KALYAN GOPINATH

Student Number: 230656208

Candidate Number: MY2647

INTRODUCTION TO PROGRAMMING II

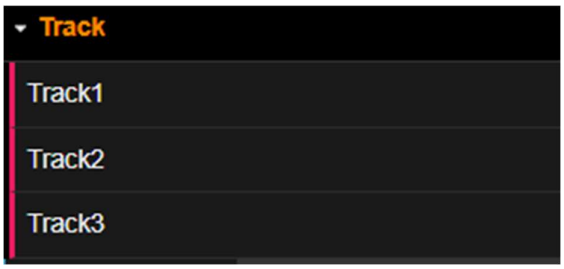
FINAL REPORT

---

1. Modifications and Extensions

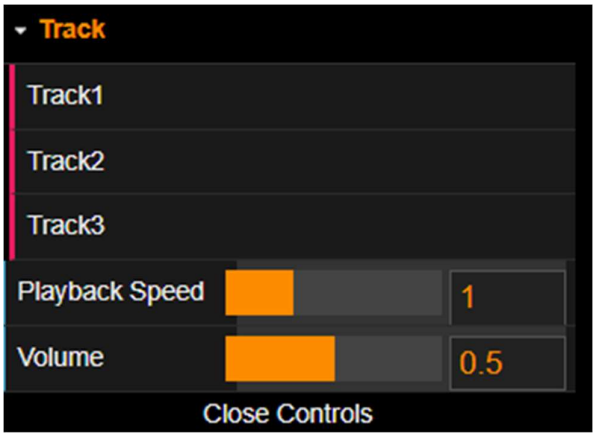
a) Function of the Extensions

- **Track Selector and Playback Control:**
  This feature allows users to choose between multiple audio tracks and dynamically adjust the playback speed. The functionality enhances user interaction by providing control over the music, which directly influences the visualizations displayed on the screen. The dynamic selection of tracks and playback speed offers a customizable listening and viewing experience tailored to user preferences.



- **Volume Control:**
  The volume control feature provides users with the ability to adjust the audio output according to their preferences. This functionality is essential for creating an adaptable environment, enabling users to enjoy the application in various settings—whether in quiet or noisy surroundings. It enhances user experience by offering more precise control over the sound level.

- **Custom GUI Style:**
  A custom-styled GUI was implemented to improve the user experience by enhancing both aesthetics and usability. The custom style includes a visually appealing layout with a consistent color scheme, making the interface more intuitive and accessible. The GUI elements, such as buttons, sliders, and track selectors, are styled to complement the visualizations, resulting in a cohesive and engaging user interface.



b) Code Integration with Template Design

- The new extensions are integrated into the existing template using a modular design approach. Each feature is implemented as a separate function or object, maintaining the readability and overall structure of the original code. This modular approach allows for easy maintenance and future scalability.
- The setupGUI function is called within the setup function to initialize the GUI components at the start of the program, ensuring all controls are available for user interaction as soon as the application runs.
- Event-driven functions like changeTrack, updateSpeed, and updateVolume handle user inputs dynamically, making the application responsive to user actions. The GUI styles are customized using both CSS and JavaScript, ensuring alignment with the visualizer's overall design.

c) Structure of the Extension Code

- The code for GUI controls, including the track selector, volume, and playback functions, is contained in the playGUI.js file. This separation of concerns helps maintain the core functionality in separate files, adhering to best practices for maintainability and scalability.
- Functions are logically grouped to promote clarity and ease of understanding: setupGUI handles GUI initialization, styleGUI deals with the aesthetic aspects, and specific handlers like changeTrack, updateSpeed, and updateVolume manage user interactions.
- The styleGUI function ensures that all GUI components have consistent styling and proper alignment, enhancing the overall look and feel of the application.

```
function setupGUI() {
    // Initialize the GUI
    gui = new dat.GUI();

    // Manually create the track selector instead of using dat.GUI
    createTrackSelector();

    // Add speed control
    gui.add(guiControls, 'speed', 0.1, 3.0).name('Playback Speed').onChange(updateSpeed);

    // Add volume control
    gui.add(guiControls, 'volume', 0.0, 1.0).name('Volume').onChange(updateVolume);

    // Apply custom styles to the GUI
    styleGUI(gui);
}
```

2. Effectiveness of the Plan (250 words)

A) Adherence to Schedule

The project plan was effectively structured with clear milestones for completing core functionalities and GUI enhancements. The initial plan was to implement fundamental features, such as track selection and basic playback controls, within the first two weeks. Advanced features like volume control and custom GUI styling were scheduled for the subsequent weeks. This plan was largely adhered to, with only minor deviations. Occasional adjustments were necessary to accommodate unforeseen challenges, but overall, the schedule was well maintained.

B) Division of Tasks and Time Management

The tasks were divided strategically to prioritize core functionalities, ensuring that the application's backbone was robust before focusing on additional features. Key tasks, such as implementing playback controls and volume adjustment, were completed first to establish a stable foundation. This method allowed subsequent tasks related to GUI refinement and user experience enhancements to be executed more smoothly. Time was allocated efficiently, with dedicated periods for testing and debugging. This approach proved invaluable, as it allowed for early detection and correction of bugs, minimizing the risk of major issues close to the deadline.

C) Unexpected Difficulties or Challenges

Unexpected difficulties arose in integrating custom GUI elements with the existing p5.js sketch, particularly in ensuring consistent behavior across different browsers. These challenges required additional time for debugging and refinement. However, continuous testing and an adaptable project plan allowed these issues to be effectively managed, ensuring that the final product met all functional and quality requirements while being delivered on time.

3. Evaluation of the Process and Final Product (250 words)

A) Self-Evaluation of the Project Completion Process

The project followed a structured and methodical approach, with clearly defined milestones that enabled steady progress and timely completion of all core features. The initial focus on

developing the foundational functionalities, such as track selection and playback control, laid a solid groundwork, allowing for subsequent GUI enhancements and refinements. In hindsight, the project could have benefited from more time allocated to cross-browser testing earlier in the development process, which would have minimized last-minute adjustments. Additionally, incorporating more iterative user feedback loops from the beginning could have led to a more polished final product.

B) System and User Testing

System testing was conducted rigorously to verify that all functionalities—such as track selection, playback speed control, and volume adjustment—performed as intended. The system was tested with various input scenarios, including different tracks and volume levels, to ensure robustness. User testing involved a small group of users interacting with the visualizer, who provided feedback on both functionality and usability.

C) Errors and User Feedback

During testing, minor issues such as occasional delays in loading tracks and minor misalignment of GUI elements were uncovered. Users responded positively to the application's functionality and design but pointed out areas for improvement, such as more intuitive track selection and smoother volume adjustments.

D) Future Improvements

To rectify these issues in future versions, I would prioritize optimizing the track loading process to reduce delays and refining the GUI alignment for a more cohesive user experience. Additionally, enhancing the feedback mechanisms for user interactions, such as visual cues when a track is selected or volume is adjusted, would further improve usability.

4. External Sources Utilized

A) Self-Evaluation of the Project Completion Process

The project followed a structured and methodical approach, with clearly defined milestones that enabled steady progress and timely completion of all core features. The initial focus on developing the foundational functionalities, such as track selection and playback control, laid a solid groundwork, allowing for subsequent GUI enhancements and refinements. In hindsight, the project could have benefited from more time allocated to cross-browser testing earlier in the development process, which would have minimized last-minute adjustments. Additionally, incorporating more iterative user feedback loops from the beginning could have led to a more polished final product.

B) System and User Testing

System testing was conducted rigorously to verify that all functionalities—such as track selection, playback speed control, and volume adjustment—performed as intended. The system was tested with various input scenarios, including different tracks and volume levels, to ensure robustness. User testing involved a small group of users interacting with the visualizer, who provided feedback on both functionality and usability.

C) Errors and User Feedback

During testing, minor issues such as occasional delays in loading tracks and minor misalignment of GUI elements were uncovered. Users responded positively to the application's functionality and design but pointed out areas for improvement, such as more intuitive track selection and smoother volume adjustments.

D) Future Improvements

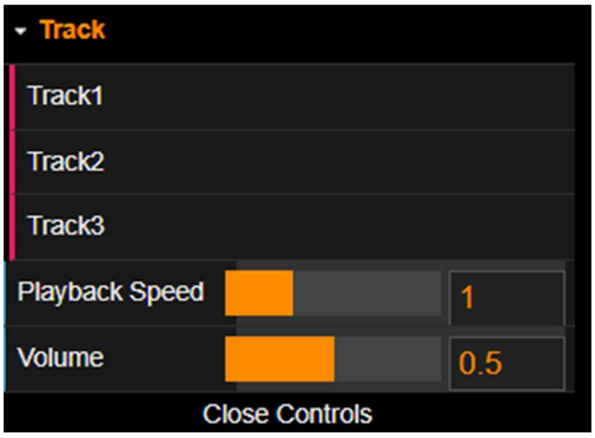
To rectify these issues in future versions, I would prioritize optimizing the track loading process to reduce delays and refining the GUI alignment for a more cohesive user experience. Additionally, enhancing the feedback mechanisms for user interactions, such as visual cues when a track is selected or volume is adjusted, would further improve usability.

5. Weekly Logs (Weeks 15-20)

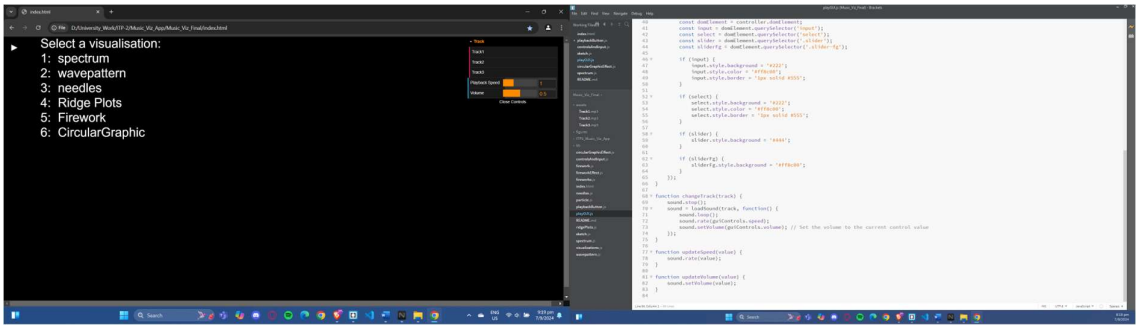
- **Week 15-16:**
    - **Targets:** Set up basic audio loading and playback functionality.
    - **Progress:** Implemented track selection and basic playback controls.
    - **Reflections:** Successfully achieved all targets; resolved minor browser compatibility issues.

```
var trackList = [
    { name: 'Track1', file: 'assets/Track1.mp3' },
    { name: 'Track2', file: 'assets/Track2.mp3' },
    { name: 'Track3', file: 'assets/Track3.mp3' }
];
```

- **Week 17-18:**
    - **Targets:** Integrate volume control and refine the GUI for better usability.
    - **Progress:** Implemented volume control and added styles to GUI elements. Completed debugging and refinement.
    - **Reflections:** GUI improvements enhanced user experience, and time was well-managed.



- **Week 19-20:**
    - **Targets:** Conduct system and user testing; finalize the report.
    - **Progress:** Completed testing, gathered user feedback, and addressed minor issues identified during testing.
    - **Reflections:** The testing process was comprehensive, and usability improvements were promptly made.



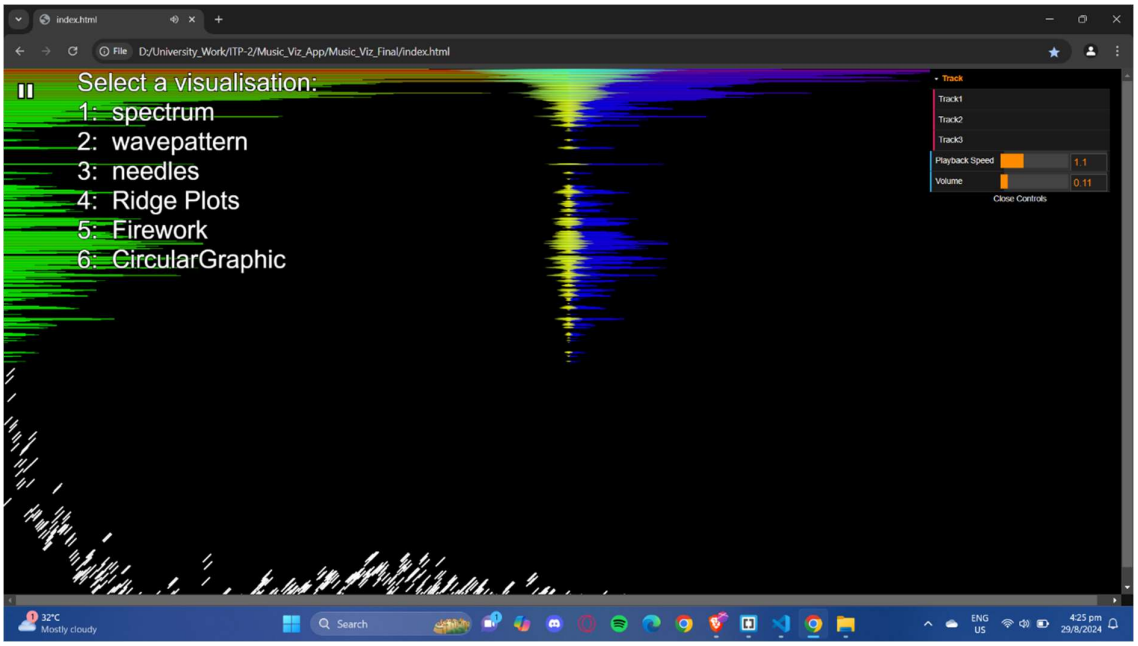
6. Testing Methodologies

- **Unit Testing:** Used JavaScript libraries like p5.js to test individual functions, such as playback controls, ensuring correct functionality.
- **Integration Testing:** Verified that different modules (e.g., track selector, playback control) worked together seamlessly.
- **System Testing:** Ensured the complete application functioned as expected, including all user-facing features.
- **User Testing:** Collected feedback from users to make further improvements based on their experiences.

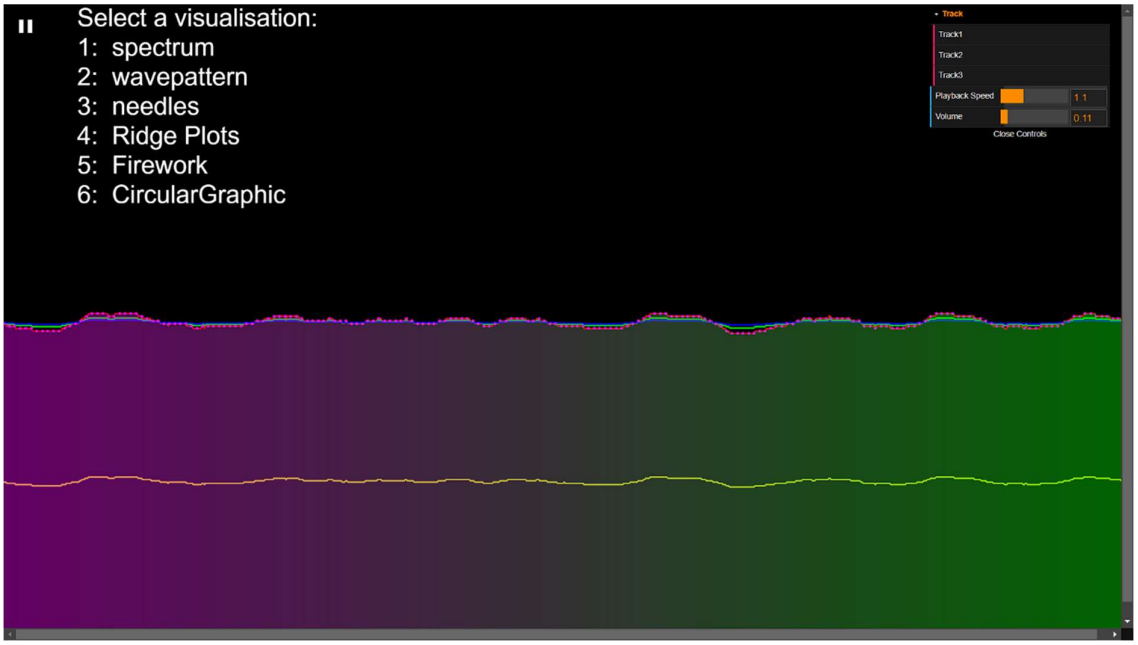
Conclusion

The snapshots provided demonstrate a visually appealing and fully functional music visualizer with interactive controls, confirming the success of the project in meeting its objectives.

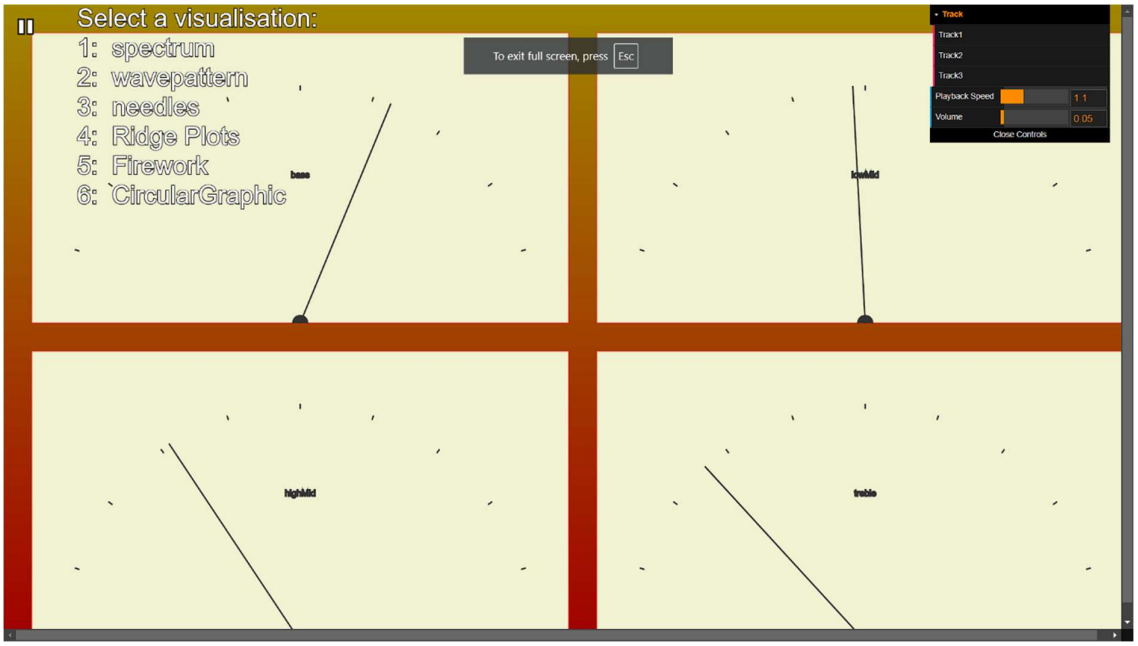
1.Spectrum



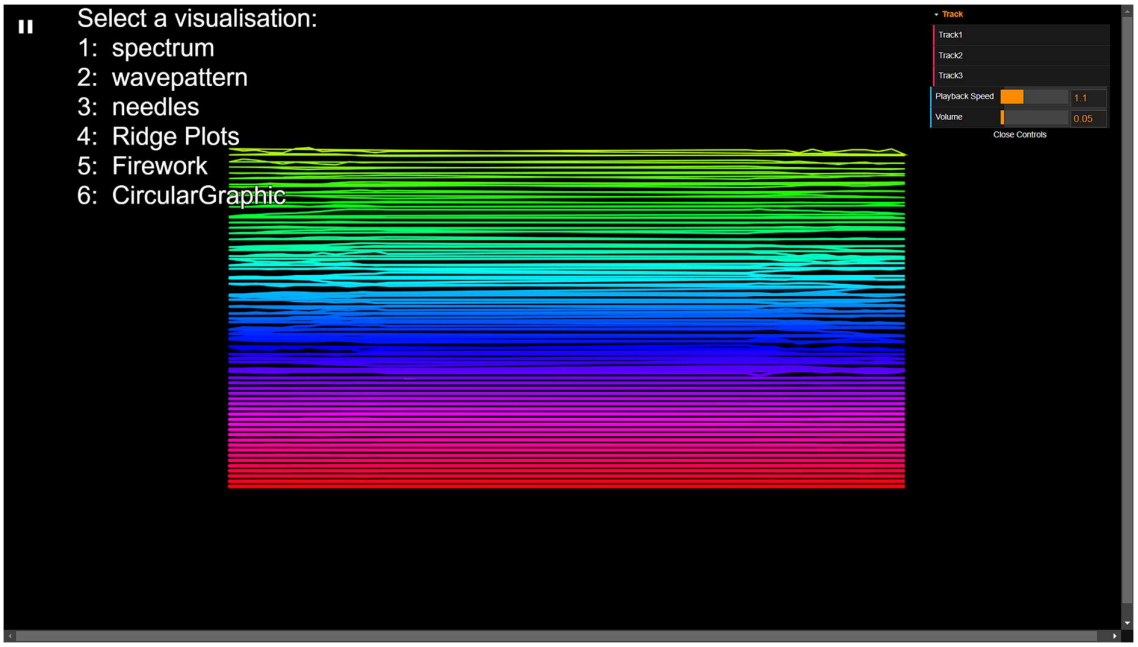
2. Wavepattern



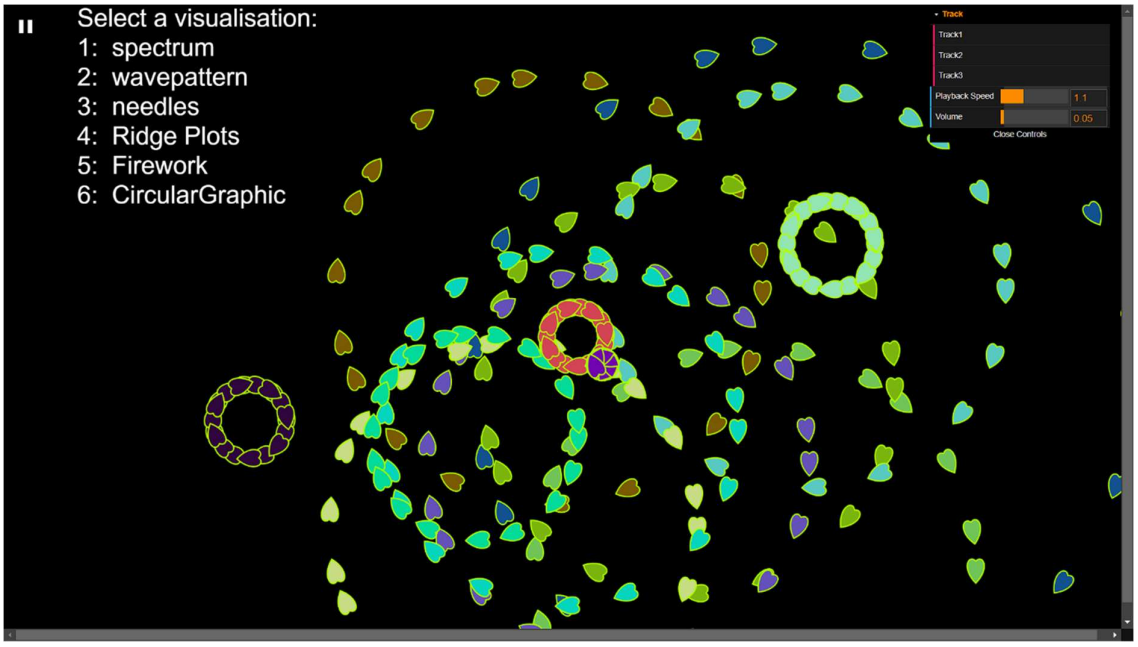
3. Needles



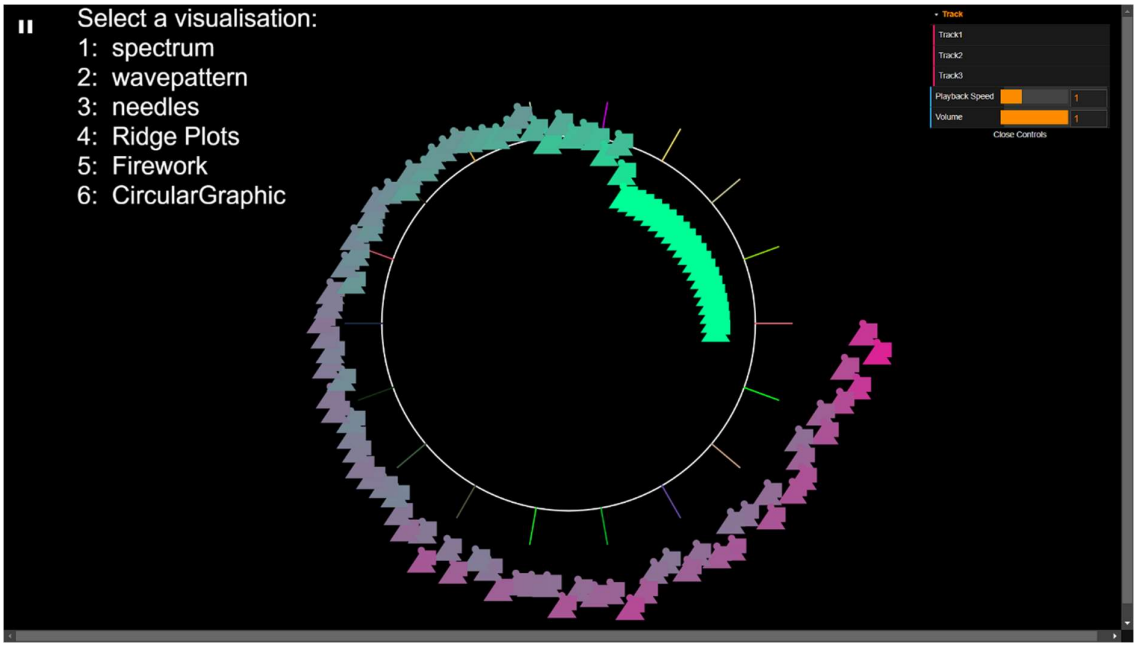
4. Ridge Plots



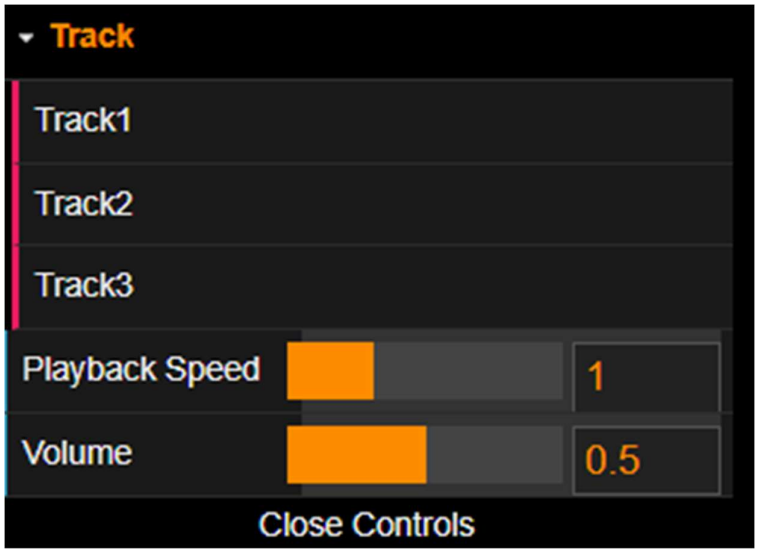
5. Fireworks



6. CircularGraphic



7. PlayGUI



The whole code for the Music Visualization Application:

1. **Index.html**

```
<!DOCTYPE html>
<html>
  <head>
    <script src="lib/p5.min.js"></script>
    <script src="lib/p5.sound.js"></script>
      <script
src="https://cdn.jsdelivr.net/npm/dat.gui/build/dat.gui.min.js
"></script> <!-- Add this for GUI -->
    <script src="sketch.js"></script>
    <script src="playbackButton.js"></script>
    <script src="controlsAndInput.js"></script>
      <script src="playGUI.js"></script> <!-- Include the GUI
script -->
    <script src="visualisations.js"></script>

    <script src="needles.js"></script>
    <script src="wavepattern.js"></script>
```

```
    <script src="spectrum.js"></script>
    <script src="ridgePlots.js"></script>
    <script src="fireworkEffect.js"></script>
    <script src="lib/beatDetect.js"></script>
    <script src="particle.js"></script>
    <script src="firework.js"></script>
    <script src="fireworks.js"></script>
    <script src="circularGraphicEffect.js"></script>


    <!--<link rel="stylesheet" type="text/css"
href="style.css">-->
    <style>
        body { padding: 0; margin: 0; }
        .dg .cr {
            background: #1e1e1e;
            color: #f0f0f0;
        }
        .dg .cr .c {
            background: #333;
            color: #fff;
        }
        .dg .c input[type="text"] {
            background: #333;
            color: #fff;
            border: 1px solid #555;
        }
        .dg .c .slider {
            background: #444;
        }
        .dg .c .slider-fg {
            background: #ff8c00;
        }
        .dg .title {
            background: #444;
            color: #ff8c00;
            font-weight: bold;
        }
        .dg li:hover {
            background: #555;
        }
    </style>
```

```
  </head>
  <body>
  </body>
</html>

```

2. beatDetect.js

```
//inspiration for beat detection comes from
//http://archive.gamedev.net/archive/reference/programming/features/beatdetection/
function BeatDetect(){

    var sampleBuffer = [];

    this.detectBeat = function(spectrum){
        var sum = 0;
        var isBeat = false;
        for(var i=0;i<spectrum.length;i++){
            sum+=spectrum[i] * spectrum[i];
        }

        if(sampleBuffer.length == 60){
            var sampleSum = 0;

            //detect a beat
            for(var i=0;i<sampleBuffer.length;i++){
                sampleSum += sampleBuffer[i];
            }
            var sampleAverage = sampleSum / sampleBuffer.length;
            //var c = 1.1;
            //work out a better c
            //start by calculating the variance
            //
            var c = calculateConstant(sampleAverage);
            if(sum > sampleAverage*c){
                //beat
                isBeat = true;
            }
            sampleBuffer.splice(0,1);
```

```
            sampleBuffer.push(sum);
        }else{
            sampleBuffer.push(sum);
        }

        return isBeat;
    }

    function calculateConstant(sampleAverage){
        var varianceSum = 0;
        for(var i=0;i<sampleBuffer.length;i++){
            varianceSum += sampleBuffer[i] - sampleAverage;
        }
        var variance = varianceSum/sampleBuffer.length;

        var m = -0.15 / (25-200);
        var b = 1 + (m*200);
        var c = (m * variance) + b;
        return c;
    }
}
```

3. circularGraphicEffect.js

```
function CircularGraphicEffect() {
    this.name = "CircularGraphic";

    this.setup = function() {
        // Any setup specific to this effect
    }

    this.draw = function() {
        background(0);
        var spectrum = fourier.analyze();
        var bass = fourier.getEnergy("bass");
        var treble = fourier.getEnergy("treble");

        // Draw central circle
        noFill();
        stroke(255);
```

```
        strokeWeight(2);
        ellipse(width / 2, height / 2, bass * 2, bass * 2);

        // Draw rotating lines
        for (let i = 0; i < 360; i += 20) {
            let x = width / 2 + cos(radians(i)) * bass;
            let y = height / 2 + sin(radians(i)) * bass;
            let x2 = width / 2 + cos(radians(i)) * (bass +
50);
            let y2 = height / 2 + sin(radians(i)) * (bass +
50);

            stroke(random(255), random(255), random(255));
            line(x, y, x2, y2);
        }

        // Draw surrounding circles
        for (let i = 0; i < spectrum.length; i += 10) {
            let angle = map(i, 0, spectrum.length, 0, TWO_PI);
            let x = width / 2 + cos(angle) * (200 +
spectrum[i]);
            let y = height / 2 + sin(angle) * (200 +
spectrum[i]);

            fill(spectrum[i], 255 - spectrum[i], 150);
            noStroke();
            ellipse(x, y, 10, 10);
            triangle(x, y, x + 20, y + 30, x - 20, y + 30);
            rect(x, y, 20, 20);
        }
    }
}

function Pulse() {
    this.name = "Pulse";
    var pulseSize = 0;

    this.draw = function() {
        var bassEnergy = fourier.getEnergy("bass");

        var size = map(pulseSize, 0, 255, width / 4, width);
        fill(255, 50);
```

```
        noStroke();
        ellipse(width / 2, height / 2, size, size);
        pulseSize *= 0.95; // Slowly decrease the size for a
pulsing effect
    };

    this.triggerBeatEffect = function() {
        // Increase the size on beat
        pulseSize = 255;
    };

    this.unSelectVisual = function() {
        console.log("Pulse unselected");
    }

    this.selectVisual = function() {
        console.log("Pulse selected");
    }
}
```

4. controlsAndInput.js

```
// Constructor function to handle the onscreen menu, keyboard
and mouse controls
function ControlsAndInput(){

    this.menuDisplayed = false;

    // Playback button displayed in the top left of the screen
    this.playbackButton = new PlaybackButton();

    // Make the window fullscreen or revert to windowed
    this.mousePressed = function(){
        // Check if the playback button has been clicked
        var isButtonClicked = this.playbackButton.hitCheck();
        // If not to make the visualisation fullscreen
        if(!isButtonClicked){
            var fs = fullscreen();
            fullscreen(!fs);
        }
    };

    // Responds to keyboard presses
```

```
    // @param keycode the ASCII code of the key pressed
    this.keyPressed = function(keycode){
        console.log(keycode);
        if(keycode == 32){
            this.menuDisplayed = !this.menuDisplayed;
        }

        if(keycode > 48 && keycode < 58){
            var visNumber = keycode - 49;
            vis.selectVisual(vis.visuals[visNumber].name);
        }
    };

    // Draws the playback button and potentially the menu
    this.draw = function(){
        push();
        fill("white");
        stroke("black");
        strokeWeight(2);
        textSize(34);

        // Playback button
        this.playbackButton.draw();
        // Only draw the menu if menuDisplayed is set to true.
        if(this.menuDisplayed){
            text("Select a visualisation:", 100, 30);
            this.menu();
        }
        pop();
    };

    this.menu = function(){
        // Draw out menu items for each visualisation
        for(var i = 0; i < vis.visuals.length; i++){
            var yLoc = 70 + i * 40;
            text((i + 1) + ":  " + vis.visuals[i].name, 100,
yLoc);
        }
    };
}
```

5. firework.js

```
function Firework(colour,x,y){
    var colour = colour;
    var x =x;
    var y=y;
    var particles = []
    this.depleted = false;
    for (var i=0;i<360;i+=18){
        particles.push(new Particle(x,y,colour,i,10));
    }
    this.draw = function(){
        for (var i=0;i<particles.length;i++){
            particles[i].draw();
        }
        if(particles[0].speed<=0){
            this.depleted = true;

        }
    }
}
```

6. fireworkEffect.js

```
function FireworkEffect()
{
    this.name = "Firework";

    var fireworks;

    this.setup = function(){
        background(0);
        angleMode(RADIANS);
```

```
        frameRate(60);
        beatDetect = new BeatDetect();
        fireworks = new Fireworks();
    }
    this.setup();

    this.draw = function(){
        background(0);
        var spectrum = fourier.analyze();
        if(beatDetect.detectBeat(spectrum)){
            fill(255,0,0);
            fireworks.addFirework();
        }
        fireworks.update();
    }
}
```

7. fireworks.js

```


function Fireworks(){
    var fireworks = [];
    this.addFirework= function(){
        var f_colour= null;
        var r = random(0,255);
        var g = random(0,255);
        var b = random(0,255);
        console.log(r+""+g+""+b);
        f_colour = color(r,g,b);
        var f_x =random(width*0.2,width*0.8);
        var f_y=random(height*0.2,height*0.8);
        var firework = new Firework(f_colour,f_x,f_y);
        fireworks.push(firework);
    }
    this.update = function(){
        for(var i=0;i<fireworks.length;i++){
            fireworks[i].draw();
            if(fireworks[i].depleted){
                fireworks.splice(i,1);
            }
```

```
        }
    }
}
```

8.needles.js

```

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 = 2;
    this.plotsDown = 2;

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

    //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 - this.pad) / this.plotsDown;
        this.dialRadius = (this.plotWidth - this.pad) / 2 - 5;
    };
    //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();
        // Draw the background gradient
        this.drawGradientBackground();

        fill('#f0f2d2');
        //nested for loop to place plots in 2*2 grid.
        for (var i = 0; i < this.plotsDown; i++) {
            for (var j = 0; j < this.plotsAcross; j++) {
                var widthOffset = (j + 1) * this.pad / 2;
                var heightOffset = (i + 1) * this.pad / 2;

                //calculate the size of the plots
                var x = j * this.plotWidth + widthOffset;
                var y = i * this.plotHeight + heightOffset;
                var w = this.plotWidth;
                var h = this.plotHeight;

                //draw a rectangle at that location and size
                rect(x, y, w, h);

                //add on the ticks
                var centreX = (this.plotWidth / 2) +
  (this.plotWidth) * j + widthOffset;
                var bottomY = (this.plotHeight) * (i + 1) +
  heightOffset;

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

                //add the needle
                this.needle(energy, centreX, bottomY);

                //add the ticks
                this.ticks(centreX, bottomY,
  this.frequencyBins[currentBin]);

                currentBin++;
            }
```

```
        }
        pop();
    };

    this.unSelectVisual = function() {
        // Functionality for unselecting the visual
    };

    this.selectVisual = function() {
        // Functionality for selecting the visual
    };

    /*
     *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) {
        push();
        stroke('#333333');
        //translate so 0 is at the bottom of the needle
        translate(centreX, bottomY);
        //map the energy to the angle for the plot
        var theta = map(energy, 0, 255, minAngle, maxAngle);
        //calculate x and y coordinates from angle for the length of needle
        var x = this.dialRadius * cos(theta);
        var y = this.dialRadius * sin(theta);
        //draw the needle
        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) {
        // 8 ticks from pi to 2pi
        var nextTickAngle = minAngle;
        push();
        stroke('#333333');
        fill('#333333');
        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 / 2));

        for (var i = 0; i < 9; i++) {
            //for each tick work out the start and end coordinates of
            //based on its angle from the needle's origin.
            var x = this.dialRadius * cos(nextTickAngle);
            var x1 = (this.dialRadius - 5) * cos(nextTickAngle);

            var y = this.dialRadius * sin(nextTickAngle);
            var y1 = (this.dialRadius - 5) * sin(nextTickAngle);

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

    /*
     * Draws a gradient background
     */
    this.drawGradientBackground = function() {
        noFill();
        for (var i = 0; i <= height; i++) {
            var inter = map(i, 0, height, 0, 1);
```

```
            var c = lerpColor(color(255, 215, 0, 100),
color(255, 0, 0, 100), inter);
            stroke(c);
            line(0, i, width, i);
        }
    };
}
```

9. Particle.js

```
function Particle(x, y, colour, angle, speed) {
    var x = x;
    var y = y;
    var angle = angle;
    this.speed = speed;
    this.colour = colour;
    this.age = 255;

    this.draw = function() {
        this.update();
        var r = red(this.colour) - (255 - this.age);
        var g = green(this.colour) - (255 - this.age);
        var b = blue(this.colour) - (255 - this.age);

        var c = color(r, g, b);
        fill(c);
        this.age -= 1;

        push();
        translate(x, y);
        rotate(angle);

        // Draw the heart shape
        beginShape();
        vertex(0, -10);
        bezierVertex(10, -20, 20, 0, 0, 20);
        bezierVertex(-20, 0, -10, -20, 0, -10);
        endShape(CLOSE);

```

```
        pop();
    };

    this.update = function() {
        this.speed -= 0.1;
        x += cos(angle) * speed + noise(frameCount) * 10;
        y += sin(angle) * speed + noise(frameCount) * 10;
    };
}
```

10. playbackButton.js

```
p// PlaybackButton class to handle play/pause functionality
function PlaybackButton() {
    this.x = 20;
    this.y = 20;
    this.width = 20;
    this.height = 20;

    // Flag to determine whether to play or pause after button click and to determine which icon to draw
    this.playing = false;

    this.draw = function() {
        push();
        fill('white');
        stroke('black');
        strokeWeight(2);
        if (this.playing) {
            // Draw pause icon
            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 {
            // Draw play icon
            triangle(this.x, this.y, this.x + this.width, this.y + this.height / 2, this.x, this.y + this.height);
        }
        pop();
    };
```

```
    // Checks for clicks on the button, starts or pauses playback
    // @returns true if clicked, false otherwise
    this.hitCheck = function() {
        if (mouseX > this.x && mouseX < this.x + this.width && mouseY > this.y && mouseY < this.y + this.height) {
            this.togglePlayPause(); // Toggle play/pause on click
            return true;
        }
        return false;
    };

    // Toggles the play/pause state of the music
    this.togglePlayPause = function() {
        if (sound.isPlaying()) {
            sound.pause();
        } else {
            sound.play();
        }
        this.playing = !this.playing;
    };
}
```

11. playGUI.js

```
// Initialize GUI controls object
var guiControls = {
    track: 'assets/Track2.mp3', // Default track
    speed: 1.0,                 // Playback speed
    volume: 0.5                 // Volume control (default is 50%)
};

var gui;

function setupGUI() {
    // Initialize the GUI
    gui = new dat.GUI();

```

```
    // Manually create the track selector instead of using dat.GUI
    createTrackSelector();

    // Add speed control
    gui.add(guiControls, 'speed', 0.1, 3.0).name('Playback Speed').onChange(updateSpeed);

    // Add volume control
    gui.add(guiControls, 'volume', 0.0, 1.0).name('Volume').onChange(updateVolume);

    // Apply custom styles to the GUI
    styleGUI(gui);
}

function createTrackSelector() {
    var folder = gui.addFolder('Track');
    trackList.forEach((track, index) => {
        folder.add({ [track.name]: function() {
            changeTrack(track.file);
        }}, track.name);
    });
    folder.open();
}

function styleGUI(gui) {
    const controllers = gui.__controllers;
    controllers.forEach(controller => {
        const domElement = controller.domElement;
        const input = domElement.querySelector('input');
        const select = domElement.querySelector('select');
        const slider = domElement.querySelector('.slider');
        const sliderFg = domElement.querySelector('.slider-fg');

        if (input) {
            input.style.background = '#222';
            input.style.color = '#ff8c00';
            input.style.border = '1px solid #555';
        }
```

```
        if (select) {
            select.style.background = '#222';
            select.style.color = '#ff8c00';
            select.style.border = '1px solid #555';
        }

        if (slider) {
            slider.style.background = '#444';
        }

        if (sliderFg) {
            sliderFg.style.background = '#ff8c00';
        }
    });
}

function changeTrack(track) {
    sound.stop();
    sound = loadSound(track, function() {
        sound.loop();
        sound.rate(guiControls.speed);
        sound.setVolume(guiControls.volume); // Set the volume to the current control value
    });
}

function updateSpeed(value) {
    sound.rate(value);
}

function updateVolume(value) {
    sound.setVolume(value);
}
```

12. ridgePlots.js

```
function RidgePlots() {
    this.name = "Ridge Plots";

    var startX;
    var startY;
```

```
    var endY;
    var spectrumWidth;
    var speed = 0.7;
    var output = [];
    var maxWaves = 65;
    var beatDetect = new BeatDetect();

    this.onResize = function () {
        startX = width / 5;
        endY = height / 10; // Increase the endY to allow more
vertical space for waves
        startY = height - endY;
        spectrumWidth = (width / 5) * 3;
    };

    this.onResize();

    this.draw = function () {
        background(0, 50); // Add some transparency to create
a trailing effect

        var spectrum = fourier.analyze();
        var isBeat = beatDetect.detectBeat(spectrum);

        if (frameCount % 10 == 0) {
            addWave();
        }

        for (var i = output.length - 1; i >= 0; i--) {
            var wave = output[i];
            var hue = map(wave[0].y, endY, startY, 0, 360);
            colorMode(HSB, 360);
            stroke(hue, 360, 360);
            fill(hue, 360, 360, 50); // Add transparency to
the fill

            beginShape();
            for (var j = 0; j < wave.length; j++) {
                wave[j].y -= speed;
                var thickness = map(wave[j].y, endY, startY,
1, 5); // Dynamic thickness
                strokeWeight(thickness);
```

```
                vertex(wave[j].x, wave[j].y);

                // Draw dynamic lines based on the beat
                if (isBeat && j % 5 == 0 && wave[j].y > endY)
{
                    var amplitude = map(wave[j].y, endY,
startY, 1, 100); // Adjust amplitude scaling as needed
                    line(wave[j].x, wave[j].y, wave[j].x,
wave[j].y - amplitude);
                }
            }
            endShape(CLOSE);

            if (wave[0].y < endY) {
                output.splice(i, 1);
            }
        }

        // RGB color mode reset
        colorMode(RGB);
    }

    function addWave() {
        var w = fourier.waveform();
        var outputWave = [];
        var smallScale = 10; // Increase smallScale for taller
waves
        var bigScale = 20; // Increase bigScale for taller
waves

        for (var i = 0; i < w.length; i++) {
            if (i % 20 == 0) {
                var x = map(i, 0, 1024, startX, startX +
spectrumWidth);

                if (i < 1024 * 0.25 || i > 1024 * 0.75) {
                    var y = map(w[i], -1, 1, -smallScale,
smallScale);
                    var o = { x: x, y: startY + y };
                    outputWave.push(o);
                }
            }
```

```
        }

        if (output.length > maxWaves) {
            output.shift(); // Remove the oldest wave to
maintain the maximum number of waves
        }

        output.push(outputWave);
    }
}

function BeatDetect() {
    var sampleBuffer = [];

    this.detectBeat = function(spectrum) {
        var sum = 0;
        var isBeat = false;
        for (var i = 0; i < spectrum.length; i++) {
            sum += spectrum[i] * spectrum[i];
        }

        if (sampleBuffer.length == 60) {
            var sampleSum = 0;

            // Detect a beat
            for (var i = 0; i < sampleBuffer.length; i++) {
                sampleSum += sampleBuffer[i];
            }
            var sampleAverage = sampleSum /
sampleBuffer.length;
            var c = calculateConstant(sampleAverage);
            if (sum > sampleAverage * c) {
                isBeat = true;
            }
            sampleBuffer.splice(0, 1);
            sampleBuffer.push(sum);
        } else {
            sampleBuffer.push(sum);
        }

        return isBeat;
    }
```

```
•
•       function calculateConstant(sampleAverage) {
•           var varianceSum = 0;
•           for (var i = 0; i < sampleBuffer.length; i++) {
•               varianceSum += sampleBuffer[i] - sampleAverage;
•           }
•           var variance = varianceSum / sampleBuffer.length;
•
•           var m = -0.15 / (25 - 200);
•           var b = 1 + (m * 200);
•           var c = (m * variance) + b;
•           return c;
•       }
•   }
•
•
```

13.sketch.js

```
•   var controls = null;
•   var vis = null;
•   var sound = null;
•   var fourier;
•
•   var trackList = [
•       { name: 'Track1', file: 'assets/Track1.mp3' },
•       { name: 'Track2', file: 'assets/Track2.mp3' },
•       { name: 'Track3', file: 'assets/Track3.mp3' }
•   ];
•
•   function preload(){
•       sound = loadSound(guiControls.track);
•   }
•
•   function setup(){
•       createCanvas(windowWidth, windowHeight);
•       background(0);
•
•       setupGUI();
```

```
•
•       controls = new ControlsAndInput();
•
•       fourier = new p5.FFT();
•
•       vis = new Visualisations();
•       vis.add(new Spectrum());
•       vis.add(new WavePattern());
•       vis.add(new Needles());
•       vis.add(new RidgePlots());
•       vis.add(new FireworkEffect());
•       vis.add(new CircularGraphicEffect());
•
•       if(vis.visuals.length > 0){
•           vis.selectVisual(vis.visuals[0].name);
•       }
•
•       sound.pause(); // Pause initially until play is triggered
•       sound.setVolume(guiControls.volume); // Set initial volume
•   }
•
•   function draw(){
•       background(0);
•
•       if(vis.selectedVisual){
•           vis.selectedVisual.draw(guiControls);
•       }
•       controls.draw();
•   }
•
•   function mouseClicked(){
•       if (controls) {
•           controls.mousePressed();
•       }
•   }
•
•   function keyPressed(){
•       if (controls) {
•           controls.keyPressed(keyCode);
•       }
•   }
•
```

```
function windowResized(){
    resizeCanvas(windowWidth, windowHeight);
    if(vis.selectedVisual &&
vis.selectedVisual.hasOwnProperty('onResize')){
        vis.selectedVisual.onResize();
    }
}
```

14. spectrum.js

```
function Spectrum() {
    this.name = "spectrum";

    this.draw = function () {
        push();
        var spectrum = fourier.analyze();
        noStroke();

        // First spectrum with green to red gradient
        var c1 = color(0, 255, 0);
        var c2 = color(255, 0, 0);
        for (var i = 0; i < spectrum.length; i++) {
            var y = map(i, 0, spectrum.length, 0, height);
            var h = map(spectrum[i], 0, 255, 0, width);
            var c = lerpColor(c1, c2, spectrum[i] / 255);
            fill(c);
            rect(0, y, h, height / spectrum.length);
        }

        // Second spectrum with blue to purple gradient
        var c3 = color(0, 0, 255);
        var c4 = color(128, 0, 128);
        for (var i = 0; i < spectrum.length; i++) {
            var y = map(i, 0, spectrum.length, 0, height);
            var h = map(spectrum[i], 0, 255, 0, width);
            var c = lerpColor(c3, c4, spectrum[i] / 255);
            fill(c);
            rect(width / 2, y, h / 2, height /
spectrum.length); // Offset the second spectrum
        }

```

```
        // Third spectrum with yellow to cyan gradient
        var c5 = color(255, 255, 0);
        var c6 = color(0, 255, 255);
        for (var i = 0; i < spectrum.length; i++) {
            var y = map(i, 0, spectrum.length, 0, height);
            var h = map(spectrum[i], 0, 255, 0, width);
            var c = lerpColor(c5, c6, spectrum[i] / 255);
            fill(c);
            ellipse(width / 2, y, h / 4, height / spectrum.length); // Use circles
        }

        // Fourth element: Diagonal lines
        stroke(255);
        for (var i = 0; i < spectrum.length; i++) {
            var x = map(i, 0, spectrum.length, 0, width);
            var h = map(spectrum[i], 0, 255, 0, height / 2);
            line(x, height - h, x + 10, height - h - 10); // Diagonal lines
        }

        pop();
    };

    this.unSelectVisual = function () {
        console.log("Spectrum unselected");
    }

    this.selectVisual = function () {
        console.log("Spectrum selected");
    }
}
```

15. Visualisation.js

```
//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){
•
•         //call unselectVisual in currentVisual
•         if(this.selectedVisual!=null)
•
   if(this.selectedVisual.hasOwnProperty("unSelectVisual")){
•             this.selectedVisual.unSelectVisual();
•         }
•
•            for(var i = 0; i < this.visuals.length; i++){
•                 if(visName == this.visuals[i].name){
•                      this.selectedVisual = this.visuals[i];
•
   if(this.selectedVisual.hasOwnProperty("selectVisual")){
•                    this.selectedVisual.selectVisual();
•                 }
•                 }
•            }
•       };
•  }
```

16. wavepattern.js

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

    // draw the waveform to the screen
    this.draw = function () {
        push();
        noFill();
        strokeWeight(2);

        // calculate the waveform from the fft
        var wave = fourier.waveform();

        // Draw first wave with red gradient
        stroke(255, 0, 0);
        beginShape();
        for (var i = 0; i < wave.length; i++) {
            var x = map(i, 0, wave.length, 0, width);
            var y = map(wave[i], -1, 1, 0, height);
            vertex(x, y);
        }
        endShape();

        // Draw second wave with green gradient
        stroke(0, 255, 0);
        beginShape();
        for (var i = 0; i < wave.length; i++) {
            var x = map(i, 0, wave.length, 0, width);
            var y = map(wave[i], -1, 1, height / 4, height * 3
/ 4);
            vertex(x, y);
        }
        endShape();

        // Draw third wave with blue gradient
        stroke(0, 0, 255);
        beginShape();
        for (var i = 0; i < wave.length; i++) {
            var x = map(i, 0, wave.length, 0, width);
            var y = map(wave[i], -1, 1, height / 2 - 100,
height / 2 + 100);
            vertex(x, y);
```

```
        }
        endShape();

        // Draw fourth wave with yellow gradient
        stroke(255, 255, 0);
        beginShape();
        for (var i = 0; i < wave.length; i++) {
            var x = map(i, 0, wave.length, 0, width);
            var y = map(wave[i], -1, 1, height / 2, height);
            vertex(x, y);
        }
        endShape();

        // Draw points along the waveform
        stroke(255, 0, 255);
        strokeWeight(4);
        for (var i = 0; i < wave.length; i += 5) {
            var x = map(i, 0, wave.length, 0, width);
            var y = map(wave[i], -1, 1, 0, height);
            point(x, y);
        }

        // Draw filled area under the first waveform with rainbow gradient
        noStroke();
        for (var i = 0; i < wave.length; i++) {
            var x = map(i, 0, wave.length, 0, width);
            var y = map(wave[i], -1, 1, 0, height);

            var r = map(i, 0, wave.length, 255, 0);
            var g = map(i, 0, wave.length, 0, 255);
            var b = map(i, 0, wave.length, 255, 0);

            fill(r, g, b, 100); // Rainbow color with transparency
            beginShape();
            vertex(x, y);
            vertex(x + (width / wave.length), y);
            vertex(x + (width / wave.length), height);
            vertex(x, height);
            endShape(CLOSE);
        }
```

```
•
•           pop();
•       };
•
•       this.unSelectVisual = function () {
•           console.log("WavePattern unselected");
•       }
•
•       this.selectVisual = function () {
•           console.log("WavePattern selected");
•       }
•   }
•
•
```