# Programming Music Camp: Using Web Audio to Teach Creative Coding

Jesse Allison
Louisiana State University
Center for Computation & Technology
jtallison@lsu.edu

Daniel Holmes
Louisiana State University
Center for Computation & Technology
dholm13@lsu.edu

Zachary Berkowitz
Louisiana State University
Center for Computation & Technology
zberko1@lsu.edu

Andrew Pfalz
Louisiana State University
Center for Computation & Technology
apfalz1@lsu.edu

William Conlin
Louisiana State University
Center for Computation & Technology
wconli1@lsu.edu

Nick Hwang
University of Wisconsin-Whitewater
Media Arts and Game Development
hwangn@uww.edu

## ABSTRACT

Programming Music Camp is a summer outreach camp designed to teach computer programming concepts to youths through the activity of music-making. Prior experiences teaching web audio technologies to secondary school students are described. The camp curriculum is then outlined, including the class activities of live coding, instrument design, and concert performance. The outcomes of the camp are evaluated and future educational opportunities using web audio technologies are considered.

## 1. INTRODUCTION

Computer programming and music are complementary in many respects. The act of composition requires information encoding, data structures, and algorithmic processes to generate music. Musical data provides interesting structures to organize, represent, and manipulate in programmatic form. Successful utilization of musical data requires the development of processes that take their acoustic and music theoretical nature into consideration.

Digital audio and audio synthesis techniques can serve as an audible and real-time introduction to computation and mathematics. Musical live coding brings these processes into a dynamic, interactive and fun environment with instant feedback. Given the right framework, computational processes can be immediately heard and evaluated by ear and revised providing for fluid and immediate iterative development.

To take advantage of these interrelations, we created Programming Music Camp[1], a summer workshop for 6-12th graders to teach creative code with web audio[1] technologies. Students learned how to use JavaScript, Gibber[6], Tone.js[3], and Braid[8] within the browser to live-code music, create audio synthesis programs, connect those programs to user interfaces to create expressive instruments, and ultimately perform with these technologies at the conclusion of the camp.

Our goals with the Programming Music camp were many:

- Exposing students to cultural computation by applying computation to the popular medium of music.
- Introducing basic programming concepts through JavaScript, including data types, object oriented programming, frameworks, etc.
- Live coding using Gibber to think about music through programming.
- Exploring basic audio synthesis and computer music techniques using Tone.js.
- Learning user interface design and user experience development processes utilizing Braid.
- Composing, rehearsing, and performing with new technologies.
- Demonstrating that it is possible for anyone to learn these technologies and create fun and exciting music applications.

### 1.1 Prior Experiences

Prior outreaches teaching web audio to middle and high school students had a strong impact on the design and organization of the Programming Music camp.

#### *1.1.1 Lee Magnet High School*

Over the course of several weeks in 2014, web design students at Lee High, a digital media magnet high school, participated in an outreach program to learn web audio concepts and web instrument design using Gibber, Gibberish[7], and NexusUI, an API for web audio interface development[9].

[1]http://lsu-emdm.github.io/Programming-Music/ProMusic2015



*Web Audio Conference WAC-2016, April 4–6, 2016, Atlanta, USA*

The curriculum focused on teaching digital audio concepts in tandem with basic JavaScript programming, with the intention that live coding would benefit students' ability to explore and learn[5]. Students began work using Gibber, but soon transitioned to using Gibber's underlying API, Gibberish, along with NexusUI. Primarily, students would learn to create their own interactive web instruments by the end of the program.

The outreach progressed smoothly, yet some problems with the curriculum emerged. First, Gibber itself was not portable. In order for students to move their audio work from Gibber to their own websites, they had to transition from Gibber's performance syntax to Gibberish, its underlying audio framework. Slight differences in syntax and property naming conventions created high levels of confusion. This turned into a time-consuming step, yet it was necessary in order to explore the potential educational benefits a live coding environment may offer when learning web audio.

A second issue manifested as students began more independent work. Although the students had some experience with HTML5 and CSS3, this outreach was their first exposure to JavaScript. The students did learn basic programming and JavaScript concepts, but they did not have time to establish a solid foundation before the focus shifted to web audio. Although most students had little difficulty picking up basic programming concepts, the abbreviated instruction left some students confused and overwhelmed. Even so, marrying basic programming and computer science concepts with some form of audio or musical instruction showed considerable promise. Fine-tuning a balance between the two and identifying which programming and musical concepts to marry would become important topics while planning the Programming Music camp.

Overall, the outreach at Lee High was a success. Students were eager and willing to share ideas and explore Gibber together, every student was able to incorporate web audio into their websites, and several students designed and created web audio instruments with mobile-ready interfaces. Some of the students continue to pursue general web development, including web audio applications. Ultimately, these early experiences at Lee High directly influenced the curriculum of the Programming Music camp as well as informing updates to both Braid and Gibber.

### *1.1.2 Girls Rock! Summer Camp*

The Girls Rock! summer camp, first held in 2014, was a week long, girls-only summer camp for ages 10-14. The camp introduced a number of topics relevant to sound engineering and electronic music practices, including mobile music performances and score notation, film/video scoring, digital audio editing, and web instrument design and development. This broad spectrum served to increase interest in computer science and engineering via music.

Due to strict time limitations, it was not feasible to teach web audio at Girls Rock! in the same fashion as at Lee High. Rather than focus on gaining in-depth understanding of certain programming concepts, the high-level nature of Gibberish and NexusUI provided the basis for a crash course. The girls chose from various HTML5/CSS3 templates and followed along with step-by-step instructions to turn their templates into individual web instruments. With no time to reinforce the instruction, this short follow-along coding session was not an efficient lesson and led to little retention. Most of the girls did complete web instruments, yet there was a considerable lack of ability to reproduce the work on their own and little variety among the instruments. Regardless, the girls responded positively to the experience once they realized how quick they could create musical web pages. This short introduction helped to demystify programming and web development for many of them.

Overall, the web audio crash course was beneficial and upheld the primary goals of the camp, but future outreaches would benefit from a redesigned curriculum and updated development tools that ultimately provide a more enriching educational experience.

## 1.2 Outline of the camp

The Programming Music camp was structured around learning programming techniques concurrent with their application to music. It continued to be an important pedagogical decision to not directly utilize the web audio framework. Instead, the camp addressed music in the browser through various environments and frameworks built on web audio: live coding syntax in Gibber, UI design within Braid, and web audio synthesis using Tone.js. As this was both an introduction to programming and an introduction to music, a selection of complimentary environments, frameworks and libraries streamlined the syntax and programming topics that needed to be covered and ultimately simplified the development process allowing for more focus on underlying concepts.

The following is a general outline of the scheduled topics, although many of the educational goals and technologies overlapped from day to day:

- **Day 1:** In Gibber the basic syntax of Javascript was introduced alongside music theory.
- **Day 2:** Tone.js was introduced. Gibber was discussed more.
- **Day 3:** Braid was introduced. Tone.js was incorporated within Braid.
- **Day 4:** Work day. Refined Braid instruments
- **Day 5:** Final touches to Braid. Sound checks and rehearsal for performance.

The general instructional format for the camp included lecture presentations, individual work, and small groups. In each case, a team consisting of several highly qualified instructors stood ready to assist individuals with both technical and educational problems. The instructors also rotated among the small groups in order to provide diverse instruction and feedback. Having an adequate team to lead the camp ensured that questions could be answered, code could be debugged, and technical failures could be dealt with without disrupting the camp's tight schedule.

# 2. LIVE CODING

The camp began with a combined introduction to Gibber, programming, and music theory basics, using live coding as an engaging and interactive learning environment.

## 2.1 Music Theory and Basic Programming

Rather than focus on digital audio concepts first (as with Lee Magnet High School), music theory topics better facilitated learning code and music at the same time. For example, learning about and creating melodies, repeating rhythms, and chords in Gibber went hand-in-hand with an introduction to variables, arrays, methods, and basic principles of object oriented programming. Other topics, like notation and musical form, were eschewed for the sake of focus and clarity, and to ensure the lessons aligned closely with both the musical and programming objectives of the camp.

Also, since the students could quickly and easily hear the sounds they were programming, they were better able to be critical of their own work. They began to express a desire to understand music theory in a broader sense: "what sounds good" was discussed in small groups as students prepared for their group performances. It was apparent to students that understanding underlying music theory concepts would help them create musical compositions.

## 2.2 Live Coding with Gibber

Following the introductory music theory and JavaScript exercises, the students dove into more specific Gibber methods and syntax. In this session, the iterative programming exercises emphasized the educational benefits of live coding as students could individually and quickly explore many changes to their code. The session progressed by introducing more advanced Gibber-specific syntax for employing chords, scales, sequences, rhythm, randomization, and effects. During these lectures, relevant code examples accompanied new concepts, and in order to minimize the time young fingers had to spend typing the examples recycled or extended as much code as possible.

Similar to the social, exploratory group sessions at Lee High, live jam sessions followed each topic as it was introduced. Students were able to develop their new musical coding chops incrementally throughout the session. This work-flow, along with the ability to share and explore together, led some students to continue to explore Gibber's built-in documentation, tutorials, and examples during their breaks.

Creative music making continued to be the focus of each technical exercise. In this regards, regular interjections of special topics and events helped to stimulate and focus the students. The highlight of the entire camp was a video call with the creator of Gibber, Charlie Roberts. The students asked several interesting questions, experienced a short live coding demonstration, and had the opportunity to try out the newly implemented group synchronization system, Gabber.

## 2.3 Ubiquitous Code

Throughout this unit, a modified Gibber syntax allowed the code the students wrote in Gibber to be maximally similar to the code they would later write with Tone.js. Specifically, the Gibber code examples were made to be more verbose than is strictly necessary for the sake of uniformity with standard JavaScript. In the following example, each pair of lines are interchangeable in Gibber. In each case, the first line is the verbose syntax we used, and the second is the typical Gibber syntax:

```
self.post("Hello, World!");
```

**Figure 1: Designing User Interfaces using Braid and Tone.js**

```
post("Hello, World!")

a.note.seq([‘d4’,‘eb4’], [0.5, 1/2]);
a.play([1,2], 0.5)

a.chord.seq([[0,2,4], [1,3,5]], [4/4]);
a.chord.seq([‘c4m’, ‘d4dim’], 1)
```

Using various methods to indicate duration, pitches, and chords encouraged discussion of data types. These small syntax choices helped us smoothly transition into designing standalone web audio instruments using Tone.js and Braid.

# 3. DESIGNING WEB INSTRUMENTS

## 3.1 Tone.js

Using Yotam Mann's Tone.js web audio synthesis framework [3], students created musical instruments and uploaded them on a web server. Some instruments programmatically performed a composition, while others had user interfaces to allow live performance. Students learned the differences in musical performability between live coding in Gibber and coding in Tone.js. While Tone.js is not intended solely for instrument creation, the instructors planned to involve user interfaces in later lessons. Having learned aspects of live coding in Gibber, Tone.js showed students aspects of interpreted computing. Since controlling many musical parameters of a single instrument would be difficult in live performance, students often treated the synthesis method of their instrument with more permanance and sought to control pitch, timbre, and loudness in performance. As such, they spent more time experimenting and ultimately selecting a specific type of synthesizer in Tone.js.

Teaching Tone.js allowed for the instruction of new key programming and musical concepts as well as reinforcing the concepts already introduced: the basics of object oriented programming through JavaScript, the use of external text editors, basic web design, creation of musical instruments, chord structures, musical timing, controlling envelopes, and basic synthesizers.

In the context of music-making, students gained understanding of general coding practices and experience typing out code, while being introduced to object oriented program-

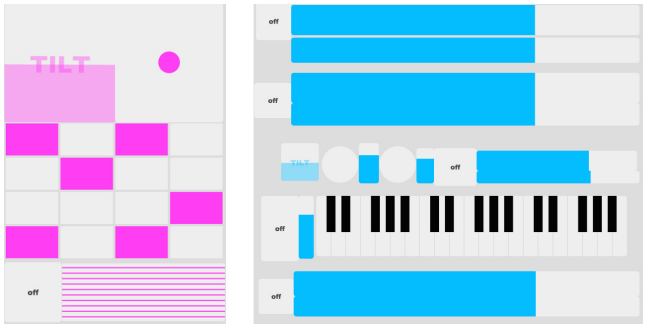


**Figure 2: Two tablet instruments made by camp students using Braid and Tone.js**

ming. They learned how to use arrays and loops by building chord patterns, melodies, and musical durations which they were able to cycle through. The concepts of basic synthesis, class parameters, and methods were taught along with lessons on Attack-Decay-Sustain-Release enveloping and how changing oscillator classes affect the tone of an instrument. Students were able to chain audio signals with effects (like reverb and distortion) while debugging their code through console logs. Then, by incorporating UI elements through NexusUI and Braid, students were able to control their instruments in live performance.

## 3.2 Braid

After being introduced to Tone.js and basic synthesis practices, students built interactive graphical interfaces using Tone.js and Braid (Browser Audio Interface Database)[2]. In Braid, students created HTML5 audio interfaces using drag-and-drop, and could write Tone.js code directly in the browser. These instruments were then saved to an online database and loaded onto each student's mobile device for performance. Since all work was done within a browser window (much like in Gibber), students could quickly revise their instruments and test different ideas.

### *3.2.1 Updating Braid*

Braid, developed by the authors, received a substantial update preceding this camp. Most significantly, the integration of Tone.js provided better support on mobile devices. Also, a newly developed responsive search screen provided easier access to instruments from mobile devices. However, using Tone required some structural changes. Previously, a "locking" mechanism was used to toggle between editing and playing modes. Now, an instrument is launched in its own browser window, so that all web audio code stays local to that window and will be disposed when the window is closed.

## 3.3 Synthesis

Teaching the fundamentals of acoustics and sound synthesis is important in any music programming curriculum. Not only is it a crucial aspect of modern digitally-produced music, but it is an excellent way to foster interdisciplinary learning among the students by directly linking music with math and science. The unit on synthesis occurred during a single morning session, with a focus on application. The

[2]http://braid.nexusosc.com/tone/

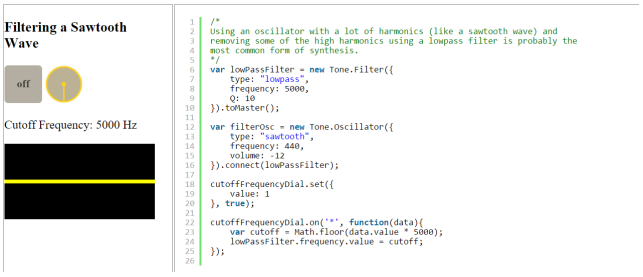


**Figure 3: Example audio synthesis code presented to students**

goals of the session were:

- Demonstrate a few basic principles of acoustics such as sound wave properties (frequency and amplitude) and the harmonic series.
- Construct a basic signal flow for subtractive synthesis.
- Build simple web audio instruments with Tone.js and Braid in order to become more fluent with fundamental ideas in coding.

The instruction for this unit featured a mixture of short lecture and short activity, and the students had access to a web page[3] with examples of basic acoustic properties and synthesis methods, including code samples. The page itself served as a technical resource: the physics examples are all also NexusUI and Tone.js examples. During the prescribed morning session, the students were able to complete the first three of the six examples: 1) a sine wave with an on/off toggle and sliders for frequency and volume, 2) adding partials of a sawtooth wave, and 3) a sawtooth wave with a resonant lowpass filter and a knob for controlling the cutoff frequency.

A short presentation describing basic, underlying scientific principles introduced each example. Then, students attempted to individually recreate the example using Braid and Tone.js. Some solutions required simple modifications to the code examples they already had, but some required an understanding of the basic coding principles covered earlier in the camp such as variables, functions, and conditional statements. For example, when students recreated the interactive example and code for a sawtooth oscillator connected to a lowpass filter (Figure 3) in Braid, it was necessary for them to adapt the code (rather than copy and paste exactly). This facilitated logical and analytical thinking as opposed to simply following exact step-by-step instructions.

After completing the prescribed exercises, the students began experimenting with Braid on their own. The sounds and interfaces created during this session often became the basis for the final form of the Braid instruments that each student developed on his or her own.

## 3.4 Outcomes

Through Braid's user interface, combined with underlying audio synthesis code written in Tone.js, students created musical instruments able to be performed on tablet devices. Many of the common interfaces used piano keyboard-style

[3]http://lsu-emdm.github.io/Programming-Music/SoundBasics/

pitch selection with accompanying sliders to control effects such as distortion and envelope shaping. A few students involved the tilt sensors and 2-dimensional touch sliders which allowed for physically gestural musicality. Many students lavished the opportunity to describe their instruments before demonstrating its capabilities in their performances. The morning prior to the performances, students shared their instruments with other fellow students to garner feedback on their interface layouts and instrument design. This feedback encouraged students to make various adjustments to their instruments, and it helped them to understand the benefits of iterative process.

# 4. REHEARSAL AND PERFORMANCE

At the beginning of the week, students divided into small groups of 3-5 members which served as their permanent small group for the entire week. After being assigned a primary coach, each group prepared a performance for the end of camp. The last hour of each day consisted of supervised and open ended rehearsals.

To introduce the students to rehearsing together, the instructors strongly encouraged that listening be the most basic skill the students practiced for live coding. The first task for most groups was executing code synchronously which required they listen carefully to prevent starting off-beat from one another.

As they became more comfortable, some groups focused on creating integrated content and patterns that fit together in clear and compelling ways. Other groups gravitated towards sounds that clashed with one another or transformed rhythms into textures that contrasted or overwhelmed other students music. Detailed discussions on concepts like contrast, balance, and blend proved to be extremely helpful. Students were surprisingly receptive to being asked to contextualize different sonic palates into a collaborative performance that could respect the choices of their peers.

# 5. CONCLUSIONS AND FUTURE DIRECTIONS

Teaching creative coding through web audio allowed for a streamlined pedagogical approach that enabled our students to learn the basics of programming in JavaScript, sound synthesis, music theory and live coding within one week. Even though the students came in with diverse backgrounds, some with little or no programming or musical experience, all were able to complete their instruments and felt confident enough to stage live coding performances and perform successfully in front of an appreciative audience. Although there are many places for improvement, considering the ambitious goals outlined for the camp, we have found this approach performed admirably.

In dealing with student attention spans, work sessions were generally more effective with open-ended creative tasks than when given more specific step-by-step instructions. Maintaining focus was at times difficult during sections involving tedious tasks or typing out large portions of code. However, students found that their attention to detail prevented errors both in live coding and instrument making. Examples with recognizable materials, like popular tunes, had greater student engagement. Although the number of instructors allowed for more individual attention to students, there were noticeable times students appeared off-task when they had to wait for help.

Many students expressed interest in continuing musical programming after the camp and understood the free availability of Gibber, Tone.js, and Braid on the web. While not all students at the start of camp had musical background, many stated they were interested in understanding music in general at a deeper level. Students and those attending the performance seemed surprised and inspired by the musical possibilities through programming.

Possible future directions include exposing students to additional applications of programming for music, as the examples by Yotam Mann and Andrew Sorensen, and visits with Charlie Roberts and Edgar Berdahl appeared to inspire the students greatly.

Examples of similar coding in other languages or frameworks like Game Maker Studio, Supercollider, or Max/MSP could be used to open their horizons on other approaches to programming in music and encourage students to pursue computation further after the camp.

More peer learning and feedback among students would be helpful to build camaraderie and increase the collaborative experiences gained in traditional musical ensembles.

Finally, incorporating more activities that demonstrate musical or programming concepts may help inspire music creation and provide welcome breaks throughout the day.

# 6. ADDITIONAL AUTHORS

Additional authors: Benjamin Taylor, email: `btayl61@lsu.edu`

# 7. REFERENCES

[1] P. Adenot, C. Wilson, and C. Rogers. Web audio api, http://webaudio.github.io/web-audio-api/.

[2] I. Bukvic. Granular learning objects for instrument design and collaborative performance in k-12 education. In *Proceedings of International Conference on New Interfaces for Musical Expression*, 2011.

[3] Y. Mann. Interactive music with tone.js. In *Proceedings of the 1st annual Web Audio Conference*, 2015.

[4] S. McCoid, J. Freeman, B. Magerko, C. Michaud, T. Jenkins, T. McKlin, and H. Kan. Earsketch: An integrated approach to teaching introductory computer music. *Organized Sound*, 18, 2013.

[5] C. Roberts, J. Allison, B. Taylor, D. Holmes, M. Wright, and J. Kuchera-Morin. Educational design of live coding environments for the browser. *Journal of Music Technology in Education*, 2016.

[6] C. Roberts and J. Kuchera-Morin. Gibber: Live coding audio in the browser. In *Proceedings of the 2012 International Computer Music Conference*, 2012.

[7] C. Roberts, G. Wakefield, and M. Wright. The web browser as synthesizer and interface. In *Proceedings of the New Interfaces for Musical Expression conference*, 2013.

[8] B. Taylor and J. Allison. Braid: A web audio instrument builder with embedded code blocks. In *Proceedings of the 1st international Web Audio Conference*, 2015.

[9] B. Taylor, J. Allison, Y. Oh, D. Holmes, and W. Conlin. Simplified expressive mobile development with nexusui, nexusup, and nexusdrop. In *Proceedings of the New Interfaces for Musical Expression conference*, 2014.