

# Teaching Computational Thinking through Musical Live Coding in Scratch

Alex Ruthmann

Dept. of Music  
Univ. of Massachusetts Lowell  
Lowell, MA 01854  
1-978-934-3879

Alex\_Ruthmann@uml.edu

Jesse M. Heines

Dept. of Computer Science  
Univ. of Massachusetts Lowell  
Lowell, MA 01854  
1-978-934-3634

heines@cs.uml.edu

Gena R. Greher

Dept. of Music  
Univ. of Massachusetts Lowell  
Lowell, MA 01854  
1-978-934-3893

Gena\_Greher@uml.edu

Paul Laidler

Student, Dept. of Computer Science  
Univ. of Massachusetts Lowell  
Lowell, MA 01854  
1-978-934-3634

plaidler@gmail.com

Charles Saulters II

Student, Dept. of Music  
Univ. of Massachusetts Lowell  
Lowell, MA 01854  
1-978-934-3634

kingd615@hotmail.com

## ABSTRACT

This paper discusses our experience in teaching an interdisciplinary general education course called *Sound Thinking* that is offered jointly by our Dept. of Computer Science and Dept. of Music. It focuses on the student outcomes we try to achieve and the projects we use to help students realize those outcomes. It explains why we are moving from a web-based environment using HTML and JavaScript to Scratch and discusses the potential for Scratch's "musical live coding" capability to reinforce those concepts even more strongly.

## Categories and Subject Descriptors

K.3.2 [Computers and Education]: Computer and Information Science Education — *computer science education, curriculum*.

## General Terms

Design, Languages

## Keywords

Performamatics, Scratch, computer science education, interdisciplinary courses, musical live coding, generative music, curriculum design.

## 1. PERFORMAMATICS BACKGROUND

Performamatics is a series of courses intended to attract students to computer science (CS) by tapping their inherent interest in performance and the arts. Toward that end, two CS professors have teamed with five Music, Theater, and Art professors to offer both introductory and advanced courses where assignments designed to reinforce CS concepts center around applications in

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

*SIGCSE'10*, March 10-3, 2010, Milwaukee, Wisconsin, USA.  
Copyright 2010 ACM 978-1-60558-183-5/09/03...\$5.00.

the Arts. These courses have been described in a number of other papers and presentations [3, 4, 5, 6, 9], and readers are referred to the project website at <http://www.performamatics.com> for links to online materials as well.

The most successful Performamatics courses to date (based on size of enrollment and student feedback) have clearly been the introductory ones. These are set up as general education (GenEd) courses and co-listed in two departments, allowing CS majors to earn Arts & Humanities GenEd credit while Arts majors earn Science & Technology GenEd credit. *Tangible Interaction Design* is a collaboration between CS and Art, while *Sound Thinking*, the course on which this paper focuses, is a collaboration between CS and Music.

## 2. SOUND THINKING, VERSION 1

One of the hurdles in getting our first offering of *Sound Thinking* approved for dual GenEd credit was to convince the GenEd committee that Music majors would learn something about technology and CS majors would learn something about music. The committee feared that if project teams had both CS and Music majors, each group would naturally navigate to its own discipline and there might be relatively little true cross-over. We therefore established the following behavioral objectives for all students.

Upon completion of this course, students should be able to:

1. Identify properties of sound and describe the organization of sound into music.
2. Design a simple notation system and describe the differences between formal and informal notation.
3. Distinguish between analog and digital audio.
4. Discuss the basic differences between various audio file formats and sound compression techniques.
5. Create a web-based computer program that plays a music file.
6. Create a web-based computer program that plays a user-definable sequence of music files.

In the first half of the semester, students created compositions for "found" instruments, invented notations for those instruments, recorded the instruments' sounds, manipulated those sounds with

audio editors, and remixed and recomposed the sounds into original compositions. In the second half, they created webpages that incorporated music, used a JavaScript Application Programmer Interface (API), and developed interactive web applications in which music played an integral part. Looking back, we feel that objectives 1, 2, 5, and 6 were achieved by all students, while 3 and 4 definitely go short shrift.

Before the course was taught, there was much discussion among the Performamatics faculty about the development platform to be used for programming assignments. A CS professor said, “The Music majors will cringe if you make them code. You’ve got to use a visual programming environment.” But a Music professor strongly disagreed, saying “One of our goals is to have them overcome any fear they have of code. We *want* them to see real code.” Given that we thought students would enjoy creating webpages that they could share with their friends, we therefore chose Dreamweaver as our development platform, because it allowed viewing the page layout and its underlying code simultaneously. This helped students easily see the cause-and-effect relationship between the code and its result. We taught the basics of underlying HTML and JavaScript (with a custom API to play sounds and sound files), and only one of the 13 students had any real trouble doing the webpage development assignments.

On the contrary, most of the Music majors were very technically savvy and the post course student evaluations revealed that they wanted to know *more* about what was “under the hood.” They enjoyed referring to the CS professor on the faculty team as a “magician” (because he could almost always make their pages do what they wanted when their CS partners were stumped), and they suggested that when the next the course is taught, the programming should be spread throughout the semester rather than confined to the second half.

### 3. SOUND THINKING, VERSION 2

We are now well along in our plans to revise *Sound Thinking* for its next offering. With the help of students funded through a Research Experience for Undergraduates supplement to our NSF CPATH grant, we investigated a number of other platforms for use in the course. We have settled on Scratch [7, 8], and the remainder of this paper discusses why we feel that Scratch is an appropriate platform for teaching computational thinking through music.

### 4. MAKING MUSIC WITH SCRATCH

Scratch has the ability to generate and play sounds using various components in its Sound category. But if one wants to begin making *music* with more than one sound line in Scratch, one needs to address the issue of synchronization.

This issue is best addressed once students are familiar with looping, an essential concept in the implementation of many musical models. As students begin working with models where multiple voices are layered, it becomes necessary to maintain the tempo using the Scratch timer. Figure 1 shows a loop that will play a hand clap (MIDI drum instrument #39) every quarter of a beat. However, this example will not have a steady tempo, and if it was used to layer multiple voices, each would eventually fall out of synchronization.

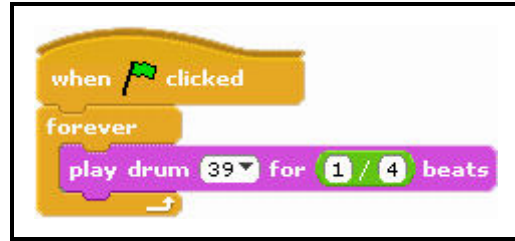


Figure 1. A hand clap loop.

The Scratch timer offers a way to address this problem. We first determine how many seconds our hand claps should be apart and then use the Scratch timer to control when each clap should occur. Figure 2 shows a more complex loop that will remain synchronized with the tempo regardless of the number of iterations performed. Each iteration waits until the Scratch timer reaches the value stored in variable *now*. This variable holds the time when a hand clap should sound. A message is then broadcast to play the hand clap. This ensures that the loop will complete and return to the “wait until” statement before the next hand clap needs to be played by delegating the actual playing to a “when I receive” event handler. The value of variable *now* is then changed to contain the time at which the next hand clap should be played. The Scratch timer ensures that the hand claps remain in tempo.

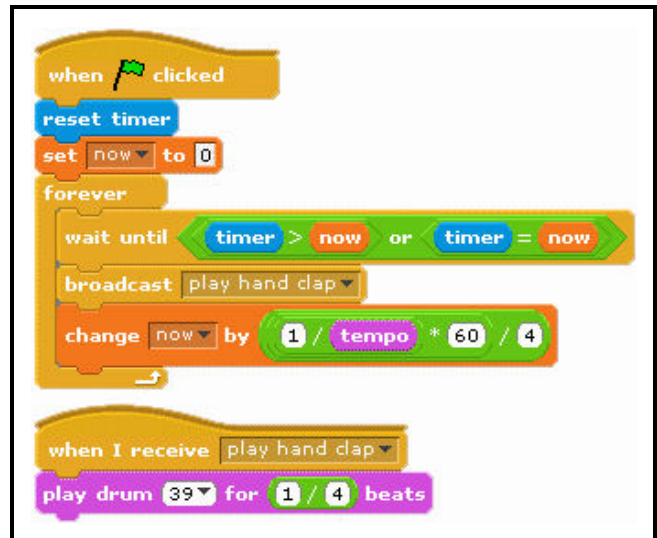


Figure 2. Hand clap loop synchronized via the Scratch timer.

Note that in addition to synchronization, many other computational thinking concepts are touched upon by this example even before getting into simple melodies.

- looping
- initialization
- use of variables
- changing variables algorithmically
- modularization
- event processing

### 5. FROM CODE TO MUSIC

Once students understand basic note and sound generation in

Scratch and can implement synchronization, more musical, generative algorithms for creating and manipulating sequences of notes can be explored.

One possible starting point is to use the “forever” loop to generate random melodies constrained by lower and upper boundaries as shown in Figure 3. In this example, random musical pitches from middle C (MIDI note 60) and the C above that (MIDI note 72) are chosen and played for half a beat. Because the “play note” function is surrounded by a “forever” loop, Scratch continues to generate notes until the Scratch stop button (●) is pressed.



Figure 3. Code for a random melody by boundary constraints.

The fact that Scratch also functions as a live interpreter/compiler makes things more interesting. This feature allows the boundaries of the random pitches as well as the duration of the sound played to be manipulated in real time through “musical live coding” [2, 11, 12] without disrupting the sounds being generated. That is, the resultant melody can be changed in real time by adjusting the upper and lower bounds of the random function and changing the duration value for the beat without stopping program execution.

The code in Figure 3 can be expanded to generate a random melody from notes provided in a pitch set as shown in Figure 4. This code implements a Scratch “list” that contains a Pentatonic (five note) pitch set. It then selects a random note from that pitch set and adds a pitch offset (MIDI note 38). In a real-time performance, the pitch could be changed by manipulating the offset to move the randomly chosen notes higher and lower through the pitch register. Additionally, through the use of the “pick random” function, the bounds of the notes chosen from the Pentatonic pitch set could be further constrained. For example, if we wanted to choose only the 2nd, 3rd, or 4th note of the set, we could change the function to “pick random 2 to 4.” This technique enables live coders to create more variety in the resultant musical output.

In most music, melodies do not move by random intervals. If one has a large pitch set, random intervals could result in very large leaps from one pitch to the next. A more natural sounding melody can be generated by implementing a “random walk” algorithm to change subsequent pitches as shown in Figures 5a and 5b. This results in a more musical melody by constraining the interval movement between -4 and +4 of the prior pitch. This approach

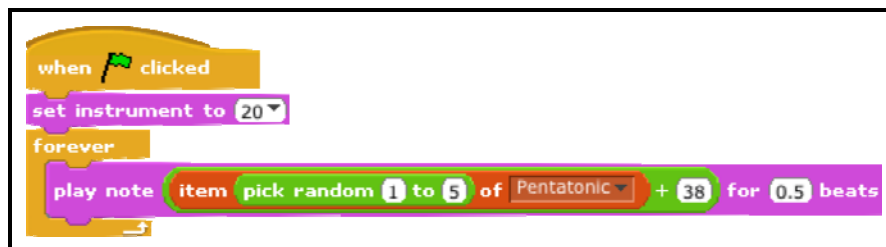


Figure 4. Code for a melody from pitch and rhythm set lists.

also enables the melody to move freely across the MIDI pitch spectrum rather than to be constrained by the length of the pitch set as was the case in the prior examples.



Figure 5a. Code for a melody via a “random walk” algorithm.

Eventually, these techniques can be expanded to model the musical styles of various composers. Our initial explorations have centered on generating music in the style of Arvo Pärt and Philip Glass. Figure 6 (one the next page) shows a small part of a larger algorithm inspired by Arvo Pärt’s *Stabat Mater* [1]. This example iterates through the AeolianPitchSet list (organized as a descending minor scale) to select pitches and then chooses random rhythm values from the RhythmSet list. The values of the RhythmSet list were derived from an aural analysis of the *Stabat Mater* and weight the probability of selecting a whole note duration twice that of selecting a half note duration. In the full algorithm, this code is duplicated twice to create three multithreaded musical parts that are triggered via keystrokes. The addition of human control to starting and stopping threads enables the performer to create dynamic variations in musical form and texture by starting and stopping sections of the overall code.



Figure 5b. Interval list for use with the “random walk” algorithm.

In addition to the live manipulation of lists, variables, and offsets shown in prior examples, Figure 6 also enables the selection of multiple pitch set lists, modification of the direction in which the list is iterated, and the ability to choose randomly or to isolate and repeat pitch values. Additional functions from the Scratch Sound and Numbers menus can be added, removed, and manipulated in real time to generate more musical control and expression. For example, a “change volume by x” function could be added to create changes in musical dynamics or to realize dynamic fading in or fading out of sections of the code. Additionally, a “change tempo by x” function could be inserted at various points to slow down or speed up the tempo. This could be set to a discrete value or by a mathematical function as shown in Figure 7.

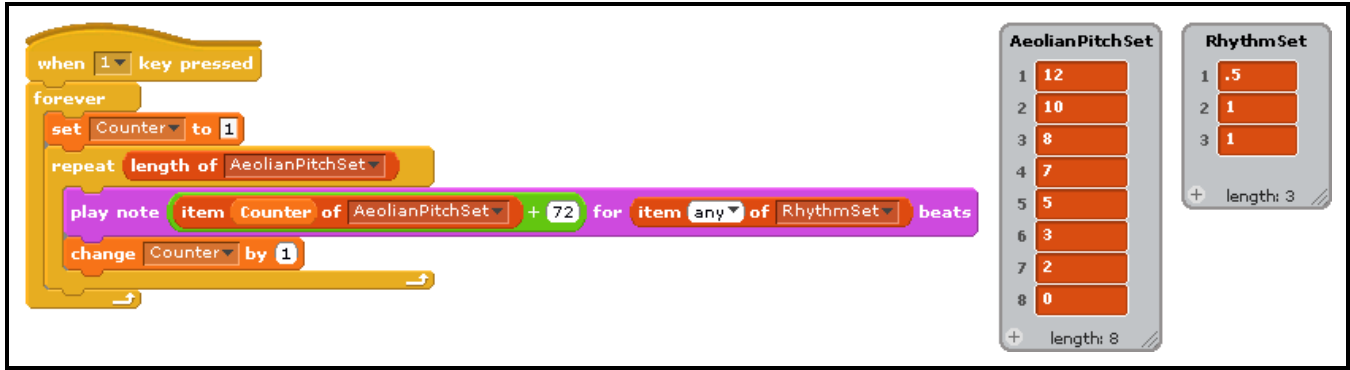


Figure 6. Melodic code inspired by Arvo Pärt's *Stabat Mater*.



Figure 7. Examples of built-in Scratch functions well suited to musical live coding.

Performing effective real-time manipulation of code (musical live coding) to create and shape generated music requires both musical and computational understanding. From a musical perspective, one needs to understand how the ongoing, generative music should sound. From a computational perspective, one needs to understand how the code can be adjusted and manipulated in real time to achieve the aural and musical changes and outcomes one desires. These are advanced skills, but they can be learned through experimentation and exploration that is both educational and fun. Scratch provides a unique, easy-to-learn platform that enables musical live coding by allowing nearly all aspects of the code to be adjusted in real time. Students can then share and showcase their work in live or pre-coded performances, which is the essence of Performamatics.

## 6. ADDING A TANGIBLE INTERFACE

As the course develops, we plan to integrate tangible computing using PicoBoards [10] (Figure 8). Using these devices' live sensing capabilities, we can implement gestural musical input and design new instruments to perform musical algorithms implemented in Scratch.

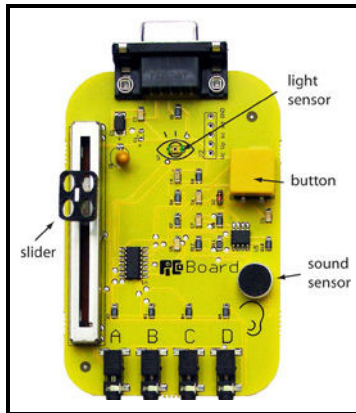


Figure 8. The PicoBoard [10].

Figure 9 shows a simple program that converts a PicoBoard into a musical instrument. A “forever” loop is used to enable continuous live sensing of the button and slider sensors on the PicoBoard. If the sensor button is pressed, the slider sensor value

is read and a note is played whose pitch corresponds to the sensor's current value. Computational thinking comes into play because the PicoBoard's slider sensor returns values between 0 and 100. To convert those values to a 7-note whole tone musical scale in which each interval is two equal half-steps apart, the value returned by the slider is combined with a copy of itself on which a modulus 2 operation has been applied. This ensures that when the slider is moved, the pitch of the note being played always jumps by a whole step rather than a half step. With the beat value set to 0.01, a continuous stream of pitches sounds when the PicoBoard button is pressed. The result is a surprisingly expressive instrument with which the user can establish a rhythm through interaction with the button sensor and play gestural pitch sweeps through manipulation of the PicoBoard slider.

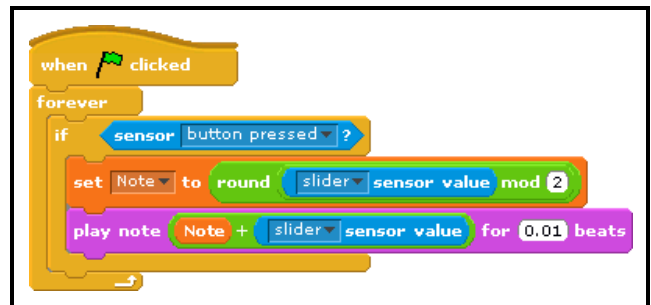


Figure 9. Code for a simple PicoBoard musical instrument.

The above example only takes advantage of two of the eight possible sensor inputs on the PicoBoard. More complex interface configurations and Scratch code are currently in development that will enable more interesting musical performances and live coding demonstrations of computational thinking.

The integration of PicoBoards as an interface for tangible computing enables discussion of CS hardware concepts such as:

- What is a device?
- What is a sensor?
- What is a signal, and how is it detected in software?
- What is an event, and how is it detected in software?
- What different types of events are triggered by various devices (real and virtual)?

Integrating physical computing with Scratch's graphical coding environment provides a unique platform for expressive computing

in real time. Programs can not only be written to create music, but they can be written to model musical environments that are performed through musical live coding or the design and interfacing of tangible computing devices such as the PicoBoard.

## 7. INTERDISCIPLINARY BENEFITS

After taking *Sound Thinking* (Version 1) in the Spring 2009 semester, Music major Charles Saulters developed a strong interest in using computational thinking as a means of developing more expressive gestural music controllers. He pursued a Research Experience for Undergraduates with us over the summer, exploring ways to apply these Performamatics concepts in even more exciting ways. He describes his work as follows.

I am interested in enabling others to achieve more than they ever thought possible through the use of computational thinking in real world situations that are relevant and interesting to students. One “hook” that I found particularly interesting is the manipulation of virtual instruments, composing for and performing using nontraditional devices such as the iPod Touch. Now more than ever, we musicians find ourselves in an age where technologically almost anything is possible. It is therefore crucial that we understand what makes computers function and acquire a strong working knowledge of programs and the coding behind them.

Interdisciplinary collaboration helps cultivate new and exciting innovations that can bring about the revitalization of CS education for which Performamatics was conceived. Using music as a hook, we can create innovative live performances and interesting visuals in conjunction with “musical live coding” to tap the imagination of people who might never have considered CS as a possible major. People (like myself) tend to be intimidated by the mystifying technical jargon. However, with more exposure to interesting multi-disciplinary projects, students can start thinking computationally and actively using that new way of thinking in a hands-on way without even realizing they are doing so. At that point, the fear is gone.

Devices such as the iPod Touch and iPhone are ideal tools for exploring computational thinking. They are easy to use, have simple, intuitive user interfaces, and have a wide range of functionality: file transfer, web browsing, MIDI control through accelerometers, light sensors, microphones, and touch sensors.

While no Scratch interface to iPhones or iPods yet exists, these sensor-rich input devices have tremendous potential as expressive interfaces to musical live coding and performance. We see our work in integrating PicoBoards into *Sound Thinking* Version 2 as an initial step in providing these benefits.

## 8. ACKNOWLEDGMENTS

The work described in this paper is based upon work supported by the National Science Foundation under Grant No. 0722161, “CPATH CB: Performamatics: Connecting Computer Science to the Performing, Fine, and Design Arts” and a complementary Research Experience for Undergraduates (REU) supplement.

*Principal Investigator:* Jesse M. Heines. *Co-Principal Investigators:* Fred G. Martin, Gena Greher, Jim Jeffers, and Karen Roehr. *Senior Personnel:* Sarah Kuhn and Nancy Selleck. *Student Researchers:* Paul Laidler and Charles Saulters II. Additional information on the Performamatics project can be found at [www.performamatics.org](http://www.performamatics.org).

Alex Ruthmann’s adaptation of musical live coding in Scratch is based on his collaboration with Andrew R. Brown and Andrew C. Sorensen at the Queensland University of Technology, Brisbane, Australia.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

## 9. REFERENCES

- [1] Brown, A.R. (2005). *Making Music with Java*. Brisbane, Australia: Lulu.com.
- [2] Brown, A.R., & Sorensen, A.C. (2009). *Interacting with generative music through live coding*. *Contemporary Music Review* **28**(1):17-29.
- [3] Greher, G.R., & Heines, J.M. (2008). *Connecting Computer Science and Music Students to the Benefit of Both*. Assoc. for Technology in Music Instruction (ATMI) 2008 Conf. Atlanta, GA.
- [4] Heines, J.M., Goldman, K.J., Jeffers, J., Fox, E.A., & Beck, R. (2008). *Interdisciplinary approaches to revitalizing undergraduate computing education*. *Jrnl. of Computing in Small Colleges* **23**(5):68-72.
- [5] Heines, J.M., Jeffers, J., & Kuhn, S. (2008). *Performamatics: Experiences With Connecting a Computer Science Course to a Design Arts Course*. *The Int'l. Jrnl. of Learning* **15**(2):9-16.
- [6] Heines, J.M., Greher, G.R., & Kuhn, S. (2009). *Music Performamatics: Interdisciplinary Interaction*. Proc. of the 40th ACM SIGCSE Technical Symposium on Computer Science Education, pp. 478-482. Chattanooga, TN: ACM.
- [7] Malan, D.J., & Leitner, H.H. (2007). *Scratch for budding computer scientists*. Proceedings of the 38th ACM SIGCSE Technical Symposium on Computer Science Education. Covington, Kentucky, USA: ACM.
- [8] Maloney, J., et al. (2004). *Scratch: A Sneak Preview*. Second Int'l. Conf. on Creating, Connecting and Collaborating through Computing (C5'04), pp. 104-109. Kyoto, Japan.
- [9] Martin, F., et al. (2009). *Joining Computing and the Arts at a Mid-Size University*. *Jrnl. of Computing Sciences in Colleges* **24**(6):87-94.
- [10] Playful Invention Company (2008). *PicoBoard: Connect real-world sensors to your Scratch projects*. [www.picocricket.com/picoboard.html](http://www.picocricket.com/picoboard.html) accessed Sep. 9, 2009.
- [11] Sorensen, A.C., & Brown, A.R. (2007). *aa-cell in practice: An approach to musical live coding*. Proceedings of the International Computer Music Conf. Copenhagen, Denmark.
- [12] Wang, G., & Cook, P.R. (2004). *On-the-fly programming: using code as an expressive musical instrument*. Proceedings of the 2004 Conf. on New Interfaces for Musical Expression, pp. 138-143. Hamamatsu, Shizuoka, Japan: National Univ. of Singapore.