Procedural Soundscape

Claire Goeckner-Wald & Amy Xiong

1 Introduction

A short overview of your project idea, why it is interesting and what you would like to learn from it.

1.1 Final Summary

We built a website with a procedural-generated soundscape inspired by 'coffeehouse jazz'. While we did not integrate the soundscape with live variables, we did integrate it with a synesthetic background image. It would not be terribly difficult to integrate the soundscape with live variables at this point, but we chose not to due to time constraints. (Amy: discuss musical 'palette' here.) We chose to use `tonejs-instruments` because it integrated open-source WAV files from real instruments with `Tone.js` [3]. We tested the final product in both Mozilla Firefox Quantum 63.0 and Google Chrome 70.0. Rather than using Google App Engine, or similar, we simply hosted the website for free on Claire's personal GitHub site, cgoecknerwald.github.io/procedural-soundscape. This greatly simplified the workflow, as we could easily push HTML/CS/JS to the GitHub repository, and it would then be immediately updated online with no extra work.

2 Detailed Description and Features

2.1 Description

Precise description of features

2.2 Technical requirements

Audio requirements (and probably also requirements necessary to run the files)

2.3 Technical challenges

2.4 Technological Review

Discuss relevant github repos and projects and articles

Wheelibin's `synaesthesia` (drum kit and synths). We based our chords.js and rhythms.js off of `synaesthesia`. Tambien's `Jazz.computer` but they had a very strange music set-up with interpolation and mediators between synths. We attempted to replicate by playing mulitple synths simulataneously but the resulting 'piano' was very tinny and sounded like Scottish bagpipes. We got mostly drums synths and pads from Yotamm.

2.5 Licensing

We chose to license with the MIT License.

3 Implementation

3.1 Technical Design

Technical design and reasons for choosing this design

We've decided to use tone.js instead of audiosynth.js by Keith William Horwood. We also chose tone.js over SuperCollider, Flocking.js, PureData, because of web integration ease. Flocking also had web integration but a smaller online community and seemed less robust and less abstracted. Audiosynth.js did not have looping technologies. We briefly considered integrating with audiosynth.js but it proved to be too difficult.

We looked at Karplus-Strong String Synthesis but determined it was only relevant if we used audiosynth.js.

3.1.1 Tone.js

> Tone.js is a Web Audio framework for creating interactive music in the browser. The architecture of Tone.js aims to be familiar to both musicians and audio programmers looking to create web-based audio applications. On the high-level, Tone offers common DAW (digital audio workstation) features like a global transport for scheduling events and prebuilt synths and effects. For signal-processing programmers (coming from languages like Max/MSP), Tone provides a wealth of high performance, low latency building blocks and DSP modules to build your own synthesizers, effects, and complex control signals. [10]

3.1.2 Flocking.js

> Flocking is a JavaScript audio synthesis framework designed for artists and musicians who are building creative and experimental Web-based sound projects. It runs in Firefox, Chrome, Safari, Edge, and Node.js on Mac OS X, Windows, Linux, iOS, and Android.
>
> Flocking is different. Its goal is to promote a uniquely community-minded approach to instrument design and composition. In Flocking, unit generators and synths are specified declaratively as JSON, making it easy to save, share, and manipulate your synthesis algorithms. Send your synths via Ajax, save them for later using HTML5 local data storage, or algorithmically produce new instruments on the fly.
>
> Because it's just JSON, every instrument you build using Flocking can be easily modified and extended by others without forcing them to fork or cut and paste your code. This declarative approach will also help make it easier to create new authoring, performance, metaprogramming, and social tools on top of Flocking.
>
> Flocking was inspired by the SuperCollider desktop synthesis environment. If you're familiar with SuperCollider, you'll feel at home with Flocking. [4]

3.1.3 Audiosynth.js

> Dynamic waveform audio synthesizer, written in Javascript. Generate musical notes dynamically and play them in your browser using the HTML5 Audio Element. No static files required. (Besides the source, of course!) [5]

3.1.4 Supercollider

SuperCollider is a platform for audio synthesis and algorithmic composition, used by musicians, artists, and researchers working with sound. It is free and open source software available for Windows, macOS, and Linux. [7]

3.1.5 PureData

Pure Data is an open source visual programming environment that runs on anything from personal computers to embedded devices (ie Raspberry Pi) and smartphones (via libpd, DroidParty (Android), and PdParty (iOS). It is a major branch of the family of patcher programming languages known as Max (Max/FTS, ISPW Max, Max/MSP, etc), originally developed by Miller Puckette at IRCAM.

Pd enables musicians, visual artists, performers, researchers, and developers to create software graphically without writing lines of code. Pd can be used to process and generate sound, video, 2D/3D graphics, and interface sensors, input devices, and MIDI. Pd can easily work over local and remote networks to integrate wearable technology, motor systems, lighting rigs, and other equipment. It is suitable for learning basic multimedia processing and visual programming methods as well as for realizing complex systems for large-scale projects.

Algorithmic functions are represented in Pd by visual boxes called objects placed within a patching window called a canvas. Data flow between objects are achieved through visual connections called patch cords. Each object performs a specific task, which can vary in complexity from very low-level mathematical operations to complicated audio or video functions such as reverberation, FFT transformations, or video decoding. Objects include core Pd vanilla objects, external objects or externals (Pd objects compiled from C or C++), and abstractions (Pd patches loaded as objects). [8]

3.2 File Directory Structure

```
procedural-soundscape
├── assets
│   ├── backgrounds
│   │   ├── bg0.jpg
│   │   ├── bg1.jpg
│   │   ├── ...
│   │   └── bgN.jpg
│   ├── css
│   │   └── main.css
│   ├── js
│   │   ├── chords.js
│   │   ├── constants.json
│   │   ├── instruments.js
│   │   ├── LICENSE-Brosowsky.md
│   │   ├── LICENSE-Mann.md
│   │   ├── LICENSE-Wheeler.md
│   │   ├── main.js
│   │   ├── music.js
│   │   ├── rhythms.js
│   │   ├── Tone.js
│   │   └── Tonejs-Instruments.js
│   └── samples
│       ├── bass-electric
│       ├── clarinet
│       ├── french-horn
│       ├── guitar-nylon
│       ├── organ
│       ├── trombone
│       ├── violin
│       ├── bassoon
│       ├── contrabass
│       ├── guitar-acoustic
│       ├── harmonium
│       ├── piano
│       ├── trumpet
│       ├── xylophone
│       ├── cello
│       ├── flute
│       ├── guitar-electric
│       ├── harp
│       ├── saxophone
│       └── tuba
├── latex
│   ├── report.tex
│   ├── report.pdf
│   └── references.bib
└── index.html
```

```
│
├── README.md
└── LICENSE.md
```

3.3 Implementation Issues

Implementation issues, note any particular technical or audio difficulties and workarounds

We looked into playing and modifying audiofiles (MP3s or WAVs) but the integration with Tone.js seemed too complicated [9]. This threw us off track for several weeks as we searched for good synths to use (we found none). Claire had switched to DuckDuckGo during the project, which has a much less 'intelligent' search algorithm, which would not return the repo "tonejs-instruments" from a search for "tone js instruments" [3]. In `tonejs-instruments`, they use publicly available WAVs from about a dozen instruments, including saxophone and piano (but no drums). Google, however, did, so eventually it was found. We relied heavily on this repo. https://github.com/Tonejs/Tone.js/issues/290 references why we can't use .wav files in Tone.js (simply not supported, it seems).

We discussed moving instruments synths into instruments.js, out of main.js. Moving roots/scales/chord progressions/rhythms/min & max on octaves in chords.js

3.4 Licensing

Licensing a no-licensed repository (wheelibin's synaesthesia) [11]. Choose-a-license, hosted and run by GitHub, was very helpful [6].

4 Analysis & Conclusion

4.1 Original Goals

How close did you come to achieving original goals?

4.2 Regrets

What would you do differently if you knew at the start what you know now.

4.3 Next step

What would be your next steps if you were to continue working on the project.

5 Code

5.1 File

Description:

```
Verbatim code.
```

6 Contact

Claire Goeckner-Wald (claire@caltech.edu) or Amy Xiong (axiong@caltech.edu)

7 Appendix

Include resources here.

7.1 Original Proposal

We aim to build a procedurally-generated soundscape inspired by ‘coffeehouse jazz’. The ideal end-product is jazz-esque music that can respond in live time to environmental variables. We selected jazz as our target because we believe it will be easier to procedurally generate due to the improvisational and diverse nature of jazz. We will initially follow this Procedural Music Generation tutorial. To begin, we select a ‘palette’ of frequencies and sounds that function well together. Then, we will add rhythm and beat to generate our base product. After this is accomplished, we may add additional complexities such as palette-changes, instrument modifications, etc. We may additionally attempt to link these complexities to environmental variables for an interactive soundscape.

We will use SuperCollider for the sound synthesis and algorithmic composition. We will begin with MdaPiano for a piano synthesizer. We will also add drums, saxophone, bass, and other instruments, as appropriate. Ideally we will have large suite of instrument synthesizers. We have considered creating a web application for this project, a la https://asoftmurmur.com/ and similar websites, which allow the client to modify the mixture directly. If we chose to do this, we might use Google App Engine.

7.2 MIT License

```
Copyright (c) <year> <copyright holders>

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

[2]

7.3 The GNU General Public License v3.0

```
<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year>  <name of author>
```

```
This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
```

[1]

References

[1] GNU General Public License.

[2] MIT License.

[3] Nicholaus P. Brosowsky. tonejs-instruments. https://github.com/nbrosowsky/tonejs-instruments, 2018.

[4] Colin Clark. Flocking. https://github.com/colinbdclark/Flocking, 2018.

[5] Keith Horwood. audiosynth. https://github.com/keithwhor/audiosynth, 2014.

[6] Choose A License. No license. https://choosealicense.com/no-permission/, 2018.

[7] James McCartney. Supercollider,. https://supercollider.github.io/, 2018.

[8] Miller Puckette. Pure Data. https://github.com/pure-data/pure-data/, 2018.

[9] Srgy Surkv. Add to tone.js real instruments for music applications. https://github.com/Tonejs/Tone.js/issues/290, 2018.

[10] Tonejs. Tone.js. https://github.com/Tonejs/Tone.js, 2018.

[11] Jon Wheeler. synaesthesia. https://github.com/wheelibin/synaesthesia/commit/5d27951bef5078c72f9c73a65983374879c9c920, 2018.