6-13-2017

SCLOrk 2.0

Juan Miguel Baluyut
Santa Clara University, jbaluyut@scu.edu

Jowy Curameng
Santa Clara University, jcurameng@scu.edu

Santa Clara University
DEPARTMENT of COMPUTER ENGINEERING

Date: 6/13/2017

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Juan Miguel Baluyut & Jowy Curameng

ENTITLED

SCLOrk 2.0

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

Daniel W. Lewis
THESIS ADVISOR

[illegible]
DEPARTMENTCHAIR

SCLOrk 2.0

by

Juan Miguel Baluyut & Jowy Curameng

SENIOR DESIGN PROJECT REPORT

Submitted in partial fulfillment of the requirements
for the degree of
Bachelor of Science in Computer Science and Engineering
School of Engineering
Santa Clara University

Santa Clara, California

June 13, 2017

Abstract

For the past 10 years, the Laptop Orchestra is an emerging interdisciplinary field that involves Computer Science, Engineering, and music. In the past 4 years, Santa Clara University has developed it's own Laptop Orchestra ensemble called SCLork. For this project, we will be making improvements to the Laptop Orchestra by incorporating a mobile phone element for audience participation. In addition, we will be designing an original composition using computer science sound synthesis techniques.

Our mobile element is implemented as a mobile-friendly website that will be accessed by audience members. This website will be used in tandem by both audience and performers. Data received by the website will be sent to a program called SuperCollider to sort data and to play a musical melodies.

Acknowledgements

This project is for the laptop orchestra community. We would like to dedicate our work to our family and friends. Thank you for supporting us through all the ups and down of our college journey. We want to thank both of our advisors: Bruno Ruviaro for the inspiration and direction for this project, and Dan Lewis for organizing and assisting in our documentation. To Danny Hernandez, our senior design mentor, thank you for your assistance and recommendations for project implementations. We would lastly like to thank Paul Ahrens, Miles Elliot, and the rest of the Santa Clara Laptop Orchestra Class of 2017! The help and feed back from you all help us create the fun and interactive final product that we have today.

Table of Contents

List of Figures

1 Introduction

Collaboration is a powerful tool and resource for all people. Its an educational approach to teaching and learning that helps a group of people collectively solve a problem, create something new or complete a task. Collaboration is a very useful resource for engagement, learning, and growth. Within the last ten years, there has been a new sort of collaboration. Laptop orchestra is a newer collaboration of music that is driven purely through electronic sound and technology.

Laptop orchestras helps solve two problems: accessibility of instruments and lack of collaborations in electronic music. Access to creating music can be limited for users because regular musical instruments are hard. It can come from a lack of musicality or limited available time to develop music skills. Creating electronic music is perceived as an lonely process. This notion is from images of a DJ playing music all by their lonesome on a stage. Laptop orchestra addresses both of these issues. Any level of musical background can be involved, and laptop orchestra can involve multiple disciplines like computer science and music.

With all the issues already addressed by laptop orchestra, we want to take laptop orchestra one step further. Our proposal is a program that will involve more audience participation. Currently the participation between laptop orchestra and audience is non-existent. We will be designing a way to allow audience members to be more actively participatory in the musical experience. Audience participation enhances the experience and makes it more attentive. The improvisational element also keeps the performers can add a new dynamic to the overall performance.

We will be working with Santa Claras laptop orchestra, also known as SCLOrk.

2 Requirements

The following requirements define the goals of the project outlined in the introduction. The functional requirements define features that must be done for the project to be considered a success, while the non-functional requirements define how the functional requirements are achieved. Requirements are categorized into critical, recommended, and suggested. Critical requirements are absolutely necessary, recommended are highly desirable, and suggested requirements are not necessary but would be very nice to add.

Design constraints are criteria that the solution must adhere to. The constraints are set by our goals of what we want the audience to do with the players on stage. The technologies we are using will also place constraints on our project. Users in this paper are referring to both audience and orchestra performers, as they are participating in performance together.

2.1 Functional Requirements

Critical:

1. Must work with iOS and android devices.
2. The network must handle data input from one user
3. Must work in real time during performance

Recommended: 1. Can handle all types of mobile devices

2. Network can handle data

2.2 Non-Functional Requirements

Critical:

1. A simple user interface for audience members
2. System has good performance for users
3. System maintains good reliability for users

Suggested: 1. Maintainable so future groups can add new performance features to the app

2.3 Design Constraints

Design goals are to create an easy-to-access website with an easily navigable UI.

Constraint

1. Design must be user-friendly
2. Design must be compatible with at least iOS devices

3 Use Cases

The following outlines anticipated uses of the system. Use cases below describes actions by the conductor and users. Users are referring to both audience and orchestra players. They play the music together.

3.1 Use Case 1:

Conductor prepares musical components that will be used for performance

Name: Upload Music

Goal: Upload for performance

Actors: Conductor

Pre-conditions:

- Conductor know data he is uploading

Post-conditions:

- Conductor knows when to stop recording data

Steps:

1. Conductor choose data to upload
2. Upload music score
3. Upload SuperCollider functions
4. Upload instrument files
5. Send initial start value for performance
6. Run server to receive data from website
7. Stop server when performance is over

Exceptions:

1. Wrong file is uploaded
2. Server does not run and receive data
    - Wrong music is played during performance
    - No music is played

3.2 Use Case 2:

Actions done by both the audience and performers for music performance.

Name: Website interaction

Goal: Send data from website to server

Actors: Audience

Pre-conditions:

- Audience is instructed by conductor to access website

Post-conditions:

- Users successfully participate in music performance

Steps:

1. Log on to website
2. Vote on music and tempo options
3. See and hear results of voting

4 Architectural Diagram

We have chosen to use an architecture based on the client/server model. In our architecture, the conductor will work with the central server. The central server will store all the necessary information such as sounds to be played. The server will communicate with both the orchestra player and the audience. All mobile devices from audience and all player laptops will be connected to central server computer.

The users will be sending voting information via website to webs server. Web server receives all data and tallies with counter to record all user-input data. Data is then send to conductor computer. Conductor computer receives data and sends various music melodies according to winner of vote. Orchestra computers receives melodies and send sound to speakers for all parties to listen.

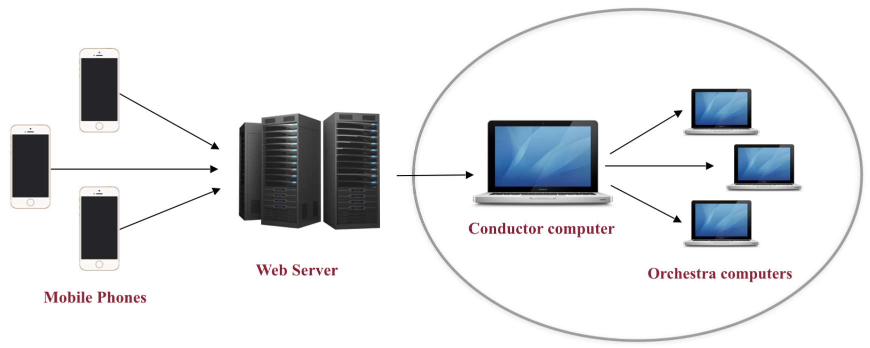


Figure 1: Diagram of data travel from users to conductor computer

5 Technologies Used

Below are a list of technologies we plan to use for our project:

HTML	Organized web page and buttons
Java Script	Used to created interactive actions on website
CSS	Made website and interactions pretty and visually appealing
NodeJS	Web server to receive data from many devices to its final destination ins a timely manner
SuperCollider	Open source platform for audio synthesis and algorithmic composition. Used to create music score and to handle server operations between conductor and orchestra computers
Digital Ocean	Cloud infrastructure provider. Used to host nodeJS server on the World Wide Web
NameCheap	Domain name provider. Used to mask IP of website hosted by digital ocean. Allowed user to access website via URL instead of typing in an IP address
GitHub	Hosting service for project code

6 Design Rationale

We chose to use SuperCollider as our primary programming language. The main reason we chose this is because of our familiarity with it. In addition, SuperCollider can handle with synthesizing new types of sounds necessary to

Due to technologies available, our design was limited to the laptops and hardware provided by the Music Department.

GitHub is a very strong documentation and collaboration tool, as it allows us to be able to not only store previous versions of our code, in case they are needed, but also allow us to be able to make our own individual changes while still being able to quickly rollback to previous versions. For example, the final version relied heavily on a much earlier version of the project.

Our group decided to use GitHub as it offered a configuration management system that allowed team members to coordinate our progress with our application. GitHub also acts as our repository for our artifacts for the project.

7 Testing Procedure

As part of our testing plan, we allocated time during the Winter Quarter to get most of our testing done. Spring quarter was used for some additional testing and debugging as well.

During our Alpha tests, we ran Shellscript simulations to check if our web application could handle the load of multiple users. The first test we ran a simulation of 500 users, which S.C.L.O.R.K. 2.0 handled satisfactorily.

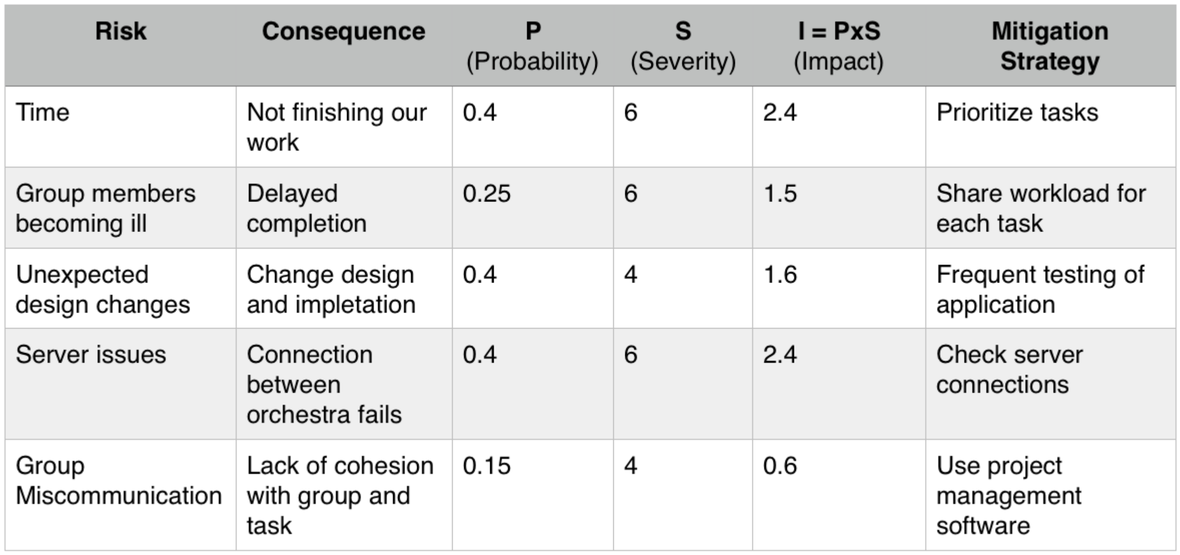
The main method of testing were our Beta Tests, which involved simulating a live performance. First, we would tell the audience a little bit of our product. We would give them the URL to access the application and give them a short demonstration on how to navigate the web application. We would tell them how each button corresponded to different melody.

Our Beta Tests consisted of three trials: a dress rehearsal, a scheduled concert, and an art exhibit show at Anno Domini. The dress rehearsal consisted of the Santa Clara University Laptop Orchestra as they got familiar with the interface. This group consists mostly of musicians and engineers. These people have more experience with the difference intersections of music and engineering. In addition to the Santa Clara University Laptop Orchestra, the concert involved a live audience as well. This group ranged from regular students to parents who all had different levels of expertise with music and technology.

We did receive some constructive criticism and feedback as well as part of our testing plan. After each of our performances, we purposefully asked the audience for some feedback on the product. One feedback was to register an actual URL for our product. During our performances, we used only a local IP Address for them to access. The long string of numbers was cumbersome for people to type out and took away from the experience. Another area our product could have improved was an explanation of the buttons. The people with less experience were not familiar with the musical jargon that we used, so a lot of them resorted to button mashing rather than having a thoughtful and immersing experience. Overall though, our product was well received.

8 Risk Analysis

For our risk analysis table, as seen in Figure 2, it is broken down into six different categories.



Risk	**Consequence**	**P** (Probability)	**S** (Severity)	**I = PxS** (Impact)	**Mitigation Strategy**
Time	Not finishing our work	0.4	6	2.4	Prioritize tasks
Group members becoming ill	Delayed completion	0.25	6	1.5	Share workload for each task
Unexpected design changes	Change design and impletation	0.4	4	1.6	Frequent testing of application
Server issues	Connection between orchestra fails	0.4	6	2.4	Check server connections
Group Miscommunication	Lack of cohesion with group and task	0.15	4	0.6	Use project management software

Figure 2: Risk Analysis

Each risk has consequences for what happens if the risk actualizes. It also carries the probability of occurring, along with the severity of the risk. Probability and severity factor into the impact the risk has on the project. Finally, the table outlines mitigation strategies our team has prepared to help lessen either the probability or severity of the risk. This helps lessen the impact of the risk should it still happen.

The largest risk to our project is our team failing to deliver an application that the audience can use. The other major risks all involve issues that impact our schedule for completing our project, such as team members becoming ill, or bugs in the system. While this is a relatively small project, it is still possible that the final deliverable will not be ready by the due date.

One large risk for this project is the server failing to connect each of the players during a performance. The server synchronizes all the laptops during the performance and it is suppose to connect the audience with the performers. Our main goal is to connect the audience with the performers. Without a working server, we do not achieve our goal.

9 Development Time-line

Below is brief overview of our project time-line follow by a figure that shows how work will be divided

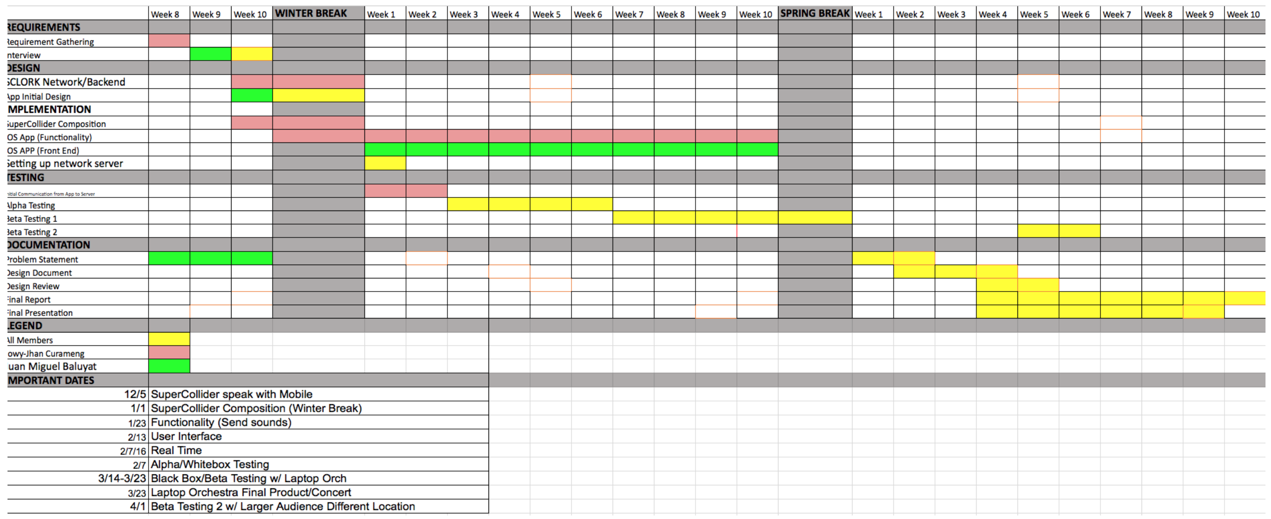
Fall	Initial planing of project. Design documentation of project will begin. Determine "What audience will do to interact with players" Determine how to determine establish server
Winter	Testing of server and music composition Alpha testing Week 2-7 Beta testing Week 8-10 Compose musical piece week 1-7 Laptop Performance Thursday Week 10
Spring	Post performance documentation Finish documentation of project Establish what can be improved for project for future senior design groups Senior Design Conference

Most of our time-line was built over Spring Quarter 2016 and Summer Quarter 2016 with the help of our advisor Bruno Ruviaro. In this time-line, we divided up components into Fall Quarter 2016, Winter Quarter 2017, and Spring Quarter 2017.

In Fall Quarter 2016, we hoped to finished three components. These components were the initial planning of the project (i.e. how to establish the server and other technical components), the initial draft of the design document, determination of the scope of the project (i.e. How the audience will interact with the Laptop Orchestra). We allocated time for the planning stages of the design process to ensure a higher probability of success for our project.

In Winter Quarter 2017, we hoped to finish two components. These components were the implementation of the composition with the web application and the testing during live performances. We allocated time for testing and implementation for this whole quarter because we wanted to get feedback as early as possible. This way we could possibly implement new changes by the end of the project.

In Spring Quarter 2017, we hope to finished the documentation and the Senior Design Conference. By planning our time this way, we give us time to ensure that we can handle any roadblocks that come our way.



	Week 8	Week 9	Week 10	WINTER BREAK	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	SPRING BREAK	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10
EQUIREMENTS																									
equirement Gathering	🟥																								
nterview		🟩	🟨																						
ESIGN																									
CLORK Network/Backend			🟥	🟥																					
pp Initial Design			🟩	🟨																					
MPLEMENTATION																									
uperCollider Composition			🟥	🟥																					
OS App (Functionality)				🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥											
OS APP (Front End)					🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩											
etting up network server					🟨																				
ESTING																									
nitial Communication from App to Server					🟥	🟥																			
lpha Testing							🟨	🟨	🟨	🟨															
eta Testing 1											🟨	🟨	🟨	🟨	🟨										
eta Testing 2																				🟨	🟨				
OCUMENTATION																									
roblem Statement	🟩	🟩	🟩													🟨	🟨								
esign Document																	🟨	🟨	🟨						
esign Review																			🟨	🟨					
inal Report																			🟨	🟨	🟨	🟨	🟨	🟨	🟨
inal Presentation																			🟨	🟨	🟨	🟨	🟨	🟨	
EGEND																									
ll Members	🟨																								
owy-Jhan Curameng	🟥																								
uan Miguel Baluyat	🟩																								

MPORTANT DATES	
12/5	SuperCollider speak with Mobile
1/1	SuperCollider Composition (Winter Break)
1/23	Functionality (Send sounds)
2/13	User Interface
2/7/16	Real Time
2/7	Alpha/Whitebox Testing
3/14-3/23	Black Box/Beta Testing w/ Laptop Orch
3/23	Laptop Orchestra Final Product/Concert
4/1	Beta Testing 2 w/ Larger Audience Different Location

Figure 3: Timeline for Project

10 Social Implications

Ethical

We believe that no ethical issues can be brought by our project because of the simplicity of our interface. A choice of right or wrong would be to difficult to make using a musical web application.

Social

The main societal issue that project addresses is the social aspect of human interaction. One of the problems we hoped to address was the need of involving people of various musical skill levels to have a rewarding musical experience. We believe that music is an activity that can be enjoyed by everybody.

In addition we would like to be able to bridge the gap between the Arts and S.T.E.M. education. Through our project, we hope to create a theme of connection and inclusiveness.

Political

We believe that no political issues can be brought by our project unless a political regime is against the use of musical web applications.

Economic

We do not believe that economic issues apply to our product.

Health and Safety

A possible health concern is the strain on eyes and ears as people use the web application. We hope to limit the risk with this by mentioning safe volume limits and limiting screen usage to health amount sof time.

Manufacturability

We do not believe that that manufacturability issues apply to our product.

Sustainability

Our project can be considered sustainable as it can last a long time. The code is reusable and can be accessed by different Operating Systems, laptops, and mobile phones. This way more people can experience S.C.L.O.R.K 2.0

Environmental Impact

As our project involves different networks of phones and laptops, we require lots of energy. We hope to find a way to minimize the energy usage in order to better preserve the world

Usability

From our testing, we have come to a general consensus that our product is easy to learn and understand for the most part. In the future, we hope to keep the user-friendliness as we scale to integrate new features.

Lifelong Learning

This project did help prepare us for future products that we have to develop. It also inspired us to understand the design process fully. Finally, we hope to continue our learning of music, engineering, and the intersection between the two.

Compassion

Through our project we hope to be more aware of the suffering of other people. We hope that our project alleviates that suffering through the enjoyment of music. Music is a social piece that has been shown to alleviate suffering.

11 Conclusion

In the future, we hope to be able to implement three main features for our system that we believe would add to the experience of S.C.L.O.R.K 2.0. The first feature is to implement more audience and orchestra options. At this point, our system only contains four different melodies and the ability to change tempo, which is the speed of the song. In the future, we hope to be able to incorporate different instrument sounds, different key, changes, and more melodies. We hope that by implementing more features that our product does not become to overwhelming for the average user. Another way of adding more audience and orchestra options would be mapping a melody to accelerometer motions. We feel that body motions would be more rewarding than just pressing a button. Everybody would feel more involved in the process of making a piece this way.

Another feature that wed like to implement is data distribution within the orchestra. Wed like our conductor laptops to be able to send information to the player laptops. At this point in time, our system is not able to handle this feature, so we had to play some of our melodies manually rather than having the system handling it.

Lastly, we would like to add data visualization on the web application. During the performances, we had to stream our own screens to prove to the audience that their votes were being tallied. We would like to implement a way to have the audience know how their activity is changing the musical piece (e.g. a tally screen on the phone).

This project taught us many things about the design process of engineering. We learned the importance of implementing early and testing. We realized that a lot of our time was spent on learning. If given another chance, we would like to be able to learn by doing.

Another lesson that we learned was the importance of outside help and feedback. Many times during this process, we realized that we had a hard time finding a certain bug. We wasted lots of time on simple features. This process taught us to not be afraid to ask for help, especially from our industry mentors.

Finally, we learned the importance of designing early and outline technologies. We did not realize the importance of limiting the scope of a project. Our goals were very open-ended and our initial designs reflected it. In the future, we would have liked to spent more time actually implementing a project rather than thinking about the possibilities.

12 References

[1] S. Cavaco, "The Carnegie Mellon Laptop Orchestra," Aug. 2007. [Online]. Available: http://repository.cmu.edu/cgi/viewcontent.cgi?article=1511&context=compsci. Accessed: Dec. 02, 2016.

[2] N. Bowen, "4Quarters,". [Online]. Available: https://ccrma.stanford.edu/~ruviaro/texts/SLEO_2012_Proceedings.pdf. Accessed: Dec. 01, 2016.

[3] SuperCollider, "SuperCollider,". [Online]. Available: http://supercollider.github.io/. Accessed: Dec. 3, 2016.

[4] L. Jiang, "Sensor-Based Music Controller,". [Online]. Available: http://scholarworks.wmich.edu/cgi/viewcontent.cgi?article=1009&context=engineer_design_projects. Accessed: Dec. 3, 2016.

[5] M. Knewston and A. Stratton, "Baton,". [Online]. Available: https://wmich.edu/sites/default/files/attachments/u593/2015/SEDP%2051.pdf.
Accessed: Dec. 02, 2016.

[6] D. Lafond, "Hemispherical Speaker Array for Live Electro-Acoustic Performance,". [Online]. Available: http://scholarworks.wmich.edu/cgi/viewcontent.cgi?article=1019&context=engineer_design_projects. Accessed: Dec. 01, 2016.

13 Appendix

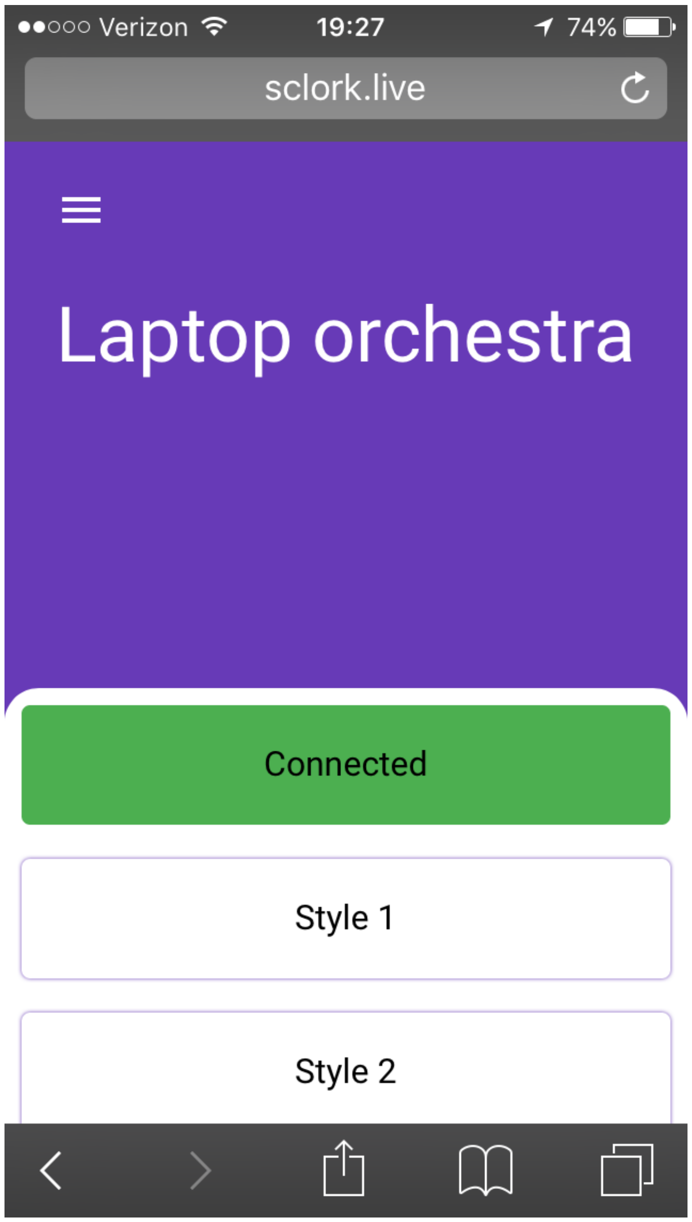


Figure 4: Website landing page part 1

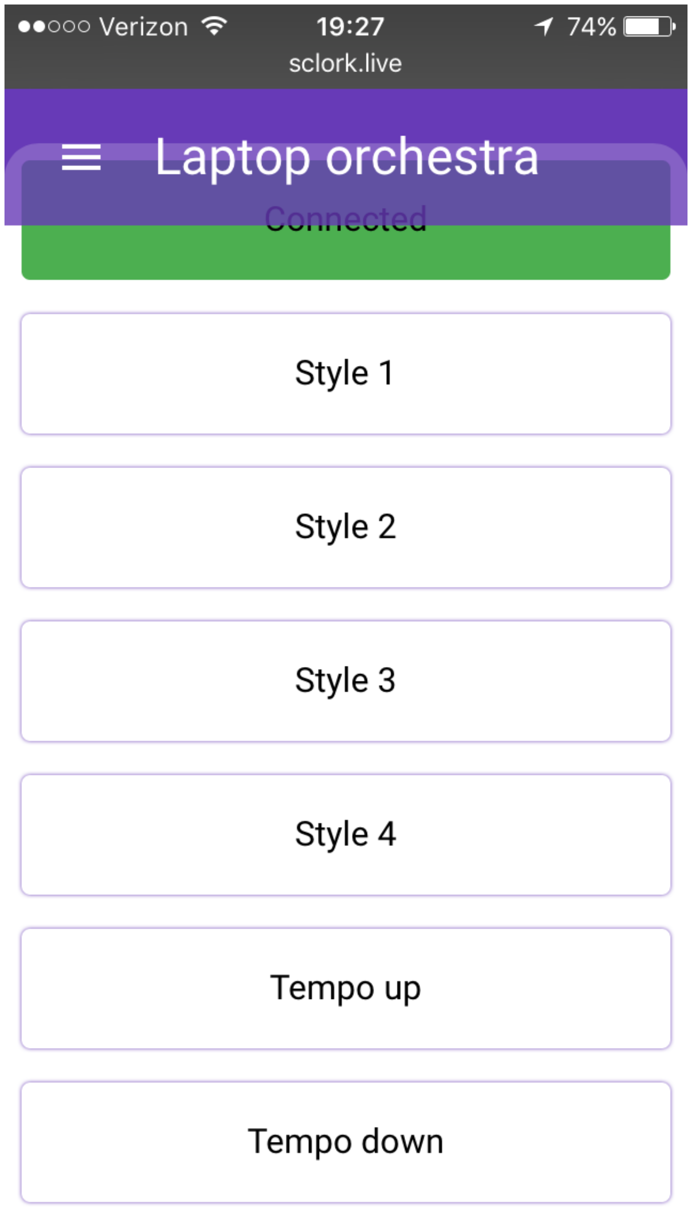
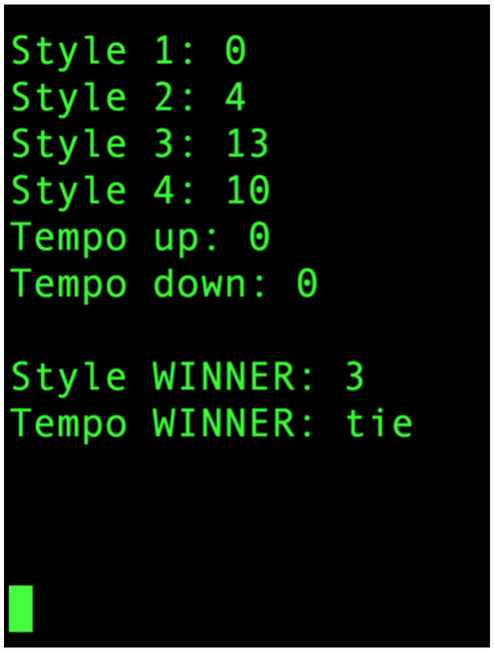
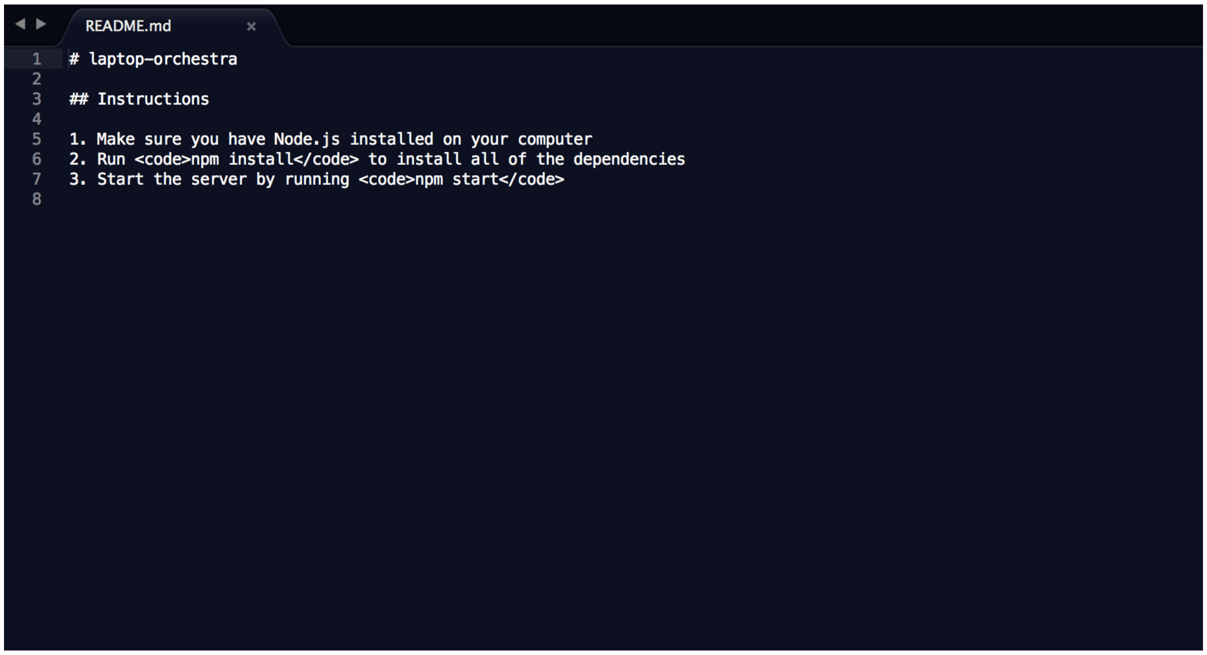


Figure 5: Website landing page part 2



```
Style 1: 0
Style 2: 4
Style 3: 13
Style 4: 10
Tempo up: 0
Tempo down: 0

Style WINNER: 3
Tempo WINNER: tie
```

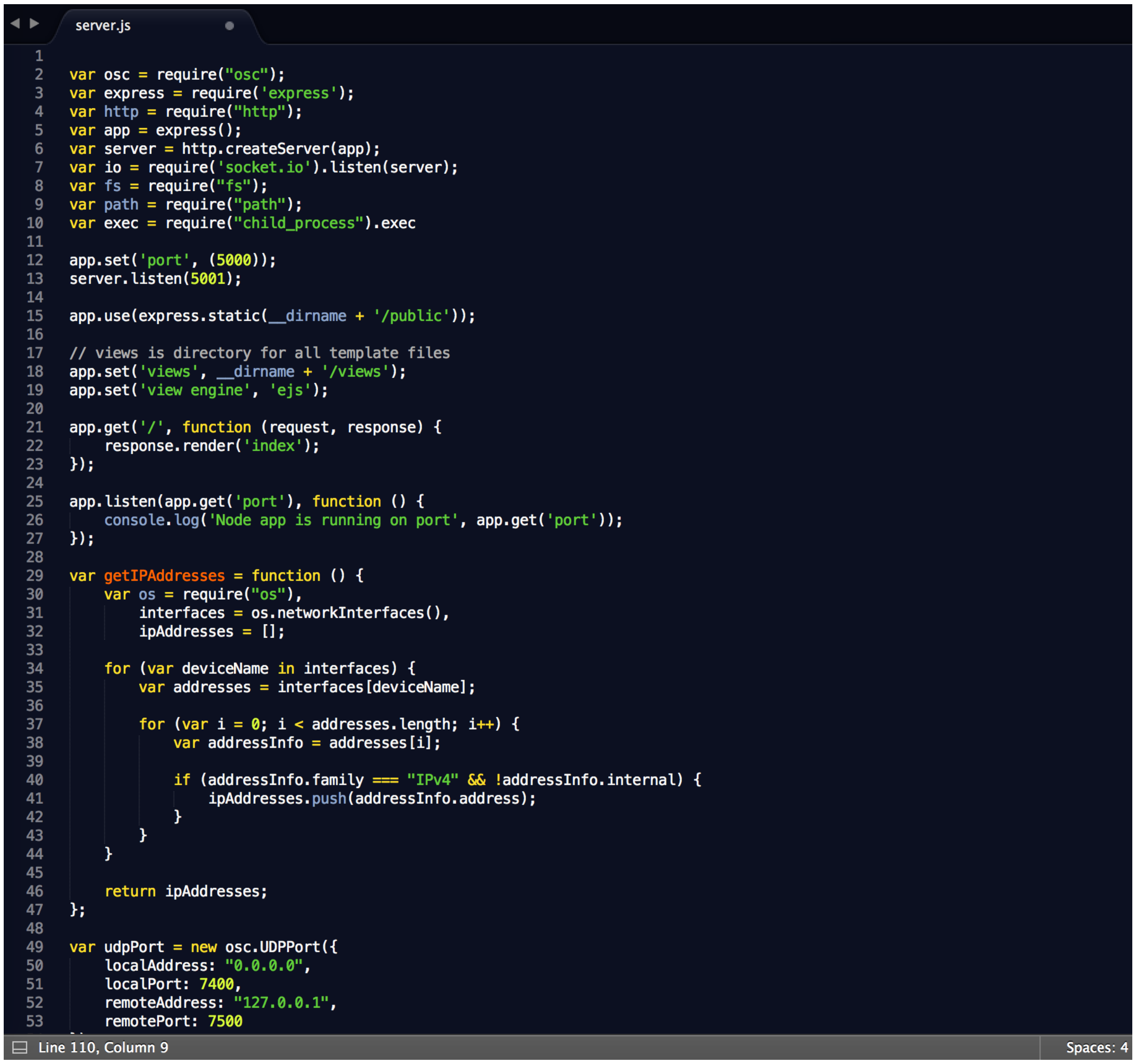
Figure 6: Display of vote counting from node.js server



README.md

```
# laptop-orchestra

## Instructions

1. Make sure you have Node.js installed on your computer
2. Run <code>npm install</code> to install all of the dependencies
3. Start the server by running <code>npm start</code>

```

Figure 7: README for project implementation

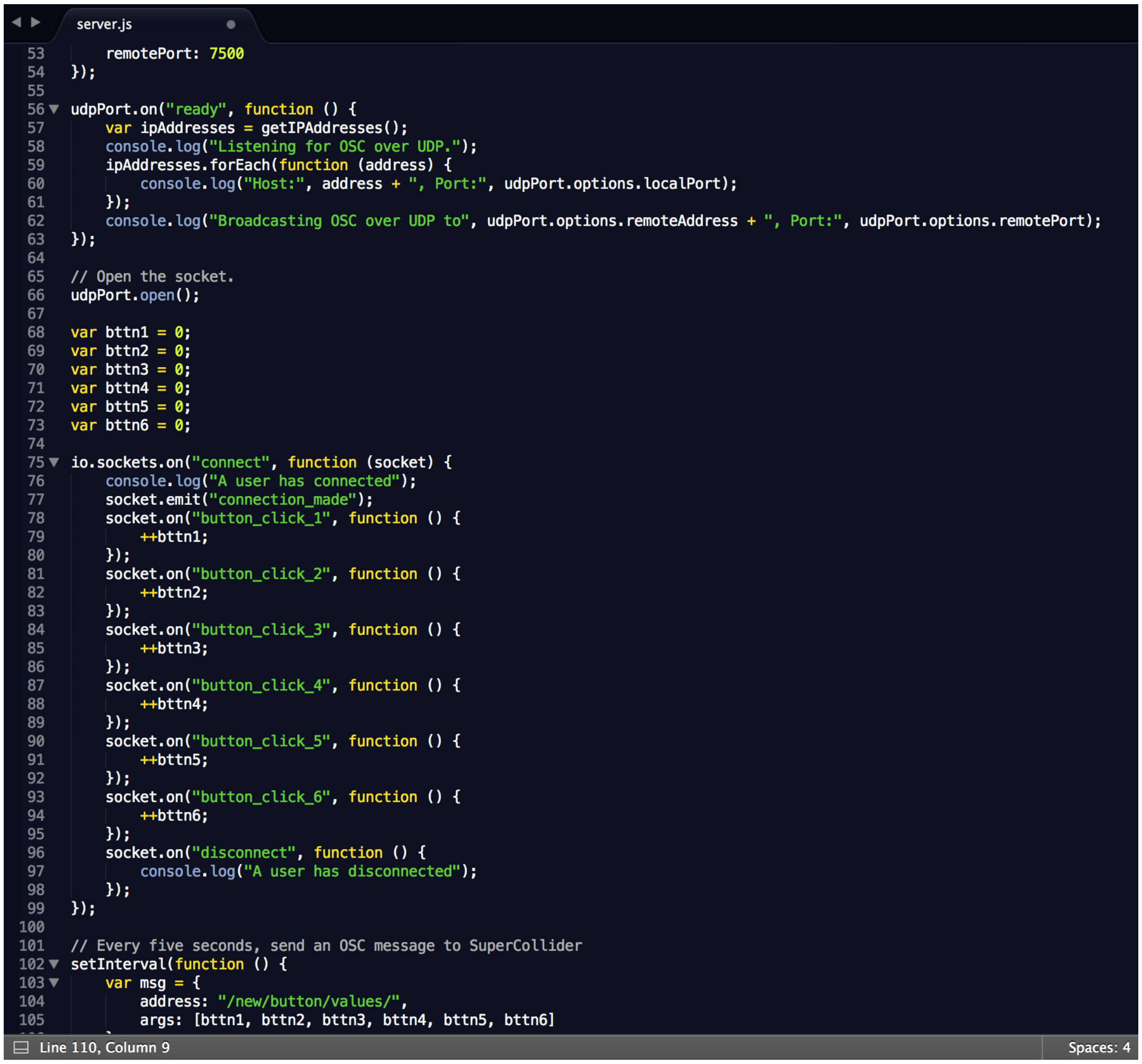


server.js

```

var osc = require("osc");
var express = require('express');
var http = require("http");
var app = express();
var server = http.createServer(app);
var io = require('socket.io').listen(server);
var fs = require("fs");
var path = require("path");
var exec = require("child_process").exec

app.set('port', (5000));
server.listen(5001);

app.use(express.static(__dirname + '/public'));

// views is directory for all template files
app.set('views', __dirname + '/views');
app.set('view engine', 'ejs');

app.get('/', function (request, response) {
    response.render('index');
});

app.listen(app.get('port'), function () {
    console.log('Node app is running on port', app.get('port'));
});

var getIPAddresses = function () {
    var os = require("os"),
        interfaces = os.networkInterfaces(),
        ipAddresses = [];

    for (var deviceName in interfaces) {
        var addresses = interfaces[deviceName];

        for (var i = 0; i < addresses.length; i++) {
            var addressInfo = addresses[i];

            if (addressInfo.family === "IPv4" && !addressInfo.internal) {
                ipAddresses.push(addressInfo.address);
            }
        }
    }

    return ipAddresses;
};

var udpPort = new osc.UDPPort({
    localAddress: "0.0.0.0",
    localPort: 7400,
    remoteAddress: "127.0.0.1",
    remotePort: 7500
```

Line 110, Column 9 Spaces: 4

Figure 8: NodeJS server code part 1

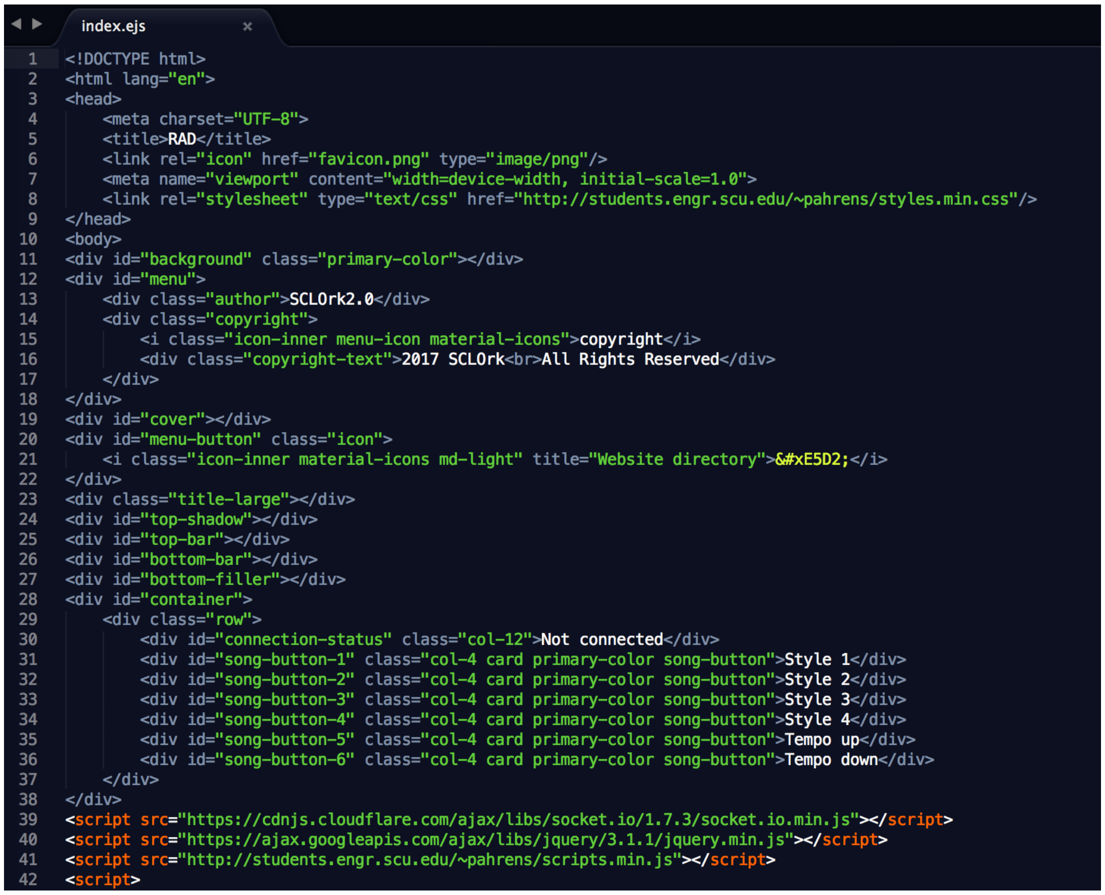


server.js

```
    remotePort: 7500
});

udpPort.on("ready", function () {
    var ipAddresses = getIPAddresses();
    console.log("Listening for OSC over UDP.");
    ipAddresses.forEach(function (address) {
        console.log("Host:", address + ", Port:", udpPort.options.localPort);
    });
    console.log("Broadcasting OSC over UDP to", udpPort.options.remoteAddress + ", Port:", udpPort.options.remotePort);
});

// Open the socket.
udpPort.open();

var bttn1 = 0;
var bttn2 = 0;
var bttn3 = 0;
var bttn4 = 0;
var bttn5 = 0;
var bttn6 = 0;

io.sockets.on("connect", function (socket) {
    console.log("A user has connected");
    socket.emit("connection_made");
    socket.on("button_click_1", function () {
        ++bttn1;
    });
    socket.on("button_click_2", function () {
        ++bttn2;
    });
    socket.on("button_click_3", function () {
        ++bttn3;
    });
    socket.on("button_click_4", function () {
        ++bttn4;
    });
    socket.on("button_click_5", function () {
        ++bttn5;
    });
    socket.on("button_click_6", function () {
        ++bttn6;
    });
    socket.on("disconnect", function () {
        console.log("A user has disconnected");
    });
});

// Every five seconds, send an OSC message to SuperCollider
setInterval(function () {
    var msg = {
        address: "/new/button/values/",
        args: [bttn1, bttn2, bttn3, bttn4, bttn5, bttn6]
```

Line 110, Column 9 Spaces: 4

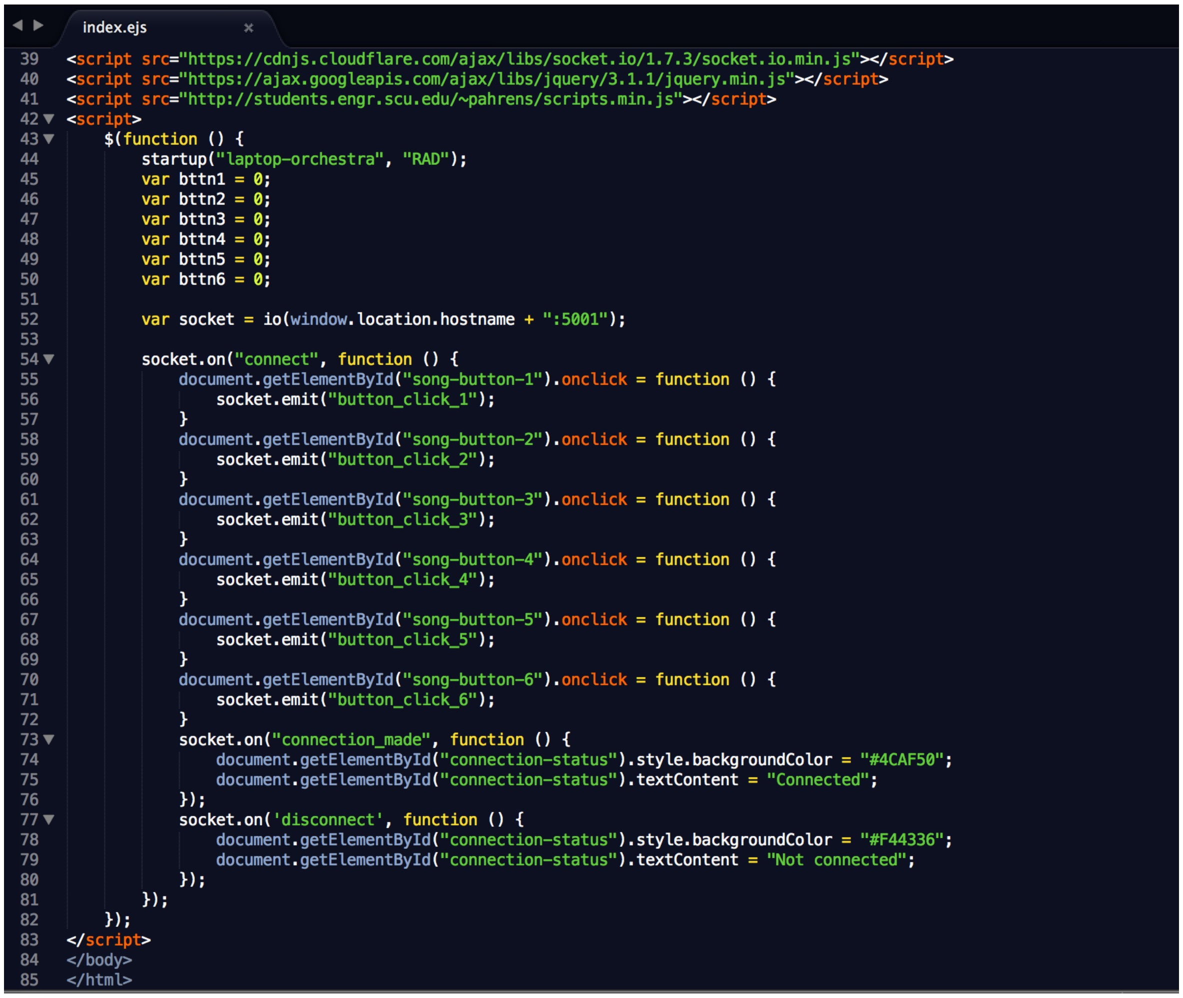
Figure 9: NodeJS server code part 2



index.ejs

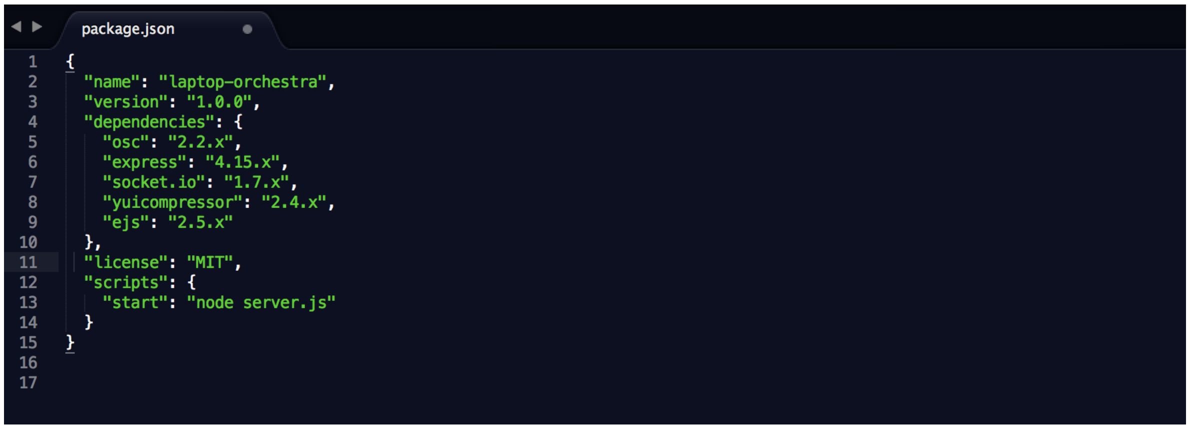
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>RAD</title>
    <link rel="icon" href="favicon.png" type="image/png"/>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" type="text/css" href="http://students.engr.scu.edu/~pahrens/styles.min.css"/>
</head>
<body>
<div id="background" class="primary-color"></div>
<div id="menu">
    <div class="author">SCLOrk2.0</div>
    <div class="copyright">
        <i class="icon-inner menu-icon material-icons">copyright</i>
        <div class="copyright-text">2017 SCLOrk<br>All Rights Reserved</div>
    </div>
</div>
<div id="cover"></div>
<div id="menu-button" class="icon">
    <i class="icon-inner material-icons md-light" title="Website directory">&#xE5D2;</i>
</div>
<div class="title-large"></div>
<div id="top-shadow"></div>
<div id="top-bar"></div>
<div id="bottom-bar"></div>
<div id="bottom-filler"></div>
<div id="container">
    <div class="row">
        <div id="connection-status" class="col-12">Not connected</div>
        <div id="song-button-1" class="col-4 card primary-color song-button">Style 1</div>
        <div id="song-button-2" class="col-4 card primary-color song-button">Style 2</div>
        <div id="song-button-3" class="col-4 card primary-color song-button">Style 3</div>
        <div id="song-button-4" class="col-4 card primary-color song-button">Style 4</div>
        <div id="song-button-5" class="col-4 card primary-color song-button">Tempo up</div>
        <div id="song-button-6" class="col-4 card primary-color song-button">Tempo down</div>
    </div>
</div>
<script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/1.7.3/socket.io.min.js"></script>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></script>
<script src="http://students.engr.scu.edu/~pahrens/scripts.min.js"></script>
<script>
```

Figure 10: HTML index code part1


index.ejs

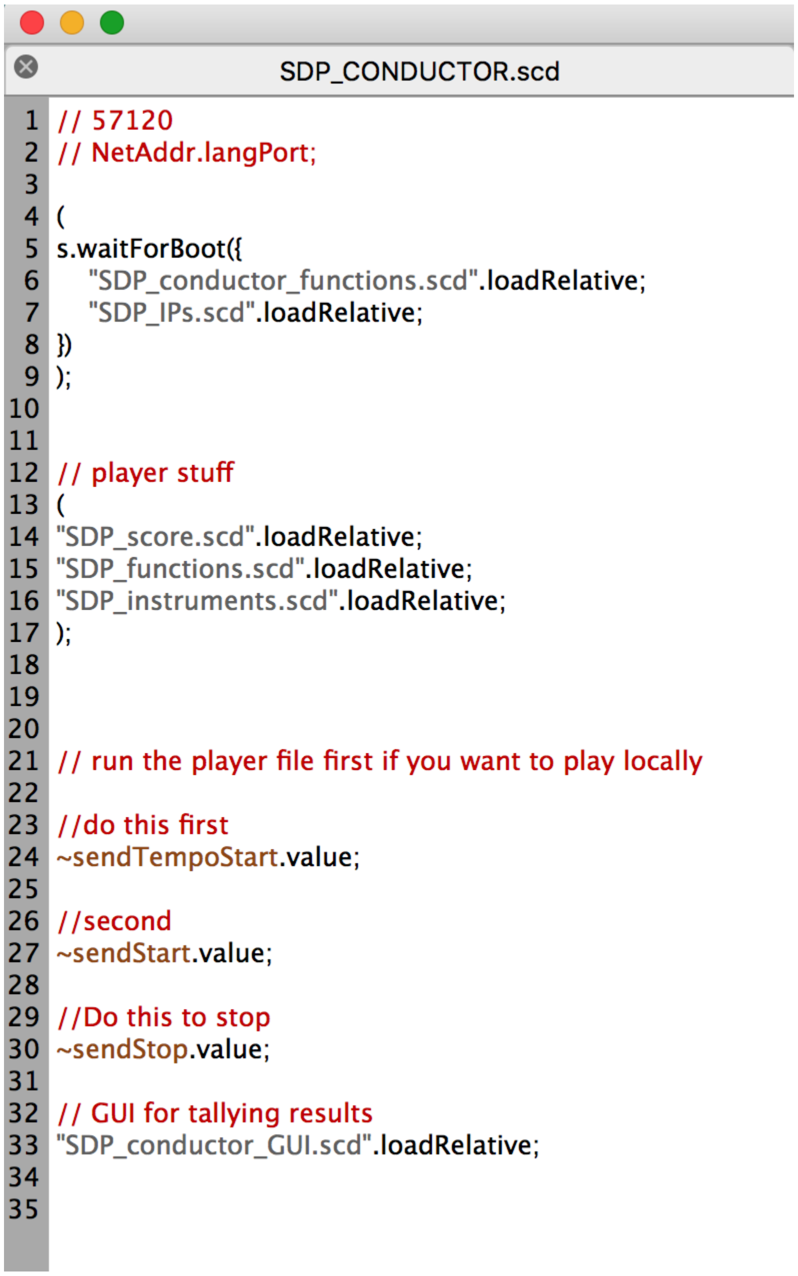
```
<script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/1.7.3/socket.io.min.js"></script>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></script>
<script src="http://students.engr.scu.edu/~pahrens/scripts.min.js"></script>
<script>
    $(function () {
        startup("laptop-orchestra", "RAD");
        var bttn1 = 0;
        var bttn2 = 0;
        var bttn3 = 0;
        var bttn4 = 0;
        var bttn5 = 0;
        var bttn6 = 0;

        var socket = io(window.location.hostname + ":5001");

        socket.on("connect", function () {
            document.getElementById("song-button-1").onclick = function () {
                socket.emit("button_click_1");
            }
            document.getElementById("song-button-2").onclick = function () {
                socket.emit("button_click_2");
            }
            document.getElementById("song-button-3").onclick = function () {
                socket.emit("button_click_3");
            }
            document.getElementById("song-button-4").onclick = function () {
                socket.emit("button_click_4");
            }
            document.getElementById("song-button-5").onclick = function () {
                socket.emit("button_click_5");
            }
            document.getElementById("song-button-6").onclick = function () {
                socket.emit("button_click_6");
            }
            socket.on("connection_made", function () {
                document.getElementById("connection-status").style.backgroundColor = "#4CAF50";
                document.getElementById("connection-status").textContent = "Connected";
            });
            socket.on('disconnect', function () {
                document.getElementById("connection-status").style.backgroundColor = "#F44336";
                document.getElementById("connection-status").textContent = "Not connected";
            });
        });
    });
</script>
</body>
</html>
```

Figure 11: HTML index code part 2



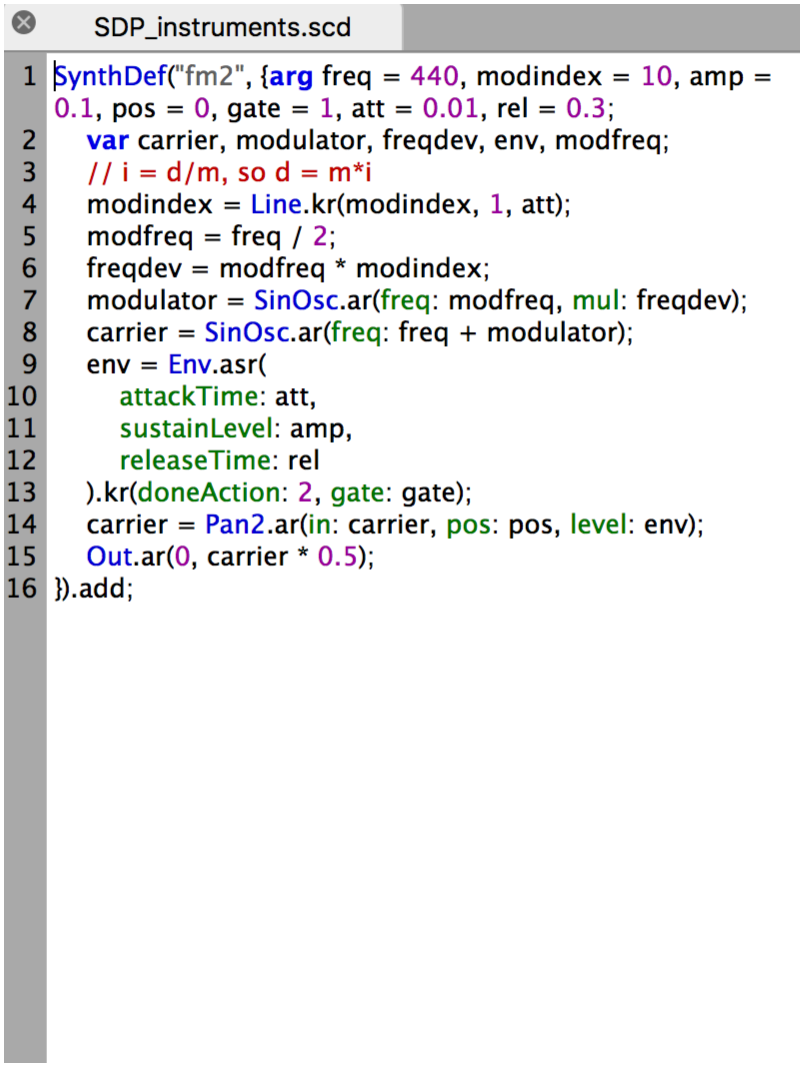
```
{
  "name": "laptop-orchestra",
  "version": "1.0.0",
  "dependencies": {
    "osc": "2.2.x",
    "express": "4.15.x",
    "socket.io": "1.7.x",
    "yuicompressor": "2.4.x",
    "ejs": "2.5.x"
  },
  "license": "MIT",
  "scripts": {
    "start": "node server.js"
  }
}
```

Figure 12: Package.JSON code


SDP_CONDUCTOR.scd

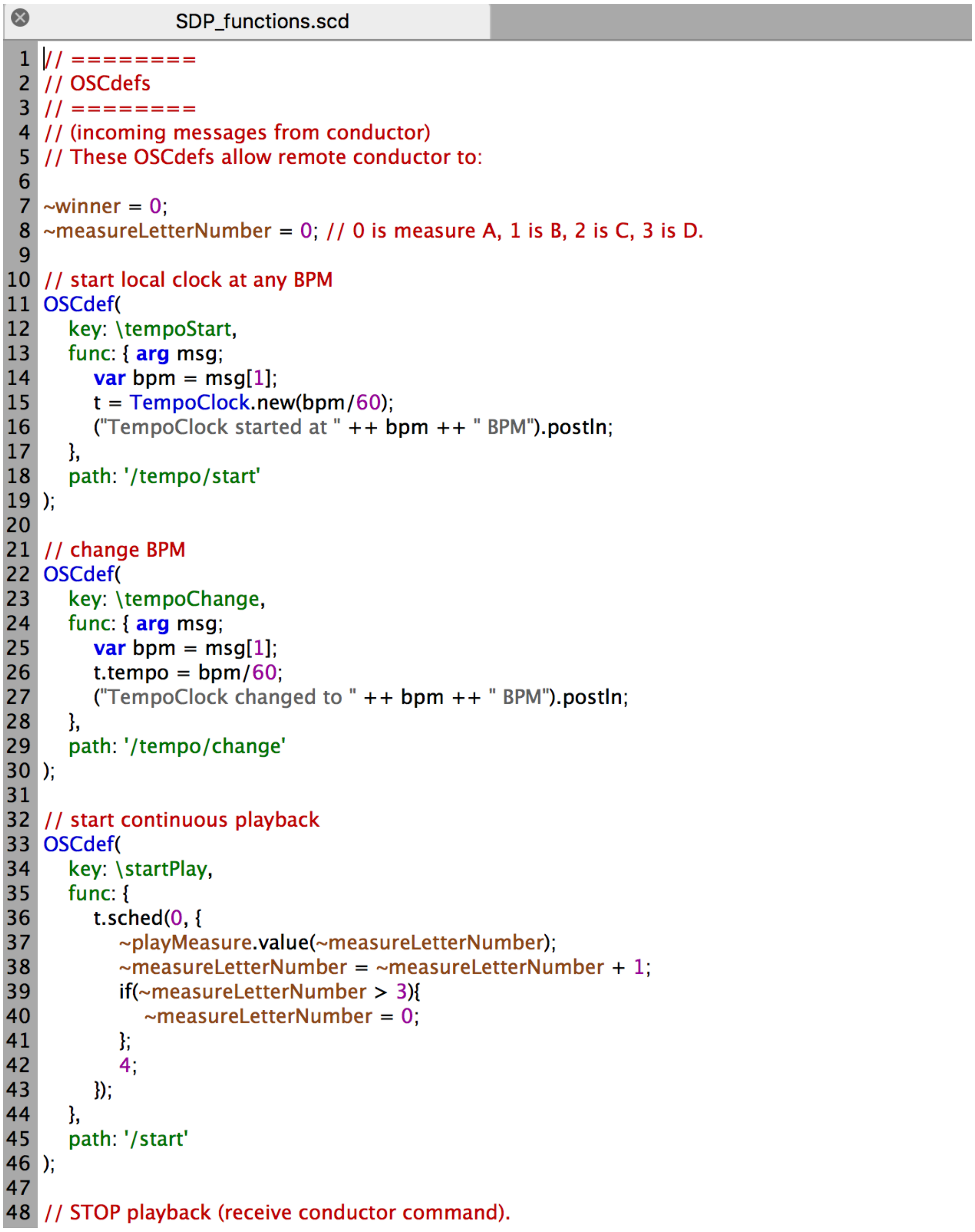
```
// 57120
// NetAddr.langPort;

(
s.waitForBoot({
    "SDP_conductor_functions.scd".loadRelative;
    "SDP_IPs.scd".loadRelative;
})
);


// player stuff
(
"SDP_score.scd".loadRelative;
"SDP_functions.scd".loadRelative;
"SDP_instruments.scd".loadRelative;
);



// run the player file first if you want to play locally

//do this first
~sendTempoStart.value;

//second
~sendStart.value;

//Do this to stop
~sendStop.value;

// GUI for tallying results
"SDP_conductor_GUI.scd".loadRelative;
```

Figure 13: SuperCollider conductor file code



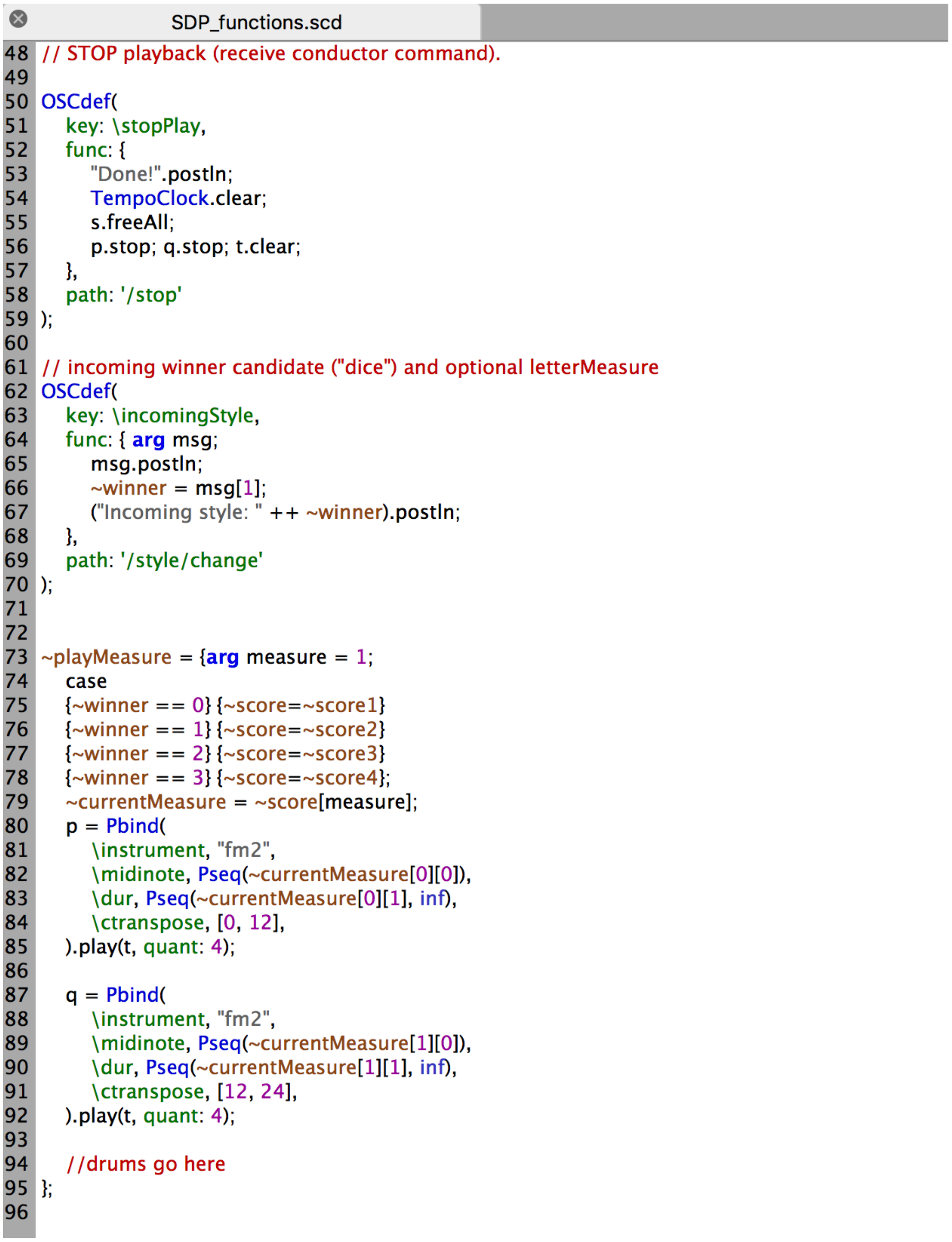
```
SynthDef("fm2", {arg freq = 440, modindex = 10, amp =
0.1, pos = 0, gate = 1, att = 0.01, rel = 0.3;
    var carrier, modulator, freqdev, env, modfreq;
    // i = d/m, so d = m*i
    modindex = Line.kr(modindex, 1, att);
    modfreq = freq / 2;
    freqdev = modfreq * modindex;
    modulator = SinOsc.ar(freq: modfreq, mul: freqdev);
    carrier = SinOsc.ar(freq: freq + modulator);
    env = Env.asr(
        attackTime: att,
        sustainLevel: amp,
        releaseTime: rel
    ).kr(doneAction: 2, gate: gate);
    carrier = Pan2.ar(in: carrier, pos: pos, level: env);
    Out.ar(0, carrier * 0.5);
}).add;
```

Figure 14: SuperCollider instrument file code


SDP_functions.scd

```
// ========
// OSCdefs
// ========
// (incoming messages from conductor)
// These OSCdefs allow remote conductor to:

~winner = 0;
~measureLetterNumber = 0; // 0 is measure A, 1 is B, 2 is C, 3 is D.

// start local clock at any BPM
OSCdef(
    key: \tempoStart,
    func: { arg msg;
        var bpm = msg[1];
        t = TempoClock.new(bpm/60);
        ("TempoClock started at " ++ bpm ++ " BPM").postln;
    },
    path: '/tempo/start'
);

// change BPM
OSCdef(
    key: \tempoChange,
    func: { arg msg;
        var bpm = msg[1];
        t.tempo = bpm/60;
        ("TempoClock changed to " ++ bpm ++ " BPM").postln;
    },
    path: '/tempo/change'
);

// start continuous playback
OSCdef(
    key: \startPlay,
    func: {
        t.sched(0, {
            ~playMeasure.value(~measureLetterNumber);
            ~measureLetterNumber = ~measureLetterNumber + 1;
            if(~measureLetterNumber > 3){
                ~measureLetterNumber = 0;
            };
            4;
        });
    },
    path: '/start'
);

// STOP playback (receive conductor command).
```

Figure 15: SuperCollider function file code part 1



SDP_functions.scd

```
// STOP playback (receive conductor command).

OSCdef(
    key: \stopPlay,
    func: {
        "Done!".postln;
        TempoClock.clear;
        s.freeAll;
        p.stop; q.stop; t.clear;
    },
    path: '/stop'
);

// incoming winner candidate ("dice") and optional letterMeasure
OSCdef(
    key: \incomingStyle,
    func: { arg msg;
        msg.postln;
        ~winner = msg[1];
        ("Incoming style: " ++ ~winner).postln;
    },
    path: '/style/change'
);


~playMeasure = {arg measure = 1;
    case
    {~winner == 0} {~score=~score1}
    {~winner == 1} {~score=~score2}
    {~winner == 2} {~score=~score3}
    {~winner == 3} {~score=~score4};
    ~currentMeasure = ~score[measure];
    p = Pbind(
        \instrument, "fm2",
        \midinote, Pseq(~currentMeasure[0][0]),
        \dur, Pseq(~currentMeasure[0][1], inf),
        \ctranspose, [0, 12],
    ).play(t, quant: 4);

    q = Pbind(
        \instrument, "fm2",
        \midinote, Pseq(~currentMeasure[1][0]),
        \dur, Pseq(~currentMeasure[1][1], inf),
        \ctranspose, [12, 24],
    ).play(t, quant: 4);

    //drums go here
};

```

Figure 16: SuperCollider function file code part 2