Tangible Distributed Computer Music for Youth

R. Benjamin Shapiro,* Annie Kelly,* Matthew Ahrens,† Ben Johnson, Heather Politi,†† and Rebecca Fiebrink§**

*ATLAS Institute
University of Colorado, Boulder
1125 18th Street 320 UCB, Boulder,
Colorado 80302, USA
{ben.shapiro, annie.kelly}@colorado.edu
†Department of Computer Science
Tufts University
161 College Avenue, Medford,
Massachusetts 02155, USA
matthew.ahrens@tufts.edu
**Manhattan School of Arts and
Academics
290 Manhattan Drive, Boulder, Colorado
80303, USA
ben.johnson@bvsd.org
††Nederland Middle-Senior High School
597 County Road 130, Nederland,
Colorado 80466, USA
heather.politi@bvsd.org
§Department of Computing
Goldsmiths, University of London
New Cross, London SE14 6NW, UK
r.fiebrink@gold.ac.uk
http://atlas.colorado.edu/lpc/blockyTalky

Abstract: Computer music research realizes a vision of performance by means of computational expression, linking body and space to sound and imagery through eclectic forms of sensing and interaction. This vision could dramatically influence computer science education, simultaneously modernizing the field and drawing in diverse new participants. In this article, we describe our work creating an interactive computer music toolkit for youth called BlockyTalky. This toolkit enables users to create networks of sensing devices and synthesizers, and to program the musical and interactive behaviors of these devices. We also describe our work with two middle-school teachers to codesign and deploy a curriculum for 11- to 13-year-old students. We draw on work with these students to evidence how computer music can support learning about computer science concepts and change students' perceptions of computing. We conclude by outlining some remaining questions around how computer music and computer science may best be linked to provide transformative educational experiences.

The creation of new digital musical instruments exemplifies how computational practices can be simultaneously creative, expressive, and technologically rich. Much research and creative work in computer music investigates new ways for computational systems to enable musical expression and interaction. Reciprocally, the task of creating interactive systems for performing computer music functions as a sort of "extreme human–computer interaction," pushing the limits of existing computing systems and demonstrating possibilities for technological advancement (Tubb 2016). Despite this boundary-pushing relationship between computer science (CS) and music, the innovations of the computer music community have not yet become a widespread part of CS education.

Computer Music Journal, 41:2, pp. 52–68, Summer 2017
doi:10.1162/COMJ_a_00420

We believe that computing education could benefit from integrating topics central to computer music performance into CS curricula, and from adopting the computer music community's understanding of computing as an expressive, creative domain.

First, computer music is a uniquely strong application domain through which we could modernize CS curricula. Computing-education research scholars, as well as industry, have bemoaned the absence of contemporary computing topics like distributed systems and concurrency from CS education (Patterson 2006). Recent standards documents like the "Computer Science Curricula 2013" (ACM/IEEE-CS 2013) mandate the inclusion of these topics within CS education pathways, but compelling examples of how to do so are currently lacking. In contrast, many computer music performance systems rely on timing-sensitive distributed computing, concurrency, and networking.

Computer music also provides an opportunity to combat pervasive and insidious misperceptions of computer science and computer scientists. American students and adults have negative stereotypes of science and scientists, often believing that they are socially distant, dangerous, workaholic, peculiar, irreligious, and missing fun in their lives (Mason, Kahle, and Gardner 1991; Losh 2010). They tend to hold similar stereotypes of computer scientists (Martin 2004; Carter 2006; Grover, Pea, and Cooper 2014). These stereotypes can be challenged by educational approaches that bring scientific methods together with topics that are relevant to people's lives and that showcase the wide range of possible prosocial impacts of science, technology, engineering, and mathematics (STEM), including computing (Grover, Pea, and Cooper 2014). Given that music is often both highly social and creative, CS education experiences that draw on computer music could drastically change students' perceptions of computing.

Integrating music into computer science education may likewise be helpful in broadening participation in CS by women, ethnic or cultural minorities, and people of low socioeconomic status, all of whom are underrepresented in CS. Integrating computer music into computing curricula could improve participation both by combating the misperceptions described earlier and by creating more pathways into computer science—as well as computer music itself—for students who are already musicians.

In this article, we describe the creation of a distributed and physical computer music systems-building and performance toolkit for adolescents aged ten years and older. We have designed this toolkit to enable youth to create new digital musical instruments and other interactive music systems, with the aims of engaging them with a variety of computing concepts, as well as challenging them to think more positively about the creative potential of computers and their own ability to make use of computing. We then describe our recent work with two middle-school teachers—one a math teacher, the other a music teacher—to codesign and deploy a curriculum for 11- to 13-year-old students. Our work with these students provides evidence of ways in which computer music can support learning about CS concepts and change students' perceptions of computing. We conclude by outlining some remaining challenges and questions around how computer music and computer science might best be linked to provide transformative educational experiences.

Design Considerations

We have constructed a toolkit for youth to create computer music systems that they can use in collaborative music performance. Our tool, called BlockyTalky, enables users to construct a variety of physical, sensor-rich interfaces, and it supports the combination of different student-made technologies in distributed systems (e.g., sensors communicating wirelessly with software sequencers and synthesizers).

Systems that integrate real-time sensing, programming logic, sound making, and networked communication are quite common in computer music, yet existing programming tools designed for young people do not target this type of design. For example, Scratch is a popular software system for youth to use to create games, stories, and animations. Scratch enables users to write

programs by dragging pieces of code onto sprites to define their behaviors. Scratch allows users to play sounds and even to build modest physical interfaces for triggering sound (via Makey Makey; www.makeymakey.com/). The sound programming features are more suited to adding sound to games than they are to composing and performing music, however. EarSketch (Freeman et al. 2014) is another educational programming environment that combines music with CS education. It enables users to programmatically sequence and manipulate synthesized sounds and samples. It is not designed for collaborative real-time performances or for using physical inputs to shape sound synthesis, however.

We wished to replicate certain successful aspects of tools like Scratch and EarSketch; for instance, their browser-based interfaces drastically reduce the complication of system use in schools. We also recognized, however, that we would need to borrow heavily from design and engineering patterns that are common in the design of new digital musical instruments such as those featured at the International Conference on New Interfaces for Musical Expression (NIME) or the Guthman Musical Instrument Design Competition. Such instruments can often be understood as assemblages of the following types of components: sound-synthesis methods that offer real-time control over their parameters; sensing hardware that obtains real-time information about performers' physical actions; software that controls the parameters of sound synthesis in response to sensor data or algorithmic processes; and networking components that support distributed sensing and sound making. The last type of component may possibly also support communication and synchronization between multiple performers. As engineers, we wished to create the first toolkit that enabled young learners (aged 10 and older) to create these types of components and connect them to make new systems for live music performance.

Furthermore, as educators, we wanted to empower users of this toolkit to learn about relevant computational concepts. For instance, BlockyTalky requires students to design their own high-level networking protocols—messages passed between different devices for synchronization and control. Students often must discover and adapt to constraints such as latency in sensing, communication, and synthesis. This is essential for students to learn as they design instruments that allow them to improvise during performances using physical inputs or modification of code. Furthermore, building instruments for live performance can lead students to discover how different configurations of sensors support different types of physical interaction.

Implementation

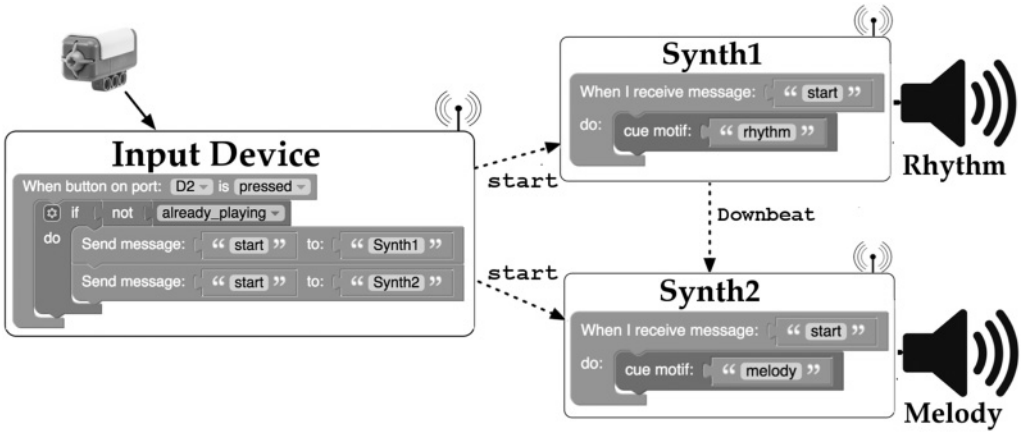
Over the past three years we have iteratively refined the BlockyTalky toolkit to meet the engineering and educational goals noted in the previous section. BlockyTalky is open-source and runs on low-cost single-board computers like the Raspberry Pi. Typically, we equip these boards with "shields" that allow the use of modular sensors and actuators, including child-friendly LEGO Mindstorms and the slightly more complex Seeed Studio Grove components.

Each BlockyTalky device runs a server that provides users with a complete Web interface for configuring and programming its hardware. Drop-down menus enable users to declare what kinds of sensors are connected to each input port, and the system provides real-time sensor readings to help users plan, monitor, and troubleshoot their designs.

A variety of programming blocks enable users to define musical motifs (including synthesized notes and samples), to send messages between devices, and to create event handlers for inputs received from physical sensors or for messages received over the network. The programming model assumes that users will typically compose and enact performances using several physical devices at once, with some devices handling sound synthesis and others handling physical inputs from users; user-created asynchronous messaging protocols coordinate activity across these devices. Figure 1 shows a typical configuration along with code for that configuration.

Users can configure BlockyTalky synthesizers to synchronize their clocks to one another by choosing one synthesizer to serve as a reference clock and

Figure 1. Typical project architecture and code. An input device with a single button transmits messages over WiFi to two synthesizers. The synthesizer Synth 2 subscribes to clock information from Synth 1 so that they can be synchronized.



programming the other synthesizers to subscribe to that clock (using a Sync to block). Then they can use Wait for blocks to specify the timing of note synthesis or sample playback. For instance, multiple synthesizers can synchronize their actions by using Wait for blocks triggered by the same event, such as the next downbeat.

The block-based programming interface is implemented using Google's Blockly toolkit (Google 2016). Users' block programs are "transpiled" into a textual domain-specific language. The domain-specific language provides convenient abstractions around common complexities for physical computing and network programming. For example, the block program in Figure 2a is transpiled to the code in Figure 2b.

The when_sensor macro manages hardware state information that is necessary to detect and dispatch events, and the send_message function encapsulates peer discovery, serialization, and transmission of messages over User Datagram Protocol or Transmission Control Protocol.

This functionality is implemented in JavaScript and in the Elixir functional programming language, which runs on the Erlang virtual machine. Erlang's actor-based architecture enables us to quickly add new capabilities to the system, such as support for new hardware, networking protocols, and user interfaces. The Phoenix Web framework (which serves both static Web content and streaming real-

Figure 2. A simple block program in BlockyTalky (a) and the text code resulting from "transpilation" into the domain-specific language used for implementing user programs (b).

(a)

```
when_sensor "PORT_1" == 1 do
    send_message("tickle", "elmo")
end
```

(b)

Figure 2

time communication between BlockyTalky devices and the browser over WebSockets) enables us to provide live information to users about device and sensor states and network communications.

Results with Youth and Their Teachers

We have used BlockyTalky with youth in a variety of different educational settings, including two month-long computer music units in middle school classrooms, two multiweek computer music summer camps (Shapiro et al. 2016), many short 1- to 2-hour workshops, and in 5- to 10-minute interactions during outreach events. Here, we present results from the two most recent iterations of our

approach, which were enacted in two middle school classes in the Rocky Mountain region of the United States. All data presented in this article were collected under the supervision of our university's institutional review board.

Codesigning Classroom Implementations

We collaborated with two middle-school teachers to design four- to six-week computer music units that they would deploy—with our support—in their classrooms. In the United States, middle-school students are typically 11–13 years old. One of these teachers, Benjamin Johnson, is a music teacher with 18 years of teaching experience, and Heather Politi is primarily a math teacher, although she also teaches courses on computing and design, using technologies such as Scratch, Arduino, and 3-D printing. She has taught for nine years. We were introduced to Johnson by another researcher, who had previously collaborated with him to integrate electronics design into his music composition class. We met Politi at an outreach event focused on connecting teachers to CS Education resources.

Our work with students in Politi's and Johnson's classrooms began in March 2016. Our codesign work with Politi and Johnson began months earlier, however, when we began meeting approximately weekly (cf. Roschelle and Penuel 2006 for background on codesign in educational settings). We used the meetings to play with BlockyTalky, to brainstorm possibilities for classroom learning, to draw connections between each teacher's pre-existing goals for student learning and approaches to teaching, to create rough lesson plans for using BlockyTalky within their classrooms, and even to modify aspects of the BlockyTalky programming language to better match Johnson's music teaching methods. Once the classroom implementations began, we continued to meet regularly to discuss how things were proceeding, and to make—or adjust—plans for subsequent days of teaching.

Politi and Johnson had distinct but overlapping goals for using BlockyTalky in their classrooms. Johnson planned to use BlockyTalky in his course on music theory and composition, a course in which students already had experience composing and performing music of their own creation using acoustic instruments. He hoped that computer music would offer students a new context to apply and further develop their composition skills, as well as push them to strengthen their teamwork skills. Students in his class had previously used Finale NotePad to support their music composition, and had even built their own acoustic musical instruments, but had not previously worked together to compose and enact joint performances.

In contrast, Politi had never taught, or even played, music before our work together, so some of our codesign sessions involved introducing her to basic musical concepts and vocabulary (e.g., the diatonic scale; how pitch can be described in terms of note or frequency; and relationships between melody, harmony, and rhythm). She was eager to participate because she believed that creating interactive musical devices could offer a challenging and interesting design domain for her students, and that it would offer them an opportunity to further develop the programming skills they had developed with Scratch. As such, whereas Johnson was primarily excited about how learning computer music with BlockyTalky could enhance students' learning of music, Politi was primarily motivated by expanding students' "computer power."

Sharing Knowledge: Algorithmic Composition for BlockyTalky

We drew upon Johnson's knowledge of teaching music theory to address a challenge that had arisen in our prior work teaching nonmusician youth to create interactive computer music systems. Many students who participated in our previous workshops and camps have been excited to re-create pop songs that they are familiar with (Shapiro et al. 2016). We have found this to be a double-edged sword: On the one hand, it is exciting to many students to be able to re-create music that they already enjoy, to make something that sounds good even without formal knowledge about music composition. On the other hand, we have frequently found this to be a time-consuming dead end for students, and we have come to call it the "Pop

Figure 3. Two different melodies that can be composed using Johnson's Recipes (see text for examples of these rules). Melodies (a) and (b) both start and end on a "1" note (tonic), but have distinct rhythms and sequences of notes in between.

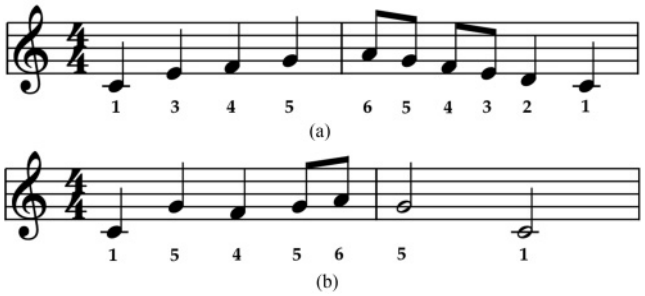
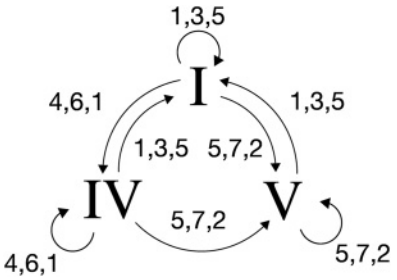


Figure 4. Diagram of a nondeterministic finite automaton used to help composers harmonize chords following classical Western rules. Each state corresponds to a particular chord.



Music Re-creation Trap." Students who succumb to this trap tend to become intensely focused on one-to-one replication of pop music, and do not explore rearrangement or improvisation around melodies and rhythms in their chosen songs. Their work focuses more on fidelity of re-creation rather than creative expression. Because re-creations are necessarily linear and noninteractive, students caught in this approach tend to write programs that consist of one or two very long, nonbranching procedures. They tend not to explore programmatic methods for flow control, event handling, and synchronization of musical motifs that would be useful in improvisation, thereby limiting their CS learning.

Because Politi's students were mostly nonmusicians, we were concerned about encountering the same computationally weak Pop Music Re-creation Trap as we had encountered with other nonmusician participants. Fortunately, in his prior work teaching music theory, Johnson had developed an algorithmic approach to composition that he calls "Johnson's Recipes." These simple rules govern the construction and integration of melody, harmony, and rhythm, and they can be presented in a way that is accessible to middle-school students. The rules are articulated in terms of scale degrees (in the classroom, these were often referred to as "finger numbers") so that they can be applied to composition in any scale (e.g., in C Major, the number 2 corresponds to the pitch D). The rules for melody composition are: (1) start and end on "1," (2) use steps frequently, and (3) use skips sparingly. Figure 3 demonstrates two examples of melodies that can be composed using these rules.

After they compose melodies, students can complement them with harmonies. Johnson's Recipes initially specified the rules for harmony composition as follows:

1. Use the I, IV, and V chords from the key of the melody. The chord is named from the root note of the chord. For instance, I is 1-3-5, IV is 4-6-1, V is 5-7-2.
2. For each note in the melody, choose a chord that contains that note.
3. Do not use a IV chord immediately following a V chord. It may be necessary to backtrack, changing the chord on a previous note in order to meet this constraint.

Upon seeing this, computer scientists on our research team were excited to realize that musical rules could be represented as state machines; in particular, nondeterministic finite automata (NFAs) offer a simple way to do so. Therefore, we rewrote Johnson's Recipes as shown in Figure 4.

This representational shift offered a way to depict musical conventions with a representation that is computationally powerful. The NFA diagram in Figure 4 also affords easy representation of the different "rule sets" associated with different musical genres (e.g., adding an additional edge from V to IV in the state machine in Figure 4 transforms it to allow for blues-style harmonization). Indeed, the power and flexibility of NFAs and other state machines have often been exploited by algorithmic composition systems in the computer music community (e.g., Choi and Wang 2010). In an educational context, an NFA diagram offers the further benefit of affording an easy way to check student work for conformance with a specific genre. This

Table 1. Two Class Plans Using BlockyTalky

Week	*Johnson's Music Class*	*Politi's Computer Power Class*
1	Introduction to BlockyTalky	Introduction to BlockyTalky
2	Small projects, show and tell	"Throwaway Project" Teach melody and harmony
3	Plan and play	Building final projects
4	Build	Final presentations and display
5	Finalizing the projects	
6	Performances	

property of the diagram, of being useful as a tool for being able to generate harmonies as well as to verify their acceptability within a given genre of music, is emblematic of how such diagrams are generally used in CS theory courses and, as such, enables a deeply musical activity (songwriting) to also be a computationally rich one.

Our codesign work with Johnson and Politi made extensive use of Johnson's Recipes. We used them to write songs together, to construct and program networks of BlockyTalky devices to play those songs, and then to design lesson plans to teach students to do the same.

Unit Organization

Owing to their differing goals and the different prior knowledge of their students, Politi and Johnson opted to work together with us to create two different sets of plans for what to do in their classrooms, including both what the products of student work would be and what the process of creating those products would be (see Table 1).

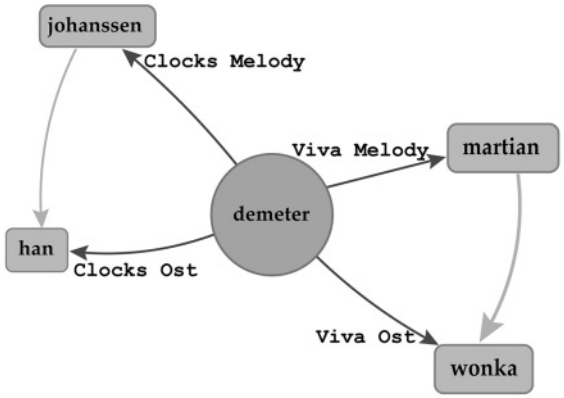
We also planned to sequence these classroom implementations such that Johnson's class would go first, and Politi's afterward. This allowed us to manage equipment resources (we did not have enough sensors or synthesizers for both classes to work simultaneously) as well as to debug any issues that arose in combining BlockyTalky with Johnson's Recipes in a schooltime classroom setting (all of our previous work with youth had taken place in summer workshops or other informal educational contexts).

Student Work

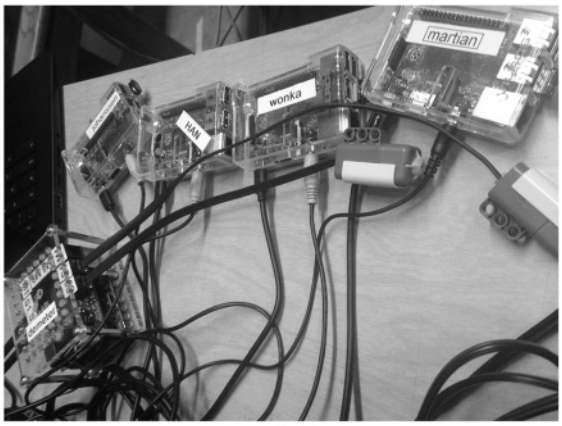
All consenting students were video- and audio-recorded as they worked together in groups. We also retained time-coded snapshots of all code that students saved on their BlockyTalky devices. In both Politi's 24-student class and Johnson's 21-student class, the students displayed their projects at the end of the workshops. Students created a number of projects. These projects involved students designing, building, and programming a variety of hardware and software structures. The projects sampled in this article describe a sample of student projects from both classes (a total of nine projects in Politi's class and five in Johnson's) and shows their final system architectures and physical construction. The architectures differed strikingly, depending upon the kinds of functionality and eventual user interaction the students hoped to support.

For example, the Coldplay Mashup system (see Figure 5) supported a group performance in which all of the performers manipulated sensors connected to a single device, Demeter, which in turn dispatched messages to the various synthesizers in the system. In contrast, the Rubik's Cube Competition Accompaniment System (see Figure 6) had a strikingly parallel network structure, one in which each Rubik's competitor interacted with a distinct two-node system consisting of a sensor (e.g., Mystique) and a synthesizer (e.g., Transparent). Prior work suggests that this intraclass variation in solution structure is associated with students' attention to peers' creative ideas (Deitrick, O'Connell, and Shapiro 2014), and greater success at problem-solving (Barron 2003).

Figure 5. Student project "Coldplay Mashup," a mashup of the songs "Clocks" and "Viva la Vida" by the band Coldplay. The students working on this project used buttons to trigger particular parts of the song as they performed. In the network diagram (a), circular nodes are devices with sensors, nodes displayed as rounded rectangles are synthesizers, dark edges are messages constructed by the students, and the lighter edges are clock subscriptions. The devices in the physical construction (b) are labeled, indicating the mapping to nodes in the diagram.



(a)



(b)

Their projects differed in their visual aesthetics. For instance, the Coldplay Mashup project (see Figure 5) was a spartan assembly of computing hardware, unadorned by decoration. This was consistent with the group's plans for them to use it within a performance of their own, in contrast to other groups who created projects for others to use, such as Catspresso and Fireball (see Figures 7 and 8).

These examples illustrate the kinds of projects that students created, as well as the complexity of the distributed systems that students designed and implemented to create them. Our qualitative data, consisting of audio and video recordings of students' working conversations with each other, their teachers, and our research team, offer us insight into the processes through which students created these systems. Specifically, they enable us to observe the processes of composition, design, programming, and problem-solving that produced these projects. We now present and interpret a pair of such conversations. These conversations were typical of students' conversations in both classes, and as such are representative of the kind of computational problem-solving that the BlockyTalky experience engendered.

Example 1: Reasoning about Message Passing

The following conversation took place between two students, Carter and Anthony, early in the work of the Catspresso group. They are trying to figure out why one of their motifs does not play after they press a button.

> **Anthony:** Does it even play?
>
> **Carter:** When port 2 is pressed, send message "ostinato" . . . now that should work, but nothing is playing. Is this speaker on? Yeah. That's the thing we've been struggling with. We've never been able to play the ostinato from Ned and the melody from Beeps at the same time.

Here, Carter is explaining the problem to his groupmate: When they push the button connected to WallE, they want to have music play on two different synthesizers—Ned and Beeps—but it is only playing on Beeps. After the boys struggle with this for a while, a research team member, Kelly (AK), comes over to help. The boys explain the situation to her, and she proceeds to help them debug their code. They start to discuss whether the synthesizers are synchronized with one another:

> **Carter:** . . . so on Beeps we have "sync Beeps to Ned." And then—we don't have any syncing stuff on this [Ned] because we only need one I think.
>
> **AK:** Yeah you're right. Can I see what happens when you press both the buttons? Does anything play at all?

Figure 6. Student project "Rubik's Cube Competition Accompaniment." Three students try to solve their Rubik's cubes while music plays. The tempo increases the longer they take to solve their cubes. The boxes in the physical implementation (b) had distance sensors attached, which the students used to trigger the start and end of the song.

Figure 7. Student project "Catspresso." A 4 × 5 grid of buttons that could trigger different combinations of melodies and instruments. The cardboard box to the right of the first prototype of the project (b) used a 4 × 4 grid of buttons.

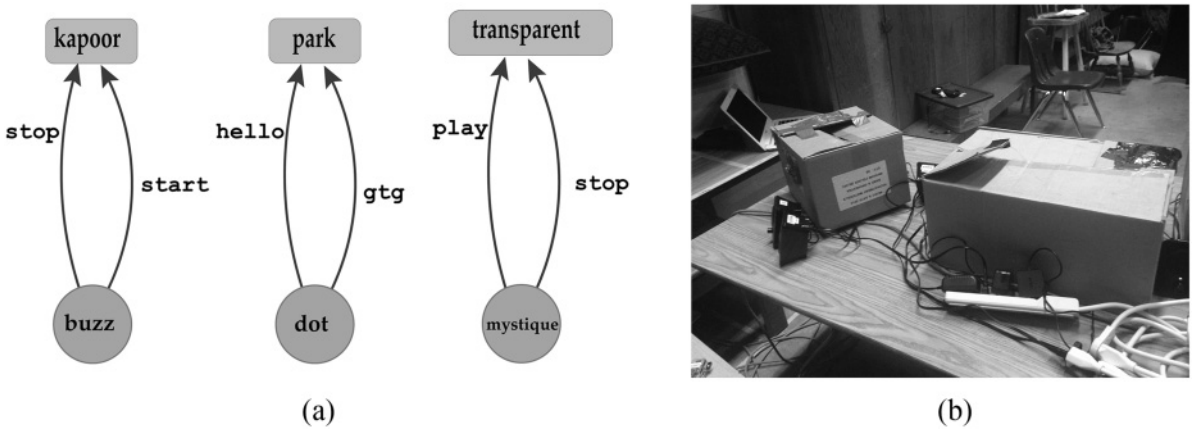


Figure 6

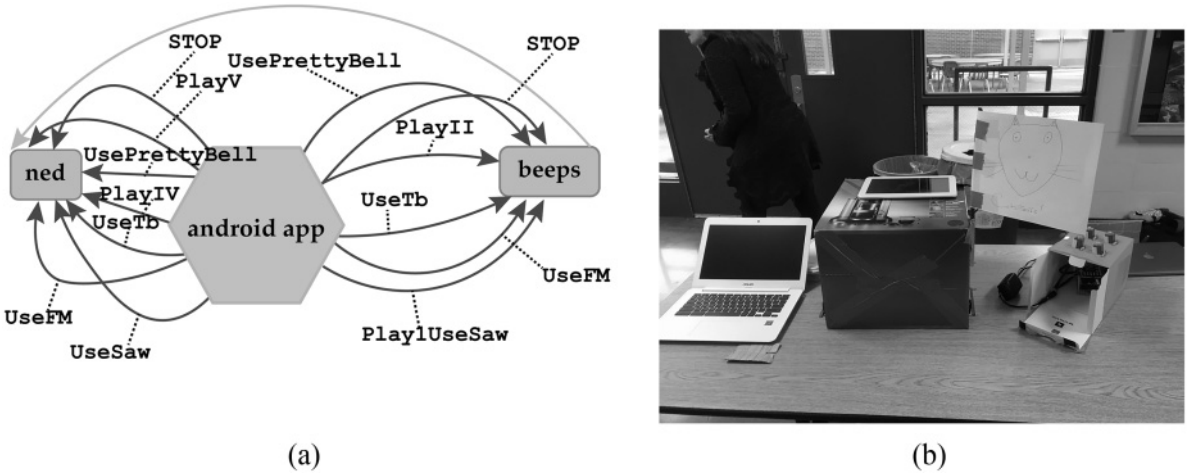
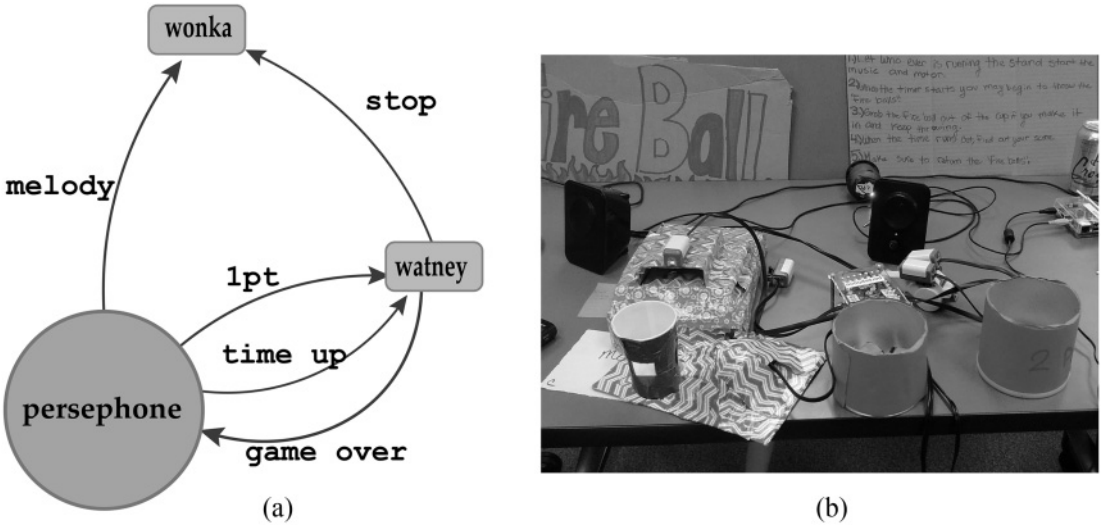


Figure 7

> **Carter:** When we press one the melody does, and when we press two it receives the message but it doesn't start playing.

Carter's explanation of the problem includes subtle details that evince a nuanced understanding of event dispatching and interdevice communication in BlockyTalky. Their program on WallE defined distinct event handlers for each button, with each handler sending a message to a different synthesizer. This arrangement has numerous places for potential failure. These include the electronic connection between the buttons and WallE; the configuration of WallE (defining the types of inputs that were connected to its various sensor ports); the programming of the event handlers on WallE (including choosing what events to respond to and what to do in response, which in this case was to send messages to two other devices when a button was pressed); the definition of event handlers on both Ned and Beeps that respond to the specific messages that WallE is programmed to send; and

Figure 8. Student project "Fireball": a carnival style game that utilized a light sensor, a motor, and a button. Players score points by tossing balls into a cup (b).



the programming of the musical motifs on Ned and Beeps. For the boys' system to function as desired, all of these elements must be implemented correctly.

When Carter notes that "it receives the message but it doesn't start playing," he is drawing upon several different pieces of knowledge. First, that pressing the button on WallE should send a message to Ned, that Ned should receive that message, and that when Ned receives the message, it should make sound. By pointing out that Ned does receive the message but does not make sound, he is pointing out that he knows that the message has been sent and has been received, and is locating the problem in the handling of that message-receipt event. He was able to determine this point of breakdown because he watched Ned's real-time event log, which displays a stream of information about all network traffic the device receives. Using this log information enabled him to troubleshoot the system relatively precisely, ruling out WallE or the network as sources of error. We believe that this is a remarkably sophisticated act of debugging for a student who only days before had never done any network programming whatsoever, and it illustrates how building computer music systems with BlockyTalky not only offers pathways into computer music, but into learning about interactive and networked systems more broadly. Ultimately, Kelly helped the students to locate the bug in their system (they had not defined the key of the motif they were trying to play), and the boys created the complex Catspresso system depicted in Figure 7.

Example 2: Synchronization Challenges

In this next conversation, a group of girls is trying to create a mashup of two songs by the band Coldplay. They are experiencing problems getting the melody and the ostinato to synchronize properly; one seems to start before the other.

> **Ivanna:** Why is it not working?
>
> **Nevaeh:** Do you still have the "wait"?
>
> **Ivanna**: Nope.
>
> **Nevaeh**: I think maybe you pressed the button too soon.

Nevaeh asks Ivanna if she has the "wait" in her code, this is in reference to a block of code called Wait for that is important to use when synchronizing two or more devices. Synchronizing synthesis in BlockyTalky involves combining three different programming language features: (1) the Sync to block, which configures a device to subscribe to another device's clock; (2) the Wait for block, which delays

subsequent synthesis until a specified beat arrives (e.g., a downbeat or a "2" in a four-count meter); and (3) the Set Tempo block, which specifies the tempo of a synthesizer in beats per minute. Using the Sync to block alone is not enough to ensure synchronized playback across devices; Wait for blocks must also be used. Because all BlockyTalky devices default to 120 beats per minute, using a Set Tempo block is only necessary for distributed synchronization if something other than the default tempo is desired, in which case all participating synthesizer nodes must use it.

In this instance, the group does not have Wait for logic in their code. When Nevaeh learns this, she responds by telling Ivanna that she may have pressed the button too soon. This is a reference to a common strategy that students developed for synchronization without programming: physically pressing multiple buttons (programmed to control different synthesizers) simultaneously. This can lead to approximate synchronization in simple musical systems, particularly those where all synthesizers use default tempos and where no Wait for blocks are used (e.g., to rest to the start of the next measure). This approach only works approximately, however (groups of novice musicians are rarely good at simultaneous button pressing), and not at all when using motifs that include any scheduling (i.e., Wait for).

After this exchange, the girls test their song again. This time the two synthesizers happen (by chance) to start at the same time. The synchronized start enables them to notice that one device seems to be playing at a faster speed than the other is.

Ivanna: Why are they going different speeds?

Nevaeh: Check your tempos.

Ivanna: Tempo 120, and . . . tempo 120.

Jasmine: There was one note that was too fast.

The girls believe that one synthesizer was playing faster than another was, so they check that both synthesizers are configured to use the same tempo. The actual problem the girls had with tempo is that they had made an error while programming the motifs on the two devices, with one motif a single beat longer than the other. This is a peculiar instance of the Pop Music Re-creation Trap. On the one hand, the girls' mashup involved arranging pieces of two different Coldplay songs, and so it was musically rich work, involving modifying the keys and tempi of those different pieces to make them compatible. Because their rearrangements also involved fidelity to extended portions of the two original Coldplay songs, however, the girls' programs also included very long motifs corresponding to those song fragments. These were considerably longer than the motifs that the students tended to compose for themselves. This caused them to end up with code on each device that was just one very long list of notes to play, making it incredibly hard to debug off-by-one errors in note durations. This led to the frustrating situation in which the girls repeatedly attempted to program a distributed BlockyTalky system using the Sync to, Wait for, and Set Tempo blocks, but were unable to achieve the results that they desired. They gave up on programming the system to be as synchronized as they desired. Ultimately, their performance of the Coldplay mashup involved choreographed button pressing to work around their motif length mismatches instead of taking advantage of programmatic synchronization features.

Evaluation of Impact on Students

In addition to capturing data (audio/video recordings and code) about the processes of students' work, we surveyed students about their enjoyment of their experience and their beliefs about computer science and music. Surveys captured information about whether the students believed that CS and music are enjoyable fields, ones that they can be successful in, and ones in which people of various ethnicities and genders can excel. Surveys also asked students whether they intend to continue with music or computing. Although most of these survey items were rated with a Likert scale, we also asked free-response questions about what students thought the best parts of their experiences were, what they thought needed improvement, and what they believed that they learned.

Table 2. Summary of Free Response Answers

Best Things		*Things Learned*		*Improvements*	
Programming	8	Programming	19	More time	5
Creating	8	Tools	7	Instruction	5
Group work	3	Building	6	Programming	5
Interest	3	Computer music	6	Participation	3
Music	2	Group work	4	Music	2
Problem-solving	2	Troubleshooting	3	Perfect	2

Free response answers given by students with counts (out of 21 students in total) who mentioned each topic in their responses.

All of the students in Johnson's class agreed to participate in our surveys, and submitted signed consent from a parent or guardian to do so. However, only a minority of Politi's students returned signed consent forms. Out of concern for how this limited participation rate might bias our sample from Politi's class, we opted to only analyze the survey data from Johnson's class.

Survey Results

Students were generally positive about their experiences. On a Likert-scale survey, 18 of 21 students reported that they somewhat or strongly liked their experiences, with "strongly liked" being the most frequent response. Three members of the research team open-coded (Strauss and Corbin 1990) the students' free responses to the prompts "the best thing about this workshop was . . . ," "describe, list, or draw three things that you learned at this workshop," and "I would improve this workshop by . . ." They iteratively converged upon the summary of students' answers that appears in Table 2.

The students overwhelmingly mentioned programming as something that they learned and enjoyed. The students also highlighted enjoying the process of building their projects and learning about the tools. It is interesting to note that even though this was a music class, for all questions music was mentioned less than other categories. This could be because the novelty was not there, given that music was the primary study in this classroom setting.

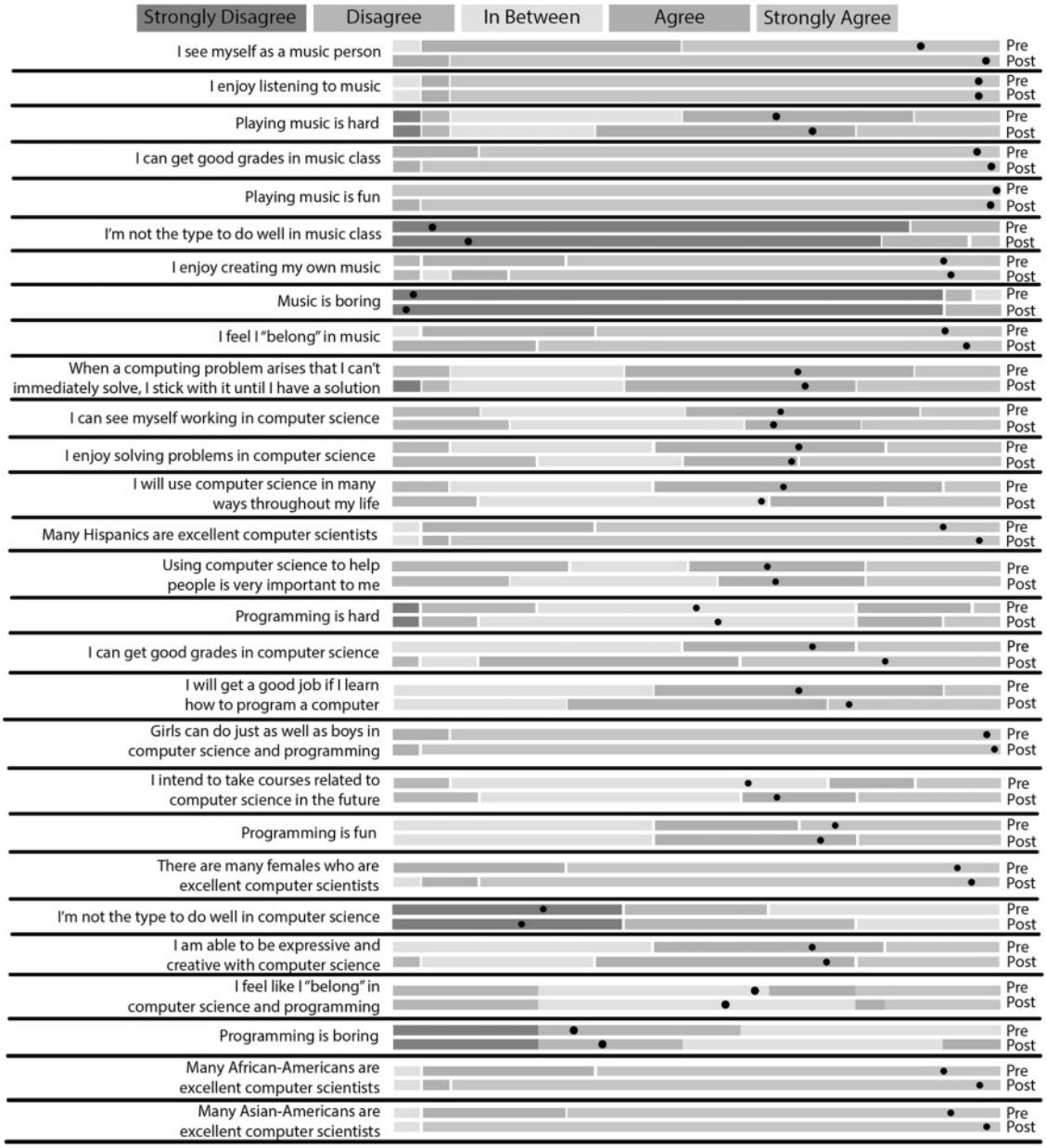
Students' preparticipation and postparticipation responses to our attitudinal questions are shown in Figure 9. Though there is considerable support for researchers treating Likert-scale data as interval data (Norman 2010; Sullivan and Artino 2013), there is also some debate about the appropriateness of doing so, with some researchers arguing that they should be treated as ordinal data only (Jamieson 2004; Allen and Seaman 2007). We have chosen to respect both perspectives by graphically presenting the distribution of responses to each survey item (depicting its ordinal character) as well as presenting descriptive and inferential statistics (an interval-oriented approach).

Students generally had small positive shifts in their view of computational and musical participation, though in many cases the survey items were limited by strong ceiling effects. In addition to computing means for all items, we took the interval perspective one step further, conducting paired Student's t-tests on all survey items. We found significant ($p > .05$) positive changes on the items about "CS grades" and "good jobs," the students seeing themselves persisting in music, and the computational excellence of people of all genders and ethnicities queried (except for girls, for which ceiling effects made significant change all but impossible). We take this as a highly positive sign that participation in programs like ours can ameliorate negative stereotypes about who can succeed in computer science.

Summary of Outcomes

The outcomes of this work suggest that creating interactive computer music systems can support

Figure 9. Distributions of students' survey responses preparticipation and postparticipation. All questions were evaluated on a Likert scale. Dots indicate mean values of responses.



creative engagement and meaningful learning. Students' projects reveal an openness to combining computing technology with music in innovative ways. They built systems that can be performed as single-player musical instruments as well as systems whose approaches to real-time interaction with music defy easy categorization. Designing, debugging, and refining these systems required students to become proficient with computational concepts beyond those encountered in typical introductory programming projects and tools for youth, including reasoning about networked communication and

synchronization. Students also designed and refined their projects with explicit attention to how the technology would integrate into human contexts of use (e.g., using music to encourage faster Rubik's-cube solving, or using manual synchronization to compensate for shortcomings in their programming of the technology). Furthermore, survey data suggest that students generally felt positive about the experiences of using BlockyTalky and that computer music may have the potential to improve perceptions of computing and to combat stereotypes.

Challenges and Open Questions

The many workshops we have run with young people have revealed several common challenges to supporting meaningful youth engagement with the design of computer music systems. As we look toward future possibilities for engaging youth in computer music creation, we also see a number of open questions ripe for exploration within the computer music community.

Embracing Pop

Making use of students' prior knowledge is a central principle of learning-environment design (Bransford, Brown, and Cocking 1999). In computer music education, nonmusicians' prior knowledge of pop music may be a pathway into deeper musical participation, both by kindling interest in participation and by offering conceptual and cultural resources for students to use within their learning. Further, students who are accomplished musicians may wish to use their knowledge of music theory to rearrange or enhance pop works. Prior work, such as in the EarSketch project, demonstrates the potential value of pop music for computer music education. We should embrace this potential. Yet our work shows how using pop music can also be fraught with perils.

Ultimately, we desire to create computer music learning experiences that are both musically and computationally rich, but the Pop Music Re-creation Trap described earlier can trap students in activities that are neither. Focusing on programmatically reproducing existing songs can lead to uninteresting systems that are neither highly interactive nor encourage computational ingenuity. Further, transcribing existing pop music into programmatic form can be prone to error for both novice and experienced musicians. We were frustrated to see how such errors led the Coldplay group, as previously described, to abandon sophisticated distributed programming techniques, which might, in turn, have facilitated a more musically satisfying performance that was less dependent on human synchronization.

Embedding music-theory education into these activities is one promising strategy to combat the Pop Music Trap. In our recent workshops, Johnson's Recipes not only provided practical means through which nonmusicians could compose songs, but also explicitly encouraged students to understand themselves as musical creators. One might further imagine using Johnson's Recipes and the derived NFAs not just as tools for teaching music theory, but as representations that could be embedded and manipulated within the BlockyTalky programming environment to support higher-level control over algorithmic music generation.

Alternatively, new tools could help students work more effectively and accurately with existing pop music. Programming tools to help students reason about how their composed musical motifs fit together over time could have helped the Coldplay group quickly identify their error and move on to other activities.

More broadly, we believe that additional design research is needed to understand how new learning experiences can offer students the benefits of drawing upon their pop music knowledge while avoiding the pitfalls that we have described.

Challenges in Real-Time Music Making

Building new educational technologies with existing hardware and open-source software can help to minimize engineering effort, provide easier pathways for teachers to gain proficiency in the tools, and achieve lower costs for schools. Support for real-time, expressive interaction with audio has not been a goal of commonly existing platforms, however.

Balancing engineering effort, cost, and suitability for real-time music-making is an ongoing concern for BlockyTalky.

Our current iteration of BlockyTalky uses Sonic Pi for sound sequencing and synthesis. Sonic Pi works well for sound synthesis, even on the relatively low-end hardware of the Raspberry Pi. It also has an existing community of educators and students. Sonic Pi is designed for live coding, however—not for performances that involve real-time control over sequencing and synthesis. It maintains a long buffer in order to avoid skipping, which means that there is often a lengthy delay, up to one second, between the time a user manipulates a sensor and when the sound they hear changes. This can be confusing to users, and the Sonic Pi project has no concrete plans to address this problem. We are currently addressing synthesis latency by replacing Sonic Pi with a new synthesis engine better geared toward real-time performance.

The cost of sensors and the need for associated "shields" or "capes" for connecting sensors to processors are also important considerations in the development of BlockyTalky. The current cost of a single BlockyTalky device with LEGO or Grove-compatible shield is US$ 100, making the cost of a set out of reach for most school programs. We are currently collaborating with the BeagleBone Foundation to port BlockyTalky to the BeagleBone Green Wireless, a US$ 40 board that is also compatible with microcontroller hardware targeting the computer music community (e.g., McPherson and Zappi 2015). This cost reduction has great potential to drive advances and wider adoption of educational technologies like BlockyTalky.

Should We Move Beyond Programming—and, if So, How?

BlockyTalky is robust and cheap enough and has sufficiently low latency to support a range of rewarding and engaging music-making activities. The drag-and-drop programming environment is also a usable tool for youth with no programming background, provided that they are open to experimenting with a wide variety of sounds and effects or that they have the musical knowledge to be able to translate their ideas efficiently into symbolic representations (e.g., lists of note names and durations). When the educational aims are to teach about musical instrument building, creative expression, design, and collaborative music performance (rather than about procedural programming or musical note reading), however, might other modes of software design be better suited to these aims? For instance, previous work has shown that building new musical instrument mappings using supervised learning—providing examples of human motions along with the musical outcomes to match those motions—can facilitate a more efficient, satisfying, and embodied approach to design (compared with programming) for professional composers (Fiebrink 2011). Might such techniques also allow youth to translate their ideas for musical instruments into real systems? Or, might techniques for symbolic transcription of sung melodies or automatic harmony generation (cf. Simon, Morris, and Basu 2008) speed up the process of "writing" programs that mimic pop songs? Might helping young people to easily realize the creative limitations of mimicry at an earlier stage of their work with technology encourage them to explore new ideas?

What Should Youth Computer Music Education Look Like?

Setting aside the question of how to embed computer music topics into CS education, what should computer music education look like for young people? How could (or should) music education itself change to incorporate computer music ideas and practices? The potential benefits of expanding musical curricula to encompass computer music topics range from increasing the relevance of music education to youth who are most excited about musical genres that rely heavily on digital production practices, to facilitating music making by youth with disabilities through custom digital instruments, to making a politically expedient argument for supporting music education because of its STEM content. But the risks include suggesting that music education is valuable only insofar as it aids

in teaching "serious" or "economically important" STEM subjects, or exacerbating disparities between well-off schools with ample resources to invest in digital-music equipment and those without them. We are excited about the potential benefits of early computer music education despite these risks, and one of our research aims is to engage the computer music research community more broadly in these questions.

Acknowledgments

We thank the National Science Foundation (CNS-1418463 and CNS-1562040), the NCWIT Academic Alliance Seed Fund, and LEGO Education for funding this work. We also thank our numerous collaborators, including Elise Deitrick, Joe Sanford, Paul Lehrman, Elena Cokova, Catherine Gao, Hilary Dwyer, Zach Lamb, Susana Gomez-Burgos, Lila Finch, and Monica Bolles.

References

ACM/IEEE-CS (Association for Computing Machinery and IEEE Computer Society Joint Task Force on Computing Curricula). 2013. "Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science." New York City: ACM Press.

Allen, I. E., and C. A. Seaman. 2007. "Likert Scales and Data Analyses." *Quality Progress* 40(7):64.

Barron, B. 2003. "When Smart Groups Fail." *Journal of the Learning Sciences* 12(3):307–359.

Bransford, J. D., A. L. Brown, and R. R. Cocking. 1999. *How People Learn: Brain, Mind, Experience, and School*. Washington, DC: National Academy Press.

Carter, L. 2006. "Why Students with an Apparent Aptitude for Computer Science Don't Choose to Major in Computer Science." In *Proceedings of the ACM SIGCSE Technical Symposium on Computer Science Education*, pp. 27–31.

Choi, H., and G. Wang. 2010. "LUSH: An Organic Eco+Music System." In *Proceedings of the International Conference on New Interfaces for Musical Expression,* pp. 112–116.

Deitrick, E., B. O'Connell, , and R. B. Shapiro. 2014. "The Discourse of Creative Problem Solving in Childhood Engineering Education." In *Proceedings of the Annual Conference of the Learning Sciences*, pp. 591–598.

Fiebrink, R. 2011. "Real-Time Human Interaction with Supervised Learning Algorithms for Music Composition and Performance." PhD dissertation, Princeton University, Department of Computer Science.

Freeman, J., et al. 2014. "Engaging Underrepresented Groups in High School Introductory Computing through Computational Remixing with EarSketch." In *Proceedings of the ACM Technical Symposium on Computer Science Education*, pp. 85–90.

Google (Google Developers). 2016. Google Blockly. Available online at developers.google.com/blockly. Accessed 17 December 2016.

Grover, S., R. Pea, and S. Cooper. 2014. "Remedying Misperceptions of Computer Science among Middle School Students." In *Proceedings of the ACM Technical Symposium on Computer Science* Education, pp. 343–348.

Jamieson, S., 2004. "Likert Scales: How to (Ab)use Them." *Medical Education* 38(12):1217–1218.

Losh, S. C. 2010. "Stereotypes about Scientists over Time among US Adults: 1983 and 2001." *Public Understanding of Science* 19(3):372–382.

Martin, C. D. 2004. "Draw a Computer Scientist." In *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, pp. 11–12.

Mason, C. L., J. B. Kahle, and A. L. Gardner. 1991. "Draw-a-Scientist Test: Future Implications." *School Science and Mathematics* 91(5):193–198.

McPherson, A., and V. Zappi. 2015. "An Environment for Submillisecond-Latency Audio and Sensor Processing on BeagleBone Black." In *Proceedings of the 138th Audio Engineering Society Convention*. Available online at www.aes.org/e-lib/browse.cfm?elib=17755 (subscription required). Accessed February 2017.

Norman, G. 2010. "Likert Scales, Levels of Measurement and the 'Laws' of Statistics." *Advances in Health Science Education* 15(625):625–632.

Patterson, D. A. 2006. "Computer Science Education in the 21st Century." *Communications of the ACM* 49(3):27–30.

Roschelle, J., and W. R. Penuel. 2006. "Co-Design of Innovations with Teachers: Definition and Dynamics." In *Proceedings of the International Conference on Learning Sciences*, pp. 606–612.

Shapiro, R. B., et al. 2016. BlockyTalky: A Physical and Distributed Computer Music Toolkit for Kids. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pp. 427–432.

Simon I., D. Morris, and S. Basu. 2008. "MySong: Automatic Accompaniment Generation for Vocal Melodies." In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pp. 725–734.

Strauss, A., and J. Corbin. 1990. "Open Coding." In *Basics of Qualitative Research: Grounded Theory Procedures and Techniques*. Thousand Oaks, California: Sage, pp. 101–121.

Sullivan, G. M., and A. R. Artino. 2013. "Analyzing and Interpreting Data from Likert-Type Scales." *Journal of Graduate Medical Education* 5(4):541–542.

Tubb, R. H. 2016. "Creativity, Exploration, and Control in Musical Parameter Spaces." PhD dissertation, Queen Mary University of London, School of Electronic Engineering and Computer Science.