Computing Project Report – Music Visualisation

Rahul Mohite, Santiago I Lopez P

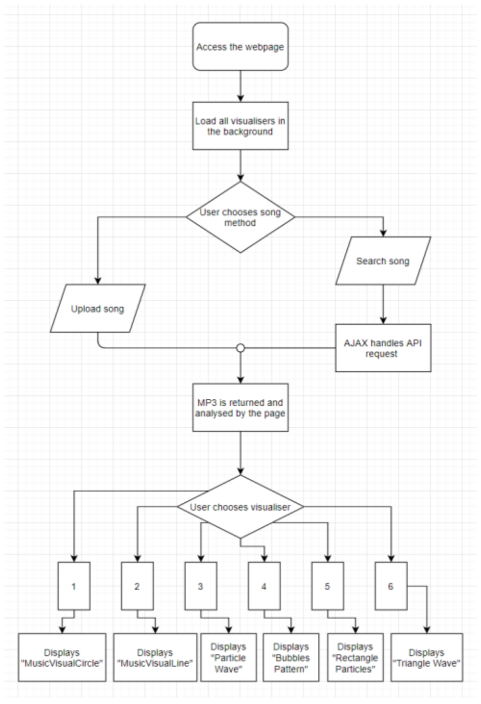
As part of the Computing Project, we were required to decide to work individually or as a team. After choosing to work as a team of 2, we selected the Music Visualisation option – which requires music to be played and a representation of it to be visible from the website we built using the p5.js library.

We also chose to go for the use of a free, open API provided by Deezer, this would allow the user to search for their preferred songs, listen to it and view how the visualisers analyse them. Due to this API being free, there are limitations, such as: the songs played will be 30 second samples. To get the full track, an integration with a premium account with Spotify (or another provider that has this functionality) would be necessary.

Documentation

To begin this work, we planned and designed the outcome. The user would see the options to choose from and the program would respond accordingly.

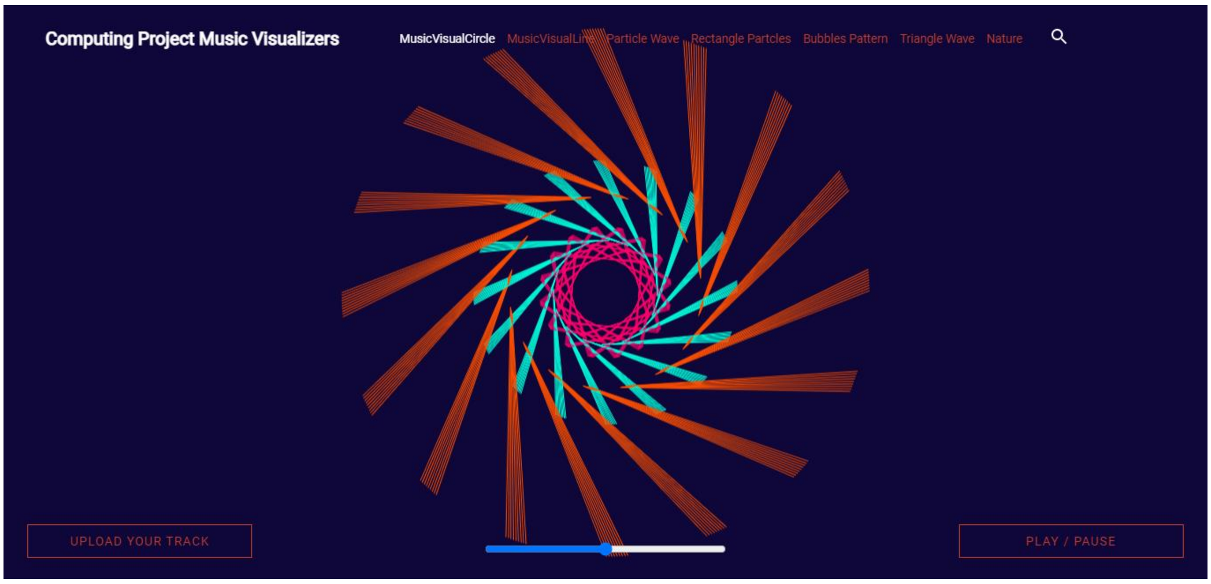
This is demonstrated visually by the following flowchart we created:


Access the webpage
Load all visualisers in the background
User chooses song method
Upload song
Search song
AJAX handles API request
MP3 is returned and analysed by the page
User chooses visualiser
1
2
3
4
5
6
Displays "MusicVisualCircle"
Displays "MusicVisualLine"
Displays "Particle Wave"
Displays "Bubbles Pattern"
Displays "Rectangle Particles"
Displays "Triangle Wave"

Flowchart / Use case representing the trajectory of a user.

There are a total of 6 visualisers, the user can navigate through them by clicking them. Each one of these have their own drawing patterns that make them unique. However, they have a few aspects in common, for example:

The first visualisation has pink, blue and orange colour strips (from inner most to outer most) that represent the noise from the music being played.



Example of the "MusicVisualCircle" visualisation.

The circle like form is a shared shape amongst all the visualisers.

One of the techniques we used was mouse detection – when the mouse is placed on the top of the screen, the visualisations are at their most zoomed out position, however, when the mouse is dragged down, it gives the impression of zooming in as the patterns cover the whole screen. The page will also detect when the mouse is on the left or right side of the screen and will alter the pattern of the shapes accordingly.

Another of the techniques we used was, in the Search button at the top right corner, a pop up will appear with search bar for the user to enter the name of a song of their preference, this request will then be handled by the Deezer servers.

We have also implemented recurring functions, which execute when the user interacts with them no matter how many times, they are as follows:

- "Play / Pause" button at the bottom right corner will play a selected song or stop it if it is being played.
- "Upload your track" button at the bottom left corner will allow the user to upload a song of their own to be played and analysed.
- At the bottom there is a slider that the user can interact with to alter the volume level of the song being played.

Implementation

The code used in the sketch.js file consists of some of the provided JavaScript code from the case study for Music Visualisation. Which includes the logic behind adding all the existing visualisations to the "visuals" array property belonging to the "Visualisations" class, to then create user interaction with the use of the "ControlsAndInput" class.

Each visualisation has its own file, where it is stored as a class. We decided to use Object Oriented Programming (OOP) for making these visualisations, as these have their own functions that can be easily called on the instance of the class.

The process of analysing sound is similar in the sense that we use the "p5.Amplitude()" and "p5.FFT()" constructors, but these are portrayed differently each time. There are three different sections of the code that are taken into account, which are as follows: Bass, Mid, Treble; these are labelled by comments in the code.

With the use of the "map()" function from the p5.js library, we were able to make the visualisations more interactive and let the user handle the mouse to alter the shape sizes or patterns generated.

The integration with Deezer's free API was made such that a GET type request would be sent when the user would search for a song/artist/album – and a JSON response is received with the list of results for the user to select. The program has already stored the values needed, so that when a song is clicked it can determine what resources it needs from the same JSON response, minimising the need of creating another request. The key "preview" has a value as a URL leading to a .mp3 file, which is then loaded using "loadSound()" and analysed as normal.

We split the work and used GitHub to upload our changes and resolve conflicts, creating a new branch when a big task was done. This was done to maintain version control and preventing us from overwriting each other's work.

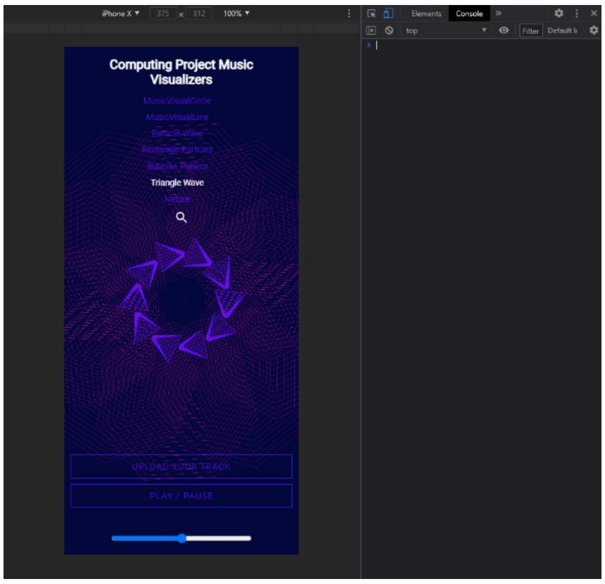
Evaluation

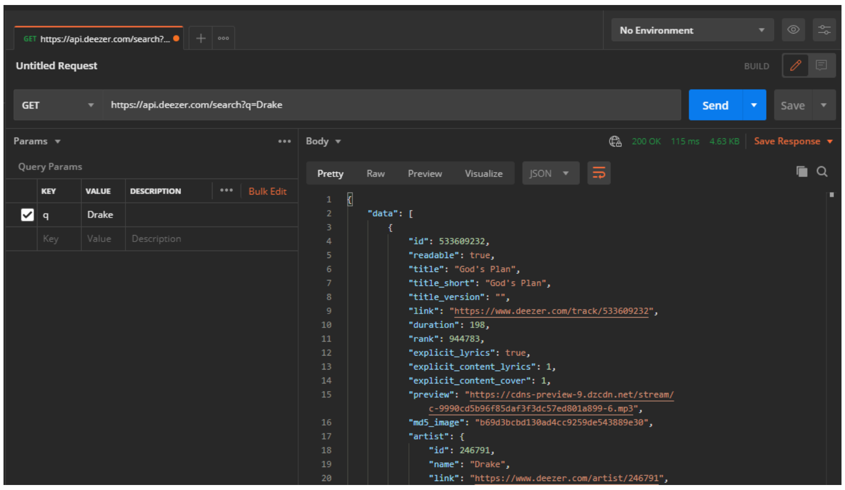
For the planning phase, we used the diagram provided in the "Documentation" section of this document to determine how the software will work.

By the Development and testing phases, we had an overview of what exactly each one of us will be working on.

The manual test we will perform would include Design and Semantic.

We used the Chrome browser's device screen height and width tool to check that the page is responsive and that the visualisers also adhere to the responsiveness; by adjusting the camera, it gives the impression that the designs have the same size as it normally would with computer screens. Here is the result when testing using the iPhone X's screen:





GET https://api.deezer.com/search?q=Drake

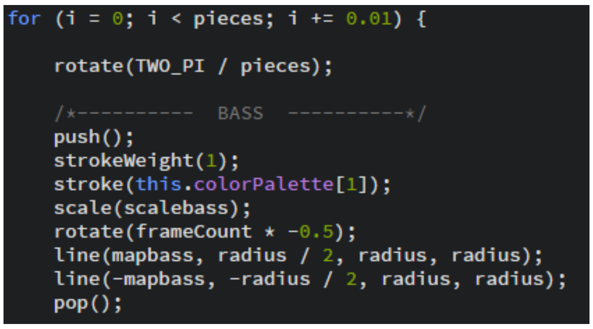
KEY	VALUE	DESCRIPTION
✓ q	Drake	
Key	Value	Description

```
{
    "data": [
        {
            "id": 533609232,
            "readable": true,
            "title": "God's Plan",
            "title_short": "God's Plan",
            "title_version": "",
            "link": "https://www.deezer.com/track/533609232",
            "duration": 198,
            "rank": 944783,
            "explicit_lyrics": true,
            "explicit_content_lyrics": 1,
            "explicit_content_cover": 1,
            "preview": "https://cdns-preview-9.dzcdn.net/stream/
                c-9990cd5b96f85daf3f3dc57ed801a899-6.mp3",
            "md5_image": "b69d3bcbd130ad4cc9259de543889e30",
            "artist": {
                "id": 246791,
                "name": "Drake",
                "link": "https://www.deezer.com/artist/246791",
```

There was a lot of testing involved in the implementation process relating to the API. The program "Postman" proved extremely useful at handling requests prior to coding it. This allows us to have a clear idea of the data we will be receiving along with its structure.

We experienced two main challenges, which were maths and proxy related.

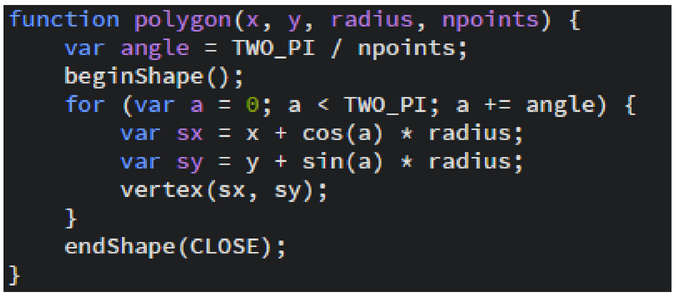
Analysing a sound was the first part, the next was to tell the software exactly what to do with it. Because of the patterns we had in mind, we needed the use of mathematics that we have gained from "Numerical Maths". However, there was still some uncertainty around how to implement figures that would look pleasant. To overcome this issue, we used our researching skills and gathered enough data to work the math out ourselves.



```
for (i = 0; i < pieces; i += 0.01) {

    rotate(TWO_PI / pieces);

    /*----------  BASS  ----------*/
    push();
    strokeWeight(1);
    stroke(this.colorPalette[1]);
    scale(scalebass);
    rotate(frameCount * -0.5);
    line(mapbass, radius / 2, radius, radius);
    line(-mapbass, -radius / 2, radius, radius);
    pop();
```

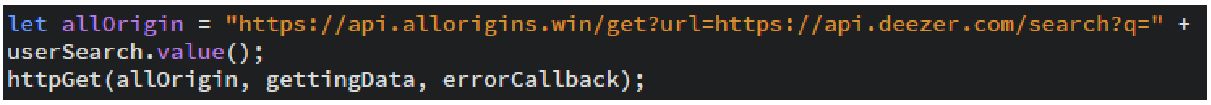


```
function polygon(x, y, radius, npoints) {
    var angle = TWO_PI / npoints;
    beginShape();
    for (var a = 0; a < TWO_PI; a += angle) {
        var sx = x + cos(a) * radius;
        var sy = y + sin(a) * radius;
        vertex(sx, sy);
    }
    endShape(CLOSE);
}
```

Solved mathematical complications.

The next issue we experienced was related to the API. Deezer supports JavaScript calls, however, these need to use their "JavaScript SDK" platform, as Cross-Origin Resource Sharing (CORS) is disabled. In order to save time, we opted on going for another option: a free proxy that allows requests to be made without the need of their SDK.

We were working with a proxy called "CORS Anywhere", though it fixed the CORS error, it generated an inconvenience: the user had to go to their page, click "Consent" and go back to our page. As we were planning on fixing this with the use of an iframe, the requests were no longer getting accepted at all. A change of proxy was necessary – the one we currently are using is called "All Origins" to avoid the Same-Origin policy.



```
let allOrigin = "https://api.allorigins.win/get?url=https://api.deezer.com/search?q=" +
userSearch.value();
httpGet(allOrigin, gettingData, errorCallback);
```

Extract from the code showing how "All Origins" allows policy bypassing.

To conclude, the implementation phase of this development was challenging. Nevertheless, we managed to overcome these challenges by taking advantage of the internet and our researching skills.

There was some re-use of code in the classes that, if given enough time, we could have re-written to have a parent class which shares methods amongst its child classes through inheritance – making the code more efficient.