Math.cos(angle) * Math.random() * maxRadius;
    const y = height /2 + Math.sin(angle) * Math.random() * maxRadius;
    const size = Math.random() * 3 + 1 * (layer + 1);
    const speed = Math.random() * 2 + 1;

    particles[layer].push(new Particle(x, y, size, speed));
  }
}
// for the draw fuction I recieved help from peers.Most of this section uses seggestions from classmates
```

```
//and I changed a few values and re-wrote the code for learning purposes.
// This section uses constants and arrays from the above section to draw the particles.
this.draw = function () {
  push();
  image(backgroundImages[3], 0, 0, width, height);
  colorMode(HSB, 360, 250, 300);
  noStroke();
  //spectrum value
  let spectrum = fourier.analyze();

  //loop through layers
  for (let layer = 0; layer < layers; layer++) {

    fill(60, 200, 300);// set particle color
    //loop through particles
    for (let i = 0; i < particles[layer].length; i++) {
    // get refrence from current particle
      const p = particles[layer][i];
      // draw ellipse
      ellipse(p.x, p.y, p.size);

      //calculate spectrum and get values
      const spectrumIndex = Math.floor(map(i, 0, particles[layer].length, 0, spectrum.length - 1));
      const fftValue = spectrum[spectrumIndex];
      const angleChange = map(fftValue, 0, 255, -Math.PI, Math.PI);
      p.angle += angleChange;
      // particle position
      p.x += Math.cos(p.angle) * p.speed;
      p.y += Math.sin(p.angle) * p.speed;

    }
```

```
    }

    //create bass burst
    var bassFrequency = fourier.getEnergy("bass");
    if (Math.floor(bassFrequency) == bassFrequency && bassFrequency > 220) {
      for (let i = 0; i < this.burstSize; i++) {
        const x = width * 0.8;
        const y = height * 0.3;
        const size = random(this.minSize, this.maxSize);
        const speed = random(this.minSpeed, this.maxSpeed);
        const angle = random(TWO_PI);
        burstParticles.push(new Particle(x, y, size, speed, angle));
      }
    }

    //draw burst particles
    fill(180, 250, 300);
    for (let i = 0; i < burstParticles.length; i++) {
      const p = burstParticles[i];
      ellipse(p.x, p.y, p.size);
      p.x += Math.cos(p.angle) * p.speed;
      p.y += Math.sin(p.angle) * p.speed;
      if (dist(p.x, p.y, width / 2, height / 2) > maxRadius) {
        burstParticles.splice(i, 1); // To burst particles when they move too far
      }
    }
    // added forground image
    image(backgroundImages[6], 0, 0, width, height);
    pop();
  };
}
```

```
// needles functions has been changed.
function Needles() {
  // name of the visualisation
  this.name = "needles";

  // how large is the arc of the needle plot.
  var minAngle = PI + PI / 10;
  var maxAngle = TWO_PI - PI / 10;

  this.plotsAcross = 4; // Change to have all dials in one row
  this.plotsDown = 1; // Change to have all dials in one row

  // frequencies used by the energy function to retrieve a value
  // for each plot.
  this.frequencyBins = ["bass", "lowMid", "highMid", "treble"];

  // Define properties for bins, needle, ticks, outer ellipse, and balls
  var binX = [];
  var binY = [];
  var binSpeedX = [];
  var binSpeedY = [];
  var needleX, needleY, needleSpeedX, needleSpeedY;
  var tickX = [];
  var tickY = [];
  var tickSpeedX = [];
  var tickSpeedY = [];
  var outerEllipseX, outerEllipseY, outerEllipseSpeedX, outerEllipseSpeedY;


  // resize the plots sizes when the screen is resized.
```

```
this.onResize = function() {
  this.pad = width / 20;
  this.plotWidth = (width - this.pad) / this.plotsAcross;
  this.plotHeight = height / 3; // Adjust plot height
  this.dialRadius = (this.plotWidth - this.pad) / 2 - 10; // Adjust dial radius
  this.ballSize = 60; // Adjust ball size
  this.balls = []; // Array to store multiple balls

  // the start speed and positions of the circle bins containing the needles
  for (var i = 0; i < this.plotsAcross; i++) {
    binX.push(random(width));
    binY.push(random(height));
    binSpeedX.push(random(-2, 2));
    binSpeedY.push(random(-2, 2));
  }

  // positions and speed for other elements
  needleX = random(width);
  needleY = random(height);
  needleSpeedX = random(-2, 2);
  needleSpeedY = random(-2, 2);

  for (var j = 0; j < this.plotsAcross; j++) {
    tickX.push(random(width));
    tickY.push(random(height));
    tickSpeedX.push(random(-2, 2));
    tickSpeedY.push(random(-2, 2));
  }

  outerEllipseX = random(width);
  outerEllipseY = random(height);
```

```
    outerEllipseSpeedX = random(-2, 2);
    outerEllipseSpeedY = random(-2, 2);

    // To initialize the positions and velocity of the balls balls
    for (var i = 0; i < 16; i++) {
      this.balls.push({
        x: random(width),
        y: random(height / 2, height),
        speedX: random(-4, 4),
        speedY: random(-4, 4),
        color: color(random(220, 270), random(150, 250), random(150, 250))
      });
    }
  };

  // call onResize to set initial values when the object is created
  this.onResize();

// draw the plots to the screen
this.draw = function() {
  // create an array amplitude values from the fft.
  var spectrum = fourier.analyze();
  // iterator for selecting frequency bin.
  var currentBin = 0;
  push();
  image(backgroundImages[5], 0, 0, width, height);
  colorMode(HSB, 360, 250, 300);
  fill(290, 100, 150, 0.4);
  noStroke();
  // loop to place plots in a single row
  for (var j = 0; j < this.plotsAcross; j++) {
```

```
    // size of the plots
    var x = (j + 0.5) * width / (this.plotsAcross + 1); // Adjust x position based on column and canvas width
    var y = height / 3; // Adjust y position to be in the middle of the canvas
    var w = this.plotWidth - this.pad; // Width of the plot
    var h = this.plotHeight - this.pad * 2; // Height of the plot

    // draw ellipse at that location and size
    ellipse(binX[j], binY[j], w - 10, this.dialRadius * 2);
    // add on the ticks
    var centreX = binX[j];
    var bottomY = binY[j];

    this.ticks(centreX, bottomY, this.frequencyBins[currentBin], w);

    var energy = fourier.getEnergy(this.frequencyBins[currentBin]);

    // add the needle
    this.needle(energy, centreX, bottomY, h);
    currentBin++;
  }

  // This for loop will update positions and handle the collisions for bins and the canvas width and height.
  for (var i = 0; i < this.plotsAcross; i++) {
    binX[i] += binSpeedX[i];
    binY[i] += binSpeedY[i];
    // Handle collisions with canvas boundaries
    if (binX[i] > width || binX[i] < 0) {
      binSpeedX[i] *= -1;
    }
```

```
    if (binY[i] > height || binY[i] < 0) {
      binSpeedY[i] *= -1;
    }
  }

 // draw and update all balls
 for (var i = 0; i < this.balls.length; i++) {
    var ball = this.balls[i];
    fill(ball.color);



    var lowMidEnergy = fourier.getEnergy("lowMid");
    ball.size = map(lowMidEnergy, 0, 255, 20, 60);

    ellipse(ball.x, ball.y, ball.size, ball.size);
    // update the position of the ball
    ball.x += ball.speedX;
    ball.y += ball.speedY;

    // check if the ball hits the walls, handles collisions with the canvas, bounce back.
    if (ball.x > width - this.ballSize / 2 || ball.x < this.ballSize / 2) {
      ball.speedX *= -1;
    }
    if (ball.y < this.ballSize / 2 || ball.y > height) {
      ball.speedY *= -1;
    }
  }

// check if the bass frequency exceeds 220 and does not have a decimal number.
var bassEnergy = fourier.getEnergy("bass");
if (Math.floor(bassEnergy)== bassEnergy && bassEnergy > 220 ) {
```

```
    for (var i = 0; i < this.balls.length; i++) {
      this.balls[i].color = color(random(180, 330), random(150, 250), random(150, 300))
    }
  }
  // chick if treble frequency is above 100, if so then increas speed and movement of balls with random in both x and y .
  var trebleEnergy = fourier.getEnergy("treble");
  if (trebleEnergy > 100) {
    for (var i = 0; i < this.balls.length; i++) {
      this.balls[i].speedX += random(-1, 1);
      this.balls[i].speedY += random(-1, 1);
    }
  }
  // if  treble frequency is greater then 50 then increase the speed.
  if (trebleEnergy < 50 && trebleEnergy > 20){
    for (var i = 0; i < this.balls.length; i++) {
      this.balls[i].speedX *= 0.95; // Reduce speed in x direction
        this.balls[i].speedY *= 0.95; // Reduce speed in y direction
  }
  }
  pop();
};

  // draws a needle to an individual plot
  // @param energy: The energy for the current frequency
  // @param centreX: central x coordinate of the plot rectangle
  // @param bottomY: The bottom y coordinate of the plot rectangle
  this.needle = function(energy, centreX, bottomY, h) {
    push();
    stroke(240, 250, 350);
    translate(centreX, bottomY);
```

```
  theta = map(energy, 0, 255, minAngle, maxAngle);
  var x = this.dialRadius * cos(theta);
  var y = this.dialRadius * sin(theta);
  line(0, 0, x, y);
  pop();
};

// draw the graph ticks on an individual plot
// @param centreX: central x coordinate of the plot rectangle
// @param bottomY: The bottom y coordinate of the plot rectangle
// @param freqLabel: Label denoting the frequency of the plot
this.ticks = function(centreX, bottomY, freqLabel, w) {
  var nextTickAngle = minAngle;
  push();
  stroke(180, 250, 300);
  fill(90, 250, 300);
  translate(centreX, bottomY);

  // draw the semi circle for the bottom of the needle
  arc(0, 0, 20, 20, PI, 2 * PI);
  textAlign(CENTER);
  textSize(12);
  text(freqLabel, 0, -(this.plotHeight / -5));

  textAlign(CENTER);
  textSize(10);
  text(freqLabel, 0, -(w / 8)); // Adjust the distance here

  for (var i = 0; i < 9; i++) {
    var x = this.dialRadius * cos(nextTickAngle);
    var x1 = (this.dialRadius - 6) * cos(nextTickAngle);
```

```
        var y = this.dialRadius * sin(nextTickAngle);
        var y1 = (this.dialRadius - 6) * sin(nextTickAngle);

        line(x, y, x1, y1);
        nextTickAngle += PI / 10;
      }
      pop();
    };
}



//function to draw cubes
function Cubes() {
   // Visualization name
   this.name = "cubes";

var rotationAngle = 0;
var rotationSpeed = 0.01;

//This section of code was all written by me based of lecture videos and other vizualizations from this app.
// Draw function, inizializes x and y locations and draws shapes to creat a 3d cube effect.
   this.draw = function () {
      push();
      var x = 0;
      var y = 0;
      var centerX = width/3;
      var centerY = height/2;

      //load bachground image
```

```
image(backgroundImages[4], 0, 0, width, height);
// learning HSB colors
colorMode(HSB, 360, 250, 300);
//map amplitude
let level = amplitude.getLevel();
let getLevel = map(level, 0, 1, 0, 70);

// rotate speed based on the amplitude
rotationAngle += rotationSpeed * getLevel;

//set rotations
translate(centerX, centerY);
rotate(rotationAngle);
noStroke();
fill(300, 100, 255);

//cube1
//front face pink
beginShape();
vertex((x + 125) + (getLevel * 6 ), (y -20) - (getLevel * 6 ));
vertex((x + 125) + (getLevel * 6 ), (y -120) - (getLevel * 6 ));
vertex((x + 25) + (getLevel * 6), (y -120) - (getLevel * 6 ));
vertex((x + 25) + (getLevel * 6), (y -20)  - (getLevel* 6 ));
endShape(CLOSE);


// Bottom face purple
fill(270, 100, 255);
beginShape();
vertex((x + 125) + (getLevel * 6), (y -20) - (getLevel * 6));
vertex((x + 100), y);
```

```
vertex(x, y);
vertex((x + 25) + (getLevel * 6), (y -20)  - (getLevel * 6));
endShape(CLOSE);

//side face green
fill(150, 100, 255);
beginShape();
vertex((x + 25) + (getLevel * 6), (y -20)  - (getLevel * 6));
vertex(x, y);
vertex(x , y -60);
vertex((x + 25) + (getLevel * 6), (y -120) - (getLevel * 6));
endShape(CLOSE);



fill(300, 100, 255);
// Cube 2
// Front face pink
beginShape();
vertex((x - 125) - (getLevel * 6 ), (y -20) - (getLevel * 6 ));
vertex((x - 125) - (getLevel * 6 ), (y -120) - (getLevel * 6 ));
vertex((x - 25) - (getLevel * 6), (y -120) - (getLevel * 6 ));
vertex((x - 25) - (getLevel * 6), (y -20)  - (getLevel* 6 ));
endShape(CLOSE);

// Bottom face purple
fill(270, 100, 255);
beginShape();
vertex((x - 125) - (getLevel * 6), (y -20) - (getLevel * 6));
vertex((x - 100), y);
vertex(x, y);
vertex((x - 25) - (getLevel * 6), (y -20)  - (getLevel * 6));
```

```
endShape(CLOSE);

// Side face green
fill(150, 100, 255);
beginShape();
vertex((x - 25) - (getLevel * 6), (y -20)  - (getLevel * 6));
vertex(x, y);
vertex(x , y -60 );
vertex((x - 25) - (getLevel * 6), (y -120) - (getLevel * 6));
endShape(CLOSE);

// Cube 3
// Front face pink
fill(300, 100, 255);
beginShape();
vertex(-(x + 125) - (getLevel * 6 ), (y +20) + (getLevel * 6 ));
vertex(-(x + 125) - (getLevel * 6 ), (y +120) + (getLevel * 6 ));
vertex(-(x + 25) - (getLevel * 6), (y +120) + (getLevel * 6 ));
vertex(-(x + 25) - (getLevel * 6), (y +20)  + (getLevel * 6 ));
endShape(CLOSE);

// Bottom face purple
fill(270, 100, 255);
beginShape();
vertex(-(x + 125) - (getLevel * 6), (y +20) + (getLevel * 6));
vertex(-(x + 100), y);
vertex(-x, y);
vertex(-(x + 25) - (getLevel * 6), (y +20)  + (getLevel * 6));
endShape(CLOSE);

// Side face green
```

```
fill(150, 100, 255);
beginShape();
vertex(-(x + 25) - (getLevel * 6), (y +20)  + (getLevel * 6));
vertex(-x, y );
vertex(-x , y +55);
vertex(-(x + 25) - (getLevel * 6), (y +120) + (getLevel * 6));
endShape(CLOSE);

// Cube 4
// Front face pink
fill(300, 100, 255);
beginShape();
vertex(-(x - 125) + (getLevel * 6 ), (y +20) + (getLevel * 6 ));
vertex(-(x - 125) + (getLevel * 6 ), (y +120) + (getLevel * 6 ));
vertex(-(x - 25) + (getLevel * 6), (y +120) + (getLevel * 6 ));
vertex(-(x - 25) + (getLevel * 6), (y +20)  + (getLevel* 6 ));
endShape(CLOSE);

// Bottom face purple
fill(270, 100, 255);
beginShape();
vertex(-(x - 125) + (getLevel * 6), (y +20) + (getLevel * 6));
vertex(-(x - 100), y);
vertex(-x, y);
vertex(-(x - 25) + (getLevel * 6), (y +20)  + (getLevel * 6));
endShape(CLOSE);

// Side face green
fill(150, 100, 255);
beginShape();
vertex(-(x - 25) + (getLevel * 6), (y +20)  + (getLevel * 6));
```

```
      vertex(-x, y);
      vertex(-x , y +55);
      vertex(-(x - 25) + (getLevel * 6), (y +120) + (getLevel * 6));
      endShape(CLOSE);
      pop();
   };
}

// a set of functions that interact with dom elements and functionality.
 // I followed a tutotial on youtube for the nav bar links , all code for nav bar functionality is taken from there//
const menu = document.querySelector('#mobile-menu');
const menuLinks = document.querySelector('.navbar__menu');

menu.addEventListener('click', function(){
   menu.classList.toggle('is-active');
   menuLinks.classList.toggle('active');
})

// followed online and video for dark mode//
function toggleDarkmode () {
   let element = document.body;
   element.classList.toggle('dark-mode');
};

// After research I found this code in overtack , it stops the screen from scrolling on the web page when pressing the spacebar.
//I wanted the space bar to pause and play the music without moving the webpage.
window.addEventListener('keydown', (e) => {
   if (e.keyCode === 32 && e.target === document.body) {
```

```
    e.preventDefault();
  }
});
```