



# Two Approaches to Interdisciplinary Computing+Music Courses

Jesse M. Heines, Gena R. Greher, S. Alex Ruthmann, and Brendan L. Reilly

University of Massachusetts Lowell

The developers of a university curriculum designed to bridge the gaps between the two disciplines have found that there are numerous ways to introduce arts majors to computing, and science and engineering majors to the arts.

**T**he intersection of computing and music can enrich pedagogy in numerous ways, from low-level courses that use music to illustrate practical applications of computing concepts to high-level ones that use sophisticated computer algorithms to process audio signals.

We explore the ground between these extremes by describing our experiences with two types of interdisciplinary courses. In the first, arts and computing students worked together to tackle a joint project, even though they were taking independent courses. In the second, all students enrolled in the same course, but every class was taught by two professors: one from music and the other from computer science. This course was designed to teach computing and music *together*, rather than as one in service to the other.

## WHO'S THE MORE CREATIVE?

It is the first day of a new semester. Two students walk into your class. You have never seen them before, and you

know nothing about them. When you identify them and check their primary fields of study, you see that one is a computer science major, the other a music major. Which do you assume is the more creative?

Surely, most of us would say it is the music major. The general perception is that people in the arts are more creative than those in the sciences, particularly those in computing. But is this truly the case?

Consider the types of learning experiences that characterize each field. In music, students mainly focus on the *re-creation* of art. They learn to master their instruments by studying someone else's original creations. The *composition* of original works is advanced study, typically pursued by only a handful of music majors and usually at the graduate level.

In CS, students might initially re-create programs that implement known algorithms, but they quickly progress to writing original programs to solve problems. Those problems might be carefully bounded, but good students tend to devise solutions that exhibit a wide range of approaches.

It is interesting to note that music students also must learn concepts and syntax. Think of staves, notes, key signatures, accidentals, fingerings, and so on. The difference is what they do with these. In general, they apply what they have learned to try to play a piece exactly as their teachers say it should be played. CS students try to apply what they have learned to *solve a problem* outlined by their teachers.

So, on reconsideration, which do you now judge to be the more creative?

**Table 1. Computing+Music course offerings.**

Listing department	Number	Percent
Music	40	77
Computer science	9	17
Co-listed	3	6
Type of instruction	Number	Percent
Single instructor	28	54
Team teaching	8	15
Not identified	16	31
Student level targeted	Number	Percent
1st- and 2nd-year undergraduates	21	40
3rd- and 4th-year undergraduates	19	37
Graduate students	5	10
Multiple levels	7	13

### INTERDISCIPLINARY LEARNING

It is not our purpose, of course, to instigate an argument over who is more creative than whom. But it certainly is our purpose to break stereotypes and to stress that when we look at science and engineering majors versus their peers in the arts, business, and other supposedly non-technical majors, it is clear that they have much to learn *from each other*. It is not much of a stretch to assert that the technologies most of our CS graduates will be working on 5 to 10 years after they graduate probably have not been invented yet. This can make it a bit hard to decide what or how we should teach them. We have therefore based our work on the following postulates.

*Once our CS students graduate, it is very likely that they will never again write a program of any significant size by themselves.* Instead, they will work in teams, and those teams will undoubtedly be interdisciplinary. Even if certain members of the team do not write a single line of code, they will have a say not only in what a program does, but also in how it is implemented.

*Basic skills will remain basic.* An array will always be an array, and a linked list will always be a linked list. With all the buzz about students seeking CS programs with concentrations in game development, programmers who succeed in that subfield will be those who understand that interesting games are built on the fundamentals of algorithms and data structures, just as musicians understand that interesting music is built on the fundamentals of melody, rhythm, and harmony. As Michael Zyda stated, “The game industry ... wants graduates with a strong background in computer science. It does not want graduates with watered-down computer science degrees, but rather an enhanced set of skills.”<sup>1</sup>

*The need for everyone to have basic computer skills will only increase.* Jeanette Wing stated that the basic skill in problem solving is “computational thinking,” which “involves solving problems, designing systems, and un-

derstanding human behavior, by drawing on the concepts fundamental to computer science.”<sup>2</sup> According to Wing, this “is a fundamental skill for everyone, not just for computer scientists.” We strongly agree, and we feel that exposing arts students to computational thinking *within their own field* has huge potential for enhancing their education.

*Everyone has something to learn from everyone else.* Virtually all jobs today involve interdisciplinary teams, and working in such teams usually requires abandoning assumptions about our coworkers’ fields. Reflecting on one of the assignments in our interdisciplinary course, a CS major wrote, “It was great to work with someone as musically (and graphically) inclined as Maria [a music major]. I lack a lot of knowledge about both of those, and her ideas made very notable improvements in the programming as well as the music and graphics.” Note that the CS major specifically mentions improvements to the *programming* based on ideas from the music major.

### COMPUTING+MUSIC COURSES

To address these issues, we developed two interdisciplinary course models that our colleague Fred Martin dubbed *synchronized* and *hybrid*.<sup>3</sup> The synchronized model pairs two independent, upper-level courses in different disciplines and requires interdisciplinary teams of students to complete a project collaboratively. The hybrid model is a single course taught by two professors from different disciplines, with both in the classroom throughout the semester.

These are, of course, but two of myriad models employed in interdisciplinary computing+music courses. To put our work in perspective, we took an *informal* look at 52 courses at 40 colleges and universities that cover computing through music or music through computing. Some of these were identified by attendees at a March 2011 workshop on this topic under the auspices of the ACM SIGCSE Music Committee<sup>4</sup> and sponsored by the NSF-funded LIKES project<sup>5</sup> ([www.likes.org.vt.edu](http://www.likes.org.vt.edu)). Additional courses were found by the student researcher on our team, who searched the Web for syllabi that combined computing and music in interdisciplinary courses.

Our search criteria specifically *excluded* audio recording and production courses that have the shaping of sound through electronics and signal processing as their primary objectives. Although these courses fall at the intersection of computing and music, they focus on *using* technology to achieve desired sounds rather than *teaching* computational and musical concepts together. Table 1 presents general information about the courses we discovered and gives an overall picture of the landscape.

Table 2 presents the content of the 52 courses, as gleaned from their posted syllabi. This is an inexact measure, to be sure, but it still gives a somewhat reasonable view of the field. (The numbers in each section do not add

up to 52 and the percentages do not total 100 percent because some entries fall into more than one category.)

There is indeed a large range of courses offered, subjects covered, perspectives taken, teaching styles employed, and software systems used. Based on a review of this data and reflections on our familiarity with some of the teachers of these courses, the following overall picture emerges:

- At the upper end of the curriculum, virtually all courses that cover computing+music are advanced offerings by music departments. We know of no upper-level CS courses dedicated to addressing issues faced by musicians (although of course there may be some unknown to us).
- Courses and research at the upper end require deep understanding of *both* computation and music. Examples include the algorithmic composition work by Michael Edwards<sup>6</sup> and by Andrew Brown and Andrew Sorenson.<sup>7</sup>
- At the lower end of the curriculum, music is typically used to demonstrate or to introduce concepts. This is music *in service to* computing, not music *integrated with* computing. An example is the media computation work by Mark Guzdial and Barbara Ericson.<sup>8</sup>

Our work attempts to fill some of the gaps between these types of courses by integrating computing and music at a high conceptual level. The synchronized course targets mid- to upper-level music and CS majors with the intent of furthering students' knowledge of both. The hybrid course is a general education offering open to all students in the university. It attempts to provide an understanding of where computing and music interact, at a level that is accessible to students without deep knowledge of one or the other. Thus, our work is at both ends of the instructional spectrum.

## COMBINED GUI PROGRAMMING AND MUSIC METHODS

One way to get started in interdisciplinary teaching and learning is to connect the students in two existing courses through a joint project. Administratively, this is a "low-hanging fruit" approach because it does not involve getting a new course approved or making any changes to the course catalog. All that is needed are professors who agree to collaborate with each other to build an interdisciplinary project into their courses.

In our case, the CS professor teaches a project-based course in graphical user interface programming, which fit nicely with a project-based course on teaching methods taught by the music professors. After reviewing the projects that we assign in our respective courses, we decided to make our initial foray into interdisciplinary teaching using a "found instruments" project that has been used in music for years.

**Table 2. Computing+Music course content.**

Disciplines covered	Number	Percent
Sound/audio	37	71
Computer science	36	69
Music (composition)	22	42
Music (theoretical)	12	25
Media	5	5
Primary focus	Number	Percent
Composition	31	60
Sound symbols	27	52
CS (introductory)	18	35
Sound processing	17	33
CS (specialized)	12	23
Music theory	7	13
Interactive media	1	2
Software used	Number	Percent
Max/MSP	11	21
Audacity	4	8
Processing	4	8
SuperCollider	3	6
ChucK, Disklavier, Pro Tools, Reason	2 each	4 each
Audition, Garage Band, Matlab, Peak, PureData, Reaktor, Scratch, Sibelius	1 each	2 each

### The music assignment

For the musicians, the purpose of our assignment is similar to Andrew Hugill's description of a project intended "to strip away previous ideas of 'musicianship,' [by] reevaluating the sounding properties of objects, how they may be made into instruments, how playing techniques might be developed, and how music may be created as a result."<sup>9</sup> Music students are asked to do the following:

- Using only household object(s), create a musical "instrument" that can produce several different pitches or timbres. Your instrument must be able to produce several different types of sounds, or sounds with several different characteristics.
- Create a composition for your instrument that employs a specific musical form of your choice. It need not be long. A 2-3 minute piece is sufficient, but it must include distinct sections that give it form. That is, your composition must include distinctive opening, middle, and closing sections.
- Devise a system of creative notation that others will be able to understand well enough to perform your composition. Your notational system should not resemble traditional musical notation in any way.



Figure 1. Mike playing his jacket as a found instrument.

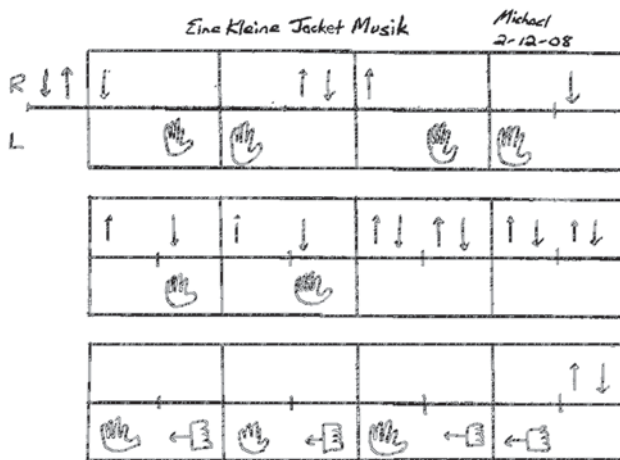


Figure 2. Mike's notation for his composition.

- Bring your instrument and notated composition to class. Come prepared to explain your work and to perform your piece.

To achieve camaraderie and pique interest, the CS majors are also given this assignment. Our experience is that the CS students “find” instruments that exhibit just as much novelty as those of their music counterparts. When we get the students from the two courses together, we do several things to build community, including having them jam on their instruments in mini-ensembles. Again, the CS students “get into” this project just as much as the music students, and the resultant “music” is, well, “interesting.”

In another class activity, students try to play each other's found instruments from the notations created for

those instruments. We have them do this without first hearing the original composer play the piece and without any verbal explanation of the notational system. This is a good test of the communicability of the notation by itself, and it opens up several avenues for discussion of human factors. As an example of this activity, see [www.youtube.com/watch?v=IJuGoYnCxSs](http://www.youtube.com/watch?v=IJuGoYnCxSs).

### The computing assignment

The found instruments project connected to computing through the creative notation. In this project, we

- introduced CS students to standard music notation software using Finale NotePad ([www.finalemusic.com/NotePad](http://www.finalemusic.com/NotePad)) and Noteflight ([www.noteflight.com](http://www.noteflight.com));
- assigned CS and music teams and charged the CS students with creating a music notation program for the notation devised by their music partners; and
- scheduled several joint classes in which the music students could work with the CS students on the programs' designs, review the CS students' works in progress and offer comments and suggestions for improving the programs, and finally act as usability test subjects on the finished products.

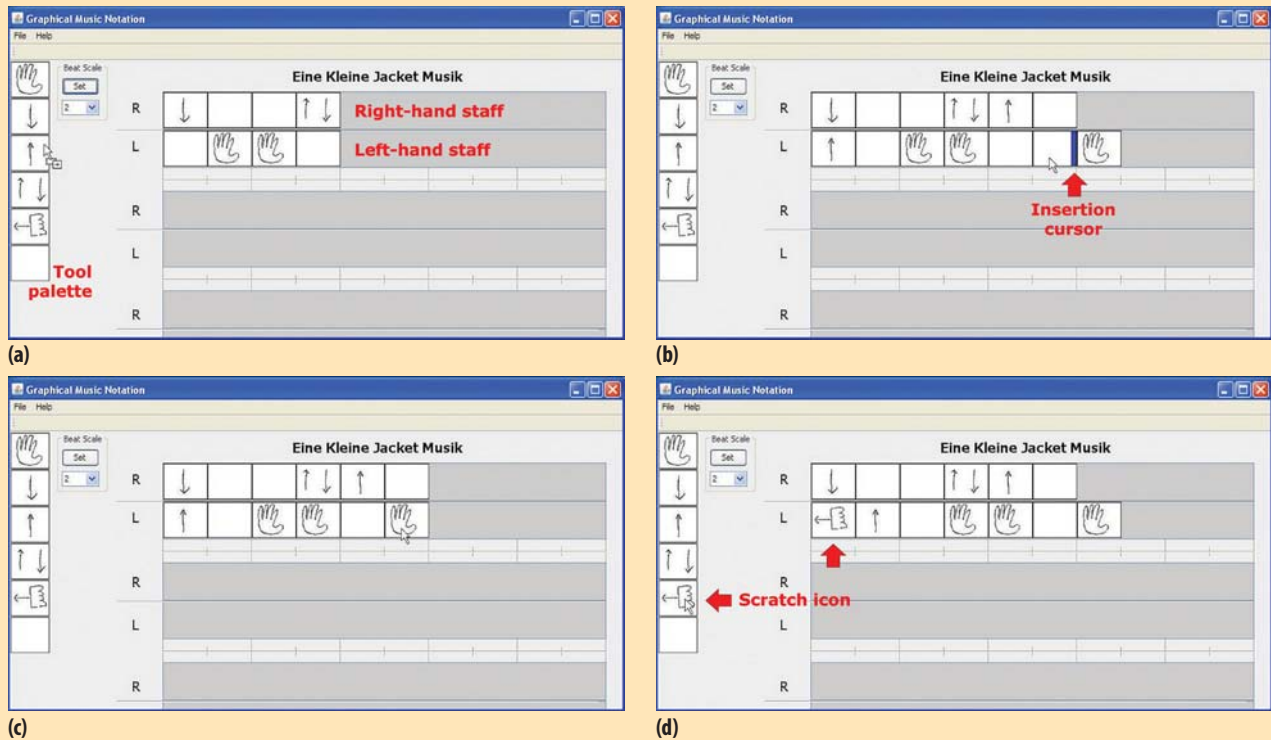
Some of the programs produced as a result of these collaborations and the lessons learned from them were truly astounding.

As Figure 1 shows, in one of the best of these projects, Mike, a music student, used his jacket as a found instrument, creating sounds by slapping it, rubbing it, working the zipper, and so on. He then created a piece satirically named *Eine Kleine Jacket Musik*. Figure 2 shows an excerpt from Mike's creative notation. Performances of Mike's piece first by Chase, a CS student, and then by Mike himself are posted at [www.youtube.com/watch?v=iD4dEZOTilg](http://www.youtube.com/watch?v=iD4dEZOTilg).

Figure 3 shows part of Mike's partner Chris's composition to demonstrate the CS concepts and skills involved in developing such a program.

In Figure 3a, a few icons from the tool palette on the left in the composition program have been placed onto the right- (R) and left-hand (L) staves in the composing area by either dragging and dropping them or double-clicking on them in the tool palette. In Figure 3b, the insertion cursor is positioned between the sixth and seventh icons on the left-hand staff, as indicated by the thick vertical bar. At this point, double-clicking on an icon in the tool palette would insert the icon to the right of the insertion cursor, which is to the left of the last icon on staff L.

In Figure 3c, the backspace key has just been pressed, and the blank (or “rest”) icon that the arrow cursor in Figure 3b pointed to has disappeared. The issue is that the thick vertical bar insertion cursor has also disappeared,



**Figure 3.** Chris’s music composition program for Mike’s jacket notation. (a) State 1: a few icons from the tool palette on the left in the composition program have been placed onto the right- (R) and left-hand (L) staves in the composing area. (b) State 2: the insertion cursor is positioned between the sixth and seventh icons on the left-hand staff. (c) State 3: the icon pointed to by the arrow cursor in Figure 3b has been deleted. (d) State 4: the scratch icon in the tool palette (indicated by the arrow cursor) has been double-clicked to insert it into the composition.

leaving users to wonder where the insertion point is. In most editors, the insertion point would not change—that is, if the user double-clicked an icon in the tool palette at this point, that icon would still be inserted to the left of the right-most hand icon on staff L.

Unfortunately, this is not what happens. Instead, when the “scratch” icon is double-clicked, it is inserted at the beginning of the staff, as Figure 3d shows. This may be fully logical to a programmer who has implemented the composition area as a pair of linked lists, but it is not at all logical to someone used to working with any sort of text editor.

When the anomaly was pointed out to Chris, he immediately recognized the problem and said, “I can’t believe I didn’t notice that.” But that’s exactly why usability tests are needed. Programmers are often “too close” to their work to see even the most obvious user interface issues. Teaching this point in a lecture setting requires students to mentally connect theory and practice. When it is learned from a peer while testing the student’s own software, the connection is far more concrete, and the lesson is learned at a deeper level that is more personal and, therefore, more effective.

Thus, the fresh views of students in other disciplines can teach valuable lessons to our computing students.

Likewise, for music majors, helping nonmusicians translate their musical concepts into computer programs can shed light on the clarity of their thinking—or lack thereof. Such reciprocal learning,<sup>10</sup> in which students learn from each other instead of just from their professors, exemplifies one of the best characteristics of interdisciplinary courses.

### SOUND THINKING

Our synchronized courses worked well at the upper end of our curricula, but we also wanted to work at the lower end so that we could introduce more students to the benefits of interdisciplinary courses. Following the pioneering work of Holly Yanco and colleagues in combining art and robotics at our university,<sup>11</sup> we developed Sound Thinking, a new hybrid course that could be offered to all students in the university (<http://soundthinking.uml.edu>).

Two characteristics about the way in which Sound Thinking was put into the course catalog contributed significantly to its success. First, it was co-listed in both the music and CS departments. Second, we applied for and were granted general education status for the course. Arts students who take it register using the CS department number and receive science and technology general education credit. Science students register using the music department number and



Figure 4. Top view of the lever drumitar.

SPL:  $\frac{1}{3}$

1	■		
2			
3	■		
4			
5	■		
6			
7	■		
8			
9	■		
10			
11	■		
12			
13	■	V	
40		V	
41	■	V	
42		V	
43		V	
44		V	
45	■	V	
46		V	
47		V	
48		V	
49	■	V	
50		V	
51		V	
52		V	

Figure 5. Notation for playing the lever drumitar.

receive arts and humanities credit. These characteristics were essential to achieving the critical number of registrations needed for the course to run, especially with two professors present at all class meetings.

### Revisiting found instruments

We also used the found instruments project at the beginning of Sound Thinking, but we took it in another direction. After students created their instruments and notations, we had them record the sounds their instruments could make and then used those as an introduction to sound editing.

Eric, a CS student, created what he called a “lever drumitar,” shown in Figure 4. He strung a guitar string across the opening of a cup, secured it with strong tape, and rigged up a carabiner to use as a lever for changing the cable’s tension. This allowed him to produce different sounds when he strummed the cable with a soda can tab.

Figure 5 shows the original notation that Eric created for his instrument. Each row represents an action. If the square in the second column is filled in, the string is to be strummed. A V in the third column indicates that the time duration is to be shortened. The length of the line in the fourth column indicates the carabiner’s position.

For the next assignment, students recorded the various sounds their found instruments could generate and loaded them into Audacity. They then created original compositions by looping and combining those sounds. To hear Eric’s original lever drumitar sounds and his remixed composition, go to [www.youtube.com/watch?v=\\_zA\\_hn\\_4T8k](http://www.youtube.com/watch?v=_zA_hn_4T8k).

### Extending found instruments

For the next assignment, students loaded their sounds using the Scratch programming language<sup>12</sup> and sequenced those sounds by chaining “play sound until done” blocks together. Initially, they just created linear chains like that shown in Figure 6a. When they wanted to repeat a sound or just use it again, they simply dragged in another block and selected the sound they wanted it to play.

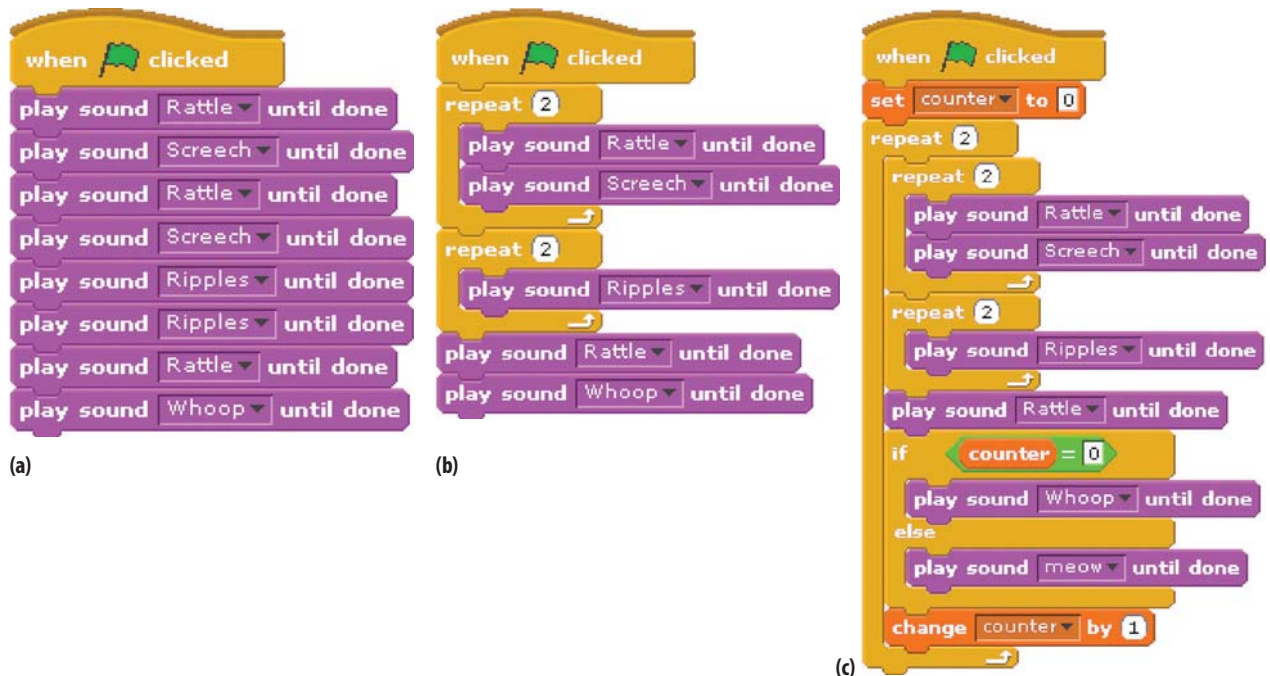
With a bit of experimentation, all the students succeeded in creating Scratch programs that used looping as shown in Figure 6b. With a bit more instruction and encouragement, most students were able to incorporate variables, nested loops, and conditional structures as shown in Figure 6c, as well.

Finally, with help from each other rather than from the professors—which indicates true student involvement in the course and is the best way for them to learn: by teaching others—some students figured out how to do more advanced things, such as playing two or more sounds simultaneously using the “play sound” and “broadcast” and its complementary “when I receive” block, leading to interesting and sometimes relatively complex discussions about synchronization.

Many CS concepts are at play here, and we use the word “play” intentionally. The Scratch development group at the MIT Media Lab is called the Lifelong Kindergarten Group for good reason. The ability to learn through *thoughtful* play that involves the use of creativity is at the heart of what we are trying to achieve. The music and arts students learn about computing, to be sure, but so do the CS and engineering students.

Using a visual programming environment like Scratch forces CS majors—who have been “brought up” on languages like C/C++ and Java and on text-based coding environments—out of their comfort zone. It is amazing how many of them stumble when they discover that a Scratch loop does not provide access to its index (counter) variable. It is pretty easy for them to implement a counter, but solving this problem requires a bit of creative thinking. In addition, explaining to their nontechnical peers what they are doing not only increases their partners’ understanding, but solidifies their own as well. As the saying goes, “If you really want to learn something, teach it to someone else.”

Sound Thinking builds on the found instruments project and its related assignments by introducing MIDI concepts and generating music using Scratch’s various



**Figure 6.** Scratch programs that chain sound blocks together to play (a) a straight sequence of sounds, (b) a sequence of sounds using loops, and (c) a looped sequence with conditionals.

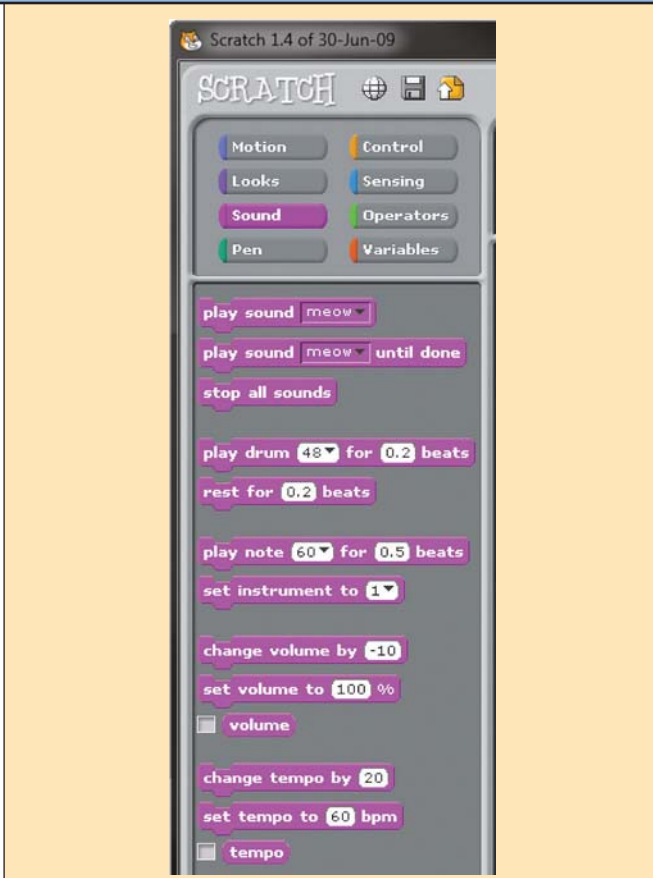
“sound” blocks, shown in Figure 7. We have created several different types of assignments using these blocks, including having students write a composition based on only major 2nds and perfect 5ths (to take music majors out of their Western music comfort zone), writing algorithms to transpose lists as either MIDI values or interval deltas into different keys, and coding multiple parts that must be carefully synchronized.

These and other assignments are described in detail at [soundthinking.uml.edu](http://soundthinking.uml.edu). Through these assignments, music majors learn about computing, and CS students learn about music.

### INTERDISCIPLINARY TEACHING

One measure of the success of our work is the lasting effect it has on students. This is difficult to assess, but the number of students who return semesters later to tell us how they applied the concepts they learned in a different context gives us confidence that at least some of the activities we developed have generated good results. We are currently working to devise more rigorous evaluations to substantiate this belief.


In addition, the effects of our interdisciplinary experiences were not limited to the students. The professors also learned from each other, not only about discipline-specific content, but also about teaching and pedagogy. As the NSF evaluator of our Performamatics project wrote in her final report:



**Figure 7.** Blocks available from the Scratch sound panel.

One CS faculty member ... changed his approach to teaching significantly in some situations, assigning more open-ended projects, a change well received by students. ... Change in faculty is an essential but often overlooked element of institutional and curricular change.

The professors' experiences in teaching with each other were so positive that they continued to do so even after the original NSF funding expired. Then in 2011, we were awarded a grant from the NSF TUES program to disseminate our work in a series of workshops for interdisciplinary pairs of professors. The first of these free workshops will be offered 21-22 June 2012. Faculty interested in attending are invited to visit [www.performamatics.org](http://www.performamatics.org) for further information and to apply.

Our explorations of ways to bridge the gaps in computing+music education are really just beginning. We believe that there are many more ways to introduce arts majors to computing and science and engineering majors to the arts, and that our approaches offer effective ways to work toward that goal in an undergraduate institution. We are constantly working to improve our current methods and to extend our work into more advanced offerings that move into live coding<sup>13-15</sup> and text-based music coding environments such as SuperCollider, Impromptu, Processing, and Max/MSP. 

## Acknowledgments

This work is supported by National Science Foundation Awards 0722161 and 1118435. In addition to the authors, Fred Martin and Sarah Kuhn of the University of Massachusetts Lowell and Scott Lipscomb of the University of Minnesota are members of these project teams. Lipscomb thoroughly reviewed an early draft of this article and provided significant suggestions for improvement. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF. Please see [www.performamatics.org](http://www.performamatics.org) for further information, including how to apply to attend our NSF-sponsored workshop.

## References

1. M. Zyda, "Computer Science in the Conceptual Age," *Comm. ACM*, vol. 52, no. 12, 2009, pp. 66-72.
2. J.M. Wing, "Computational Thinking," *Comm. ACM*, vol. 49, no. 3, 2006, pp. 33-35.
3. F. Martin et al., "Joining Computing and the Arts at a Mid-size University," *J. Computing Sciences in Colleges*, vol. 24, no. 6, 2009, pp. 87-94.
4. R. Beck and J. Burg, "Report on the LIKES Workshop on Computing and Music," ACM SIGCSE Music Committee, to be published in 2012.
5. W. Chung et al., "LIKES: Educating the Next Generation of Knowledge Society Builders," *Proc. 15th Americas Conf. Information Systems (AMCIS 09)*, Assoc. for Information Systems, 2009; [www.likes.org.vt.edu/files/LIKES\\_AMCIS09\\_pub.pdf](http://www.likes.org.vt.edu/files/LIKES_AMCIS09_pub.pdf).
6. M. Edwards, "Algorithmic Composition: Computational Thinking in Music," *Comm. ACM*, vol. 54, no. 7, 2011, pp. 58-67.
7. A.R. Brown and A. Sorensen, "Interacting with Generative Music through Live Coding," *Contemporary Music Rev.*, vol. 28, no. 1, 2009, pp. 17-29.
8. M. Guzdial and B. Ericson, *Introduction to Computing and Programming in Java: A Multimedia Approach*, Prentice Hall, 2005.
9. A. Hugill, *The Digital Musician*, Routledge, 2008.
10. H.F. Silver, R.W. Strong, and M.J. Perini, *Strategic Teacher: Selecting the Right Research-Based Strategy for Every Lesson*, chapt. 13, Assoc. for Supervision & Curriculum Development, 2008.
11. H.A. Yanco et al., "Artbotics: Combining Art and Robotics to Broaden Participation in Computing," *Proc. AAAI Spring Symp. Robots and Robot Venues: Resources for AI Education*, 2007; [www.cs.hmc.edu/roboteducation/papers2007/c39\\_yancoArtbotics.pdf](http://www.cs.hmc.edu/roboteducation/papers2007/c39_yancoArtbotics.pdf).
12. M. Resnick et al., "Scratch Programming for All," *Comm. ACM*, vol. 52, no. 11, 2009, pp. 60-67.
13. A. Brown, "Generative Structures Performance," 2011; <http://vimeo.com/26193440>.
14. A. Ruthmann, "Live Coding & Ichiboard-Enhanced Performance," 2011; [www.youtube.com/watch?v=qehSEroHn4E](http://www.youtube.com/watch?v=qehSEroHn4E).
15. A. Sorenson and A. Brown, "aa-cell Live Coding at the Loft 2," 2007; [www.youtube.com/watch?v=tj74-q\\_Mxrg](http://www.youtube.com/watch?v=tj74-q_Mxrg).

**Jesse M. Heines** is a professor of computer science at the University of Massachusetts Lowell, with a strong interest in music and its power to interest students in computing. Heines received an EdD in educational media and technology from Boston University. Heines and Gena Greher are writing a book on interdisciplinary teaching that is currently under contract with Oxford University Press. Contact him at [jesse\\_heines@uml.edu](mailto:jesse_heines@uml.edu).

**Gena R. Greher** is an associate professor of music education at the University of Massachusetts Lowell. Her research focuses on creativity and listening skill development in children and on examining the influence of integrating multimedia technology in urban music classrooms. Greher received an EdD in music and music education from the Teachers College of Columbia University. Contact her at [gena\\_greher@uml.edu](mailto:gena_greher@uml.edu).

**S. Alex Ruthmann** is an assistant professor of music education at the University of Massachusetts Lowell, where he teaches courses at the intersection of music education and arts computing. His research explores social/digital media musicianship and creativity, as well as the development of technologies for music learning. Ruthmann received a PhD in music education from Oakland University, Michigan. Contact him at [alex\\_ruthmann@uml.edu](mailto:alex_ruthmann@uml.edu).

**Brendan L. Reilly** is an undergraduate computer science major at the University of Massachusetts Lowell. He has played bass since grade school, participating in every musical group available to him. Contact him at [brendan\\_reilly@student.uml.edu](mailto:brendan_reilly@student.uml.edu).