9-2009

A music context for teaching introductory computing

Ananya Misra

Doug Blank
Bryn Mawr College, dblank@brynmawr.edu

Deepak Kumar
Bryn Mawr College, dkumar@brynmawr.edu

Citation

Ananya Misra, Douglas Blank, and Deepak Kumar. A Music Context for Teaching Introductory Computing. ACM SIGCSE Bulletin - ITiCSE '09 41.3 (September 2009): 248-252.

A Music Context for Teaching Introductory Computing

Ananya Misra
Princeton University
35 Olden St.
Princeton, NJ, USA
amisra@cs.princeton.edu

Douglas Blank
Bryn Mawr College
101 N. Merion Ave.
Bryn Mawr, PA, USA
dblank@cs.brynmawr.edu

Deepak Kumar
Bryn Mawr College
101 N. Merion Ave.
Bryn Mawr, PA, USA
dkumar@cs.brynmawr.edu

ABSTRACT

We describe `myro.chuck`, a Python module for controlling music synthesis, and its applications to teaching introductory computer science. The module was built within the Myro framework using the ChucK programming language, and was used in an introductory computer science course combining robots, graphics and music. The results supported the value of music in engaging students and broadening their view of computer science.

Categories and Subject Descriptors

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

General Terms

Design, Experimentation

Keywords

CS1, computer science, education, pedagogy, music, Python

1. INTRODUCTION

Computers and music have a long association. Years ago, Ada Lovelace had hypothesized that the Analytical Engine "might compose elaborate and scientific pieces of music of any degree of complexity or extent" [6]. Although modern computers have realized this hypothesis in many ways, this aspect of computing is often invisible to introductory students. However, given accessible tools, music can serve as a powerful means to engage students in introductory computer science courses (CS1), highlighting the creative as well as analytical sides of computing. Our goal was to introduce music as an application of computers and employ it to teach computer science concepts to new students.

Many traditional CS1 courses are taught with the programming language as the focus. We believe that students learn better when the focus is a particular context, and the programming language supports those goals. Contexts already successfully explored at Bryn Mawr College include robotics as part of the Institute for Personal Robots in Education (IPRE) initiative, and games [12, 1, 17]. In these classes, creative tasks such as making the robot dance provided the motivation to learn. Students used the Python-based Myro library (see Section 2.1) to control robots, create graphics, and design games. We wanted to retain the advantages of these tools and also present music as yet another context, taking a "the more, the merrier" perspective. Existing support for music in Myro included commands to make the robot or computer beep with specified frequencies and durations, and to write songs composed of successive beeps. However, programming music has wider scope, even at an introductory level. Thus, we sought richer music programming tools that would suit first-time programmers and complement and enhance the Myro framework.

Existing work on teaching CS1 via sound or music includes the use of high-level Java MIDI implementations [11, 5], as well as sample-level sound synthesis in the Squeak environment [8]. The Princeton Laptop Orchestra (PLOrk) teaches students to program music for live performance, exposing some computer science concepts while focusing on the end product [15]. Musical composition has also contributed to teaching more advanced topics, such as design patterns [9]. Most of the existing pedagogical tools, however, use languages other than Python; we hesitated to confuse novices by teaching in multiple languages. While introductory material based on media computing offer Python APIs to manipulate sound [7], we also hoped to focus on the experience of "creating music" rather than "processing sound".

Current Python systems for synthesizing music include an interface to the music programming language CSound [2] and SWIG bindings to the music analysis and synthesis framework MARSYAS [13]. Python scripting to control sound synthesis is also available in the PySndObj module [10]. These are excellent tools for computer musicians, but have not been designed for pedagogical purposes. CSound, for instance, includes a potentially confusing distinction between the score file and the orchestra file of a piece. PySndObj and MARSYAS offer low-level control over sound synthesis, but do not offer a wide range of readymade instruments to play, as present in music programming languages or in the Synthesis ToolKit (STK)[4].

Hence, we developed a Myro module for synthesizing music, offering a simple Python interface coherent with the rest of Myro, and outsourcing the synthesis to the real-time audio programming language ChucK [14]. The module was

ITiCSE'09, July 6–9, 2009, Paris, France.

deployed in a CS1 course at Bryn Mawr College. The rest of this paper describes the system, presents some code examples and student feedback, and discusses the results.

2. THE MYRO/CHUCK SYSTEM

The components and architecture of the Myro/ChucK system are described below, followed by a brief discussion of the support it provides for teaching computer science concepts.

2.1 Myro

Myro is the name of the Python module developed by the Institute for Personal Robots in Education (IPRE). Myro was designed to allow introductory students to quickly and easily begin to explore interesting computational problems through simple, low-cost robots. Using imperative commands, students can explore programs that move a robot, read sensors, take pictures, and perform basic image processing on the resulting images. Myro supports a variety of interfaces for exploring graphics, games, artificial intelligence, and now music.

2.2 ChucK

ChucK is a specialized audio programming language that allows precise control over audio synthesis, performance and analysis, at any time granularity [14]. For our purposes, one of its key strengths is that the audio synthesis takes place in real-time. Since sound is produced while the ChucK code runs, it is also possible to modify the sound on-the-fly based on input from external devices or programs, using message passing protocols. ChucK includes a set of built-in *unit generator* objects that output different types of sounds. In particular, it incorporates many of the instruments from STK [4], providing easy access to their parameters and output. ChucK has been used extensively in PLOrk [15].

2.3 Putting them together

Combining Myro and ChucK in a student-friendly way involves several levels of translation (see Figure 1). A student's Python code invokes functions and objects from a special `myro.chuck` Python module. This module uses the Open Sound Control (OSC) protocol [16], via the `simpleOSC` API, to communicate with ChucK. Each function in the `myro.chuck` module sends out an OSC message with a specific label and value, containing all the information needed to update the sound synthesis appropriately.

When `myro.chuck` is initialized, it starts running a separate ChucK server that listens for the OSC messages. The ChucK code includes a static instance of each instrument offered in `myro.chuck`. On receiving an OSC message, the ChucK server modifies the corresponding instrument parameters accordingly. The sound produced by ChucK goes directly to the computer's speakers or headphones. Brief updates are printed to a terminal or command window every time an OSC message is received, for debugging or logging.

2.4 Teaching computer science concepts

The interface for students hides the technical details described above. From the student's perspective, sound is made by creating an instrument, connecting it to the speaker, and performing actions on it in some order. Many instrument types exist: saxophone, sitar, mandolin, voice, shakers, and more. All share common functionalities, but some can

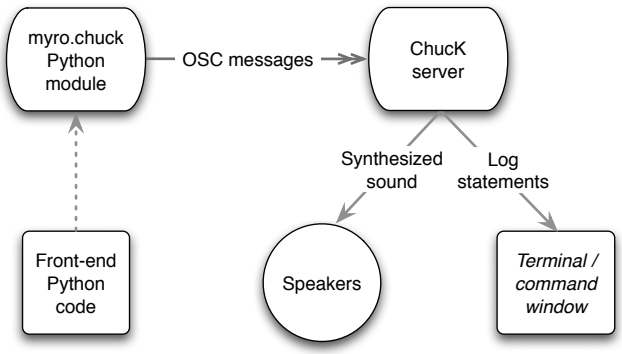

Figure 1: A diagram of the Myro/ChucK system.

be further manipulated in specific ways. Thus, programming music supports the teaching of imperative programming, introduces the notion of objects and polymorphism, and familiarizes students with Python object syntax. More abstractly, it continues the mapping between code and real-life actions present in the rest of Myro.

Parallelism is another concept well-suited to study through music. Programmatically, parallelism could be implemented via threads. However, rather than grappling with the technical details of creating and managing threads for CS1 students, we introduced a new programming construct: `doTogether(f1, f2, ...)`, where `f1`, `f2` are names of Python functions each individually controlling a single instrument. `doTogether`, provided in the Myro library, takes care of managing and executing all functions in parallel threads. This way, we have managed to introduce the idea of parallelism to CS1 students as well as the notion of using functions as first class arguments. The name "doTogether" comes from a function of the same name, but different usage, in Alice [3]. We found that students also extended their use of this construct to control graphics and robot dances in parallel with music synthesis. Myro includes several other similar programming constructs that were designed to facilitate beginners to conceptualize computational ideas in the application context without succumbing to the often "dirty" details of their underlying implementation.

One of the greatest strengths of programming music in an introductory computer science class is that it provides a broader perspective, early on, of what we can do with computers. The more possibilities offered, the better chance students have to "bond" with computing. Learning about music, robots, and graphics in the same class shows students the widespread applications of computers.

3. EXAMPLES

We shall now describe and give examples of Python code for writing music in Myro/ChucK.

3.1 The basics

To begin, the `myro.chuck` module is imported and initialized as follows:

```
from myro.chuck import *
initChuck()
```

This sets up the communication and synthesis framework

and makes the myro.chuck instruments available for use. An instrument can then be created and used like this:

```
s = SineWave()  # create a SineWave generator
s.connect()     # connect it to sound output device
wait(2)         # wait 2 seconds
s.disconnect()  # disconnect it from sound output
```

A SineWave is a simple instrument that generates a continuous tone. The above code results in a pure tone at 220 Hz., played for 2 seconds. Most other myro.chuck instruments, like physical instruments, require a little more work to produce sound. The following example sets up a saxophone and plays two notes on it, separated by half a second of silence:

```
x = Saxophone() # create a virtual Saxophone
x.connect()     # connect it to sound output
x.noteOn(1)     # start playing a note on it
wait(.5)        # wait .5 seconds
x.noteOff(1)    # stop playing the note
wait(.5)        # wait .5 seconds in silence
x.noteOn(.5)    # start playing a softer note
wait(1)         # keep playing for 1 second
x.noteOff(.5)   # stop playing the note
x.disconnect()  # disconnect from sound output
```

Some instruments, such as plucked string instruments, do not require a noteOff call as their sound automatically decays as in their physical counterparts. Still, the following operations are available for all instruments:

connect(): connect to sound output device
disconnect(): disconnect from sound output
noteOn(velocity): start playing a note
noteOff(velocity): stop playing a note
setGain(gain): set instrument gain/amplitude/loudness[1]
setFrequency(freq): set instrument frequency/pitch[2]

In addition, several instruments have more specialized operations, such as control over vibrato, the size or stiffness of the instruments, or other ways to change the timbre of the sound played. These controls passed from STK [4] to ChucK [14] to myro.chuck. Some of the specialized operations are redundant; wind instruments, for instance, have a startBlowing function, indistinguishable from the general noteOn function. This was a conscious choice, since having many ways to achieve the same goal is a common computer science scenario.

A complete list of myro.chuck instruments and functions is available at http://wiki.roboteducation.org/ChucK.

3.2 More sophisticated examples

These building blocks suffice for programming interesting melodies and rhythms. A melody for one instrument can be created via a series of setFrequency, setGain, noteOn, wait and noteOff operations. The repetition of this sequence of operations for each note motivates a discussion on organizing code into re-usable blocks. A helper function for playing a note on an instrument might look like this:

[1]Loudness is not equivalent to gain, but closely related.
[2]Pitch is not equivalent to frequency, but closely related.

```
# play a note on the given instrument, according to
# the parameters specified
def playNote(instrument, freq, gain, duration):
  instrument.setFrequency(freq)
  instrument.setGain(gain)
  instrument.noteOn(1)
  wait(duration)
  instrument.noteOff(1)
```

A larger program to play a melody on an instrument can then invoke the helper function repeatedly:

```
# play a short melody
# 1.  set up the instrument
x = Saxophone()
x.connect()
# 2.  play the tune
playNote(x, 262, 0.8, 0.5) # C
playNote(x, 330, 0.9, 0.5) # E
playNote(x, 392, 1.0, 1.0) # G
# 3.  shut down the instrument
x.disconnect()
```

In practice, students have further clarified their code by storing standard note frequencies as variables (e.g. C = 262) or defining an array of frequencies to be played in order. Some have devised more creative solutions such as defining two arrays, one with frequencies to be played sequentially and another with corresponding note durations. One student discovered dictionaries in the Python documentation and used one to associate note names with frequencies.

A further level of sophistication consists of playing multiple scores simultaneously. Below, we illustrate how two instruments, a shaker and a struck bar, can be controlled to play different scores together in parallel using the doTogether construct described in Section 2.4:

```
# helper function
def playOnce(instrument, time, strength):
  instrument.noteOn(strength)
  wait(time)

# score 1
def playShakers():
  s = Shakers()
  s.connect()
  beat = 0.4
  for i in range(5):
    playOnce(s, beat, 1)

# score 2
def playBar():
  b = StruckBar()
  b.connect()
  beat = 0.4
  for i in range(5):
    playOnce(b, beat/2, .8)
    playOnce(b, beat/2, 1)

# play both scores together
doTogether(playShakers, playBar)
```

Playing the functions in parallel via doTogether is similar

to a band's performing a piece together, with each musician playing her own part. While this is a simple example to demonstrate the concept, students have used similar constructs to create more complex and melodious pieces.

4. EVALUATION

4.1 Usage

The `myro.chuck` framework was used in a 23-student introductory class with a combination of would-be majors, non-majors, and undecided majors. The first half of the course focused on robots; music was introduced in the second half, and graphics towards the end. Students programmed two assignments using `myro.chuck`, although many also chose to integrate it into their final projects.

The first music assignment asked students to turn their robot into a real-time sound controller by mapping its sensor data to musical instrument parameters. They chose the specific sensors, instruments, and mappings to use. This served as a transition from robots to music, and also offered an alternate perspective on both. The second assignment supported more standard musical orchestration, asking students to use `doTogether` to create music with two or more instruments. The focus lay on ensuring that they understood the programming constructs, but it also led to complex pieces and further exploration of Python by several students (`http://cs.brynmawr.edu/music/`).

A number of students also chose to use `myro.chuck` for their final project (which was to do something interesting with robots, graphics, and/or music). Memorable projects involving `myro.chuck` include:

- A gamepad-controlled jukebox playing a selection of programmed songs,
- A synchronized song-and-dance sequence in which a robot danced in time to synthesized music,
- Several projects combining graphics/animation and music synthesis.

4.2 Student feedback

At the end of the course, student feedback was gathered from written answers to a question on what they learned about computing and what role robots, music and graphics played in the process. The responses supported the inclusion of music in several ways, summarized below.

Feedback referring solely to the music component included statements of surprise, such as, "The music was something I never really associated with computing, but it was something interesting and new." A student who had played instruments from an early age wrote, "The intricacies behind creating music in Myro surprised and challenged me. That was most instrumental in my growth of Python and computing knowledge." Another, who had difficulties with her robot, mentioned, "Music definitely rekindled my interest." Some were inspired by a particular aspect of music. One student described an interest in algorithmic composition combining "the deterministic calculability of computer code" with "random variation". Another wrote, "The most interesting part of the course was working with music and computers. This led to a revelation of the scope of possibilities that computer science provides. Before this, I never would have considered the possibility that robots and computers can work together to form their very own orchestra." Thus, music synthesis made the course more appealing to certain students.

Other comments referred to combinations of the different course components: "I was genuinely pleasantly surprised that 'computing' did not apply exclusively to the calculation of numbers. It encompasses a wide range of things." Some students enjoyed both music and graphics because "they clearly tied computers to art and then the computer just became another medium to express myself," or because "I could really see myself using computer science in a real world setting." Others were amazed that "one language, Python, could be used to program so many different things," or further observed that "even though the end products can be very different, the basic concepts are similar."

It was also interesting to note what students had learned about computing in general, from the entire course. Several described it as "a problem-solving process" or "logic puzzles". Some shared their insights into the programming process, such as "how programming can build on itself—starting from extremely simple code and leading to something very intricate and complex." One student was "always amazed by the dozens of different ways that students in the class solved the same task. I was surprised to find that the working process was less mathematical and precise than I had expected it to be—it's analytical but also requires a degree of creativity." Others also mentioned "the enormous amount of creativity in designing programs." As one wrote, "I've always thought that I was bad at math and good at the arts and humanities because my mind was better at dealing with abstract questions and ideas than dealing with concrete rules, but I'm starting to wonder if the two are really all that different."

Not all the feedback was as encouraging. One student, for instance, wrote, "The music taught me to respect electronic-based bands who work with that sort of thing for a living, and the graphics baffled me entirely. As to how this course may impact my future studies, I will leave you with a quote from that recalled Barbie: 'Math is hard!'" However, the overall reaction to the course seemed more in line with the following comment: "I also learned that computer science and technology are more accessible than I realized. They don't always have to be daunting, boring material reserved for physicists or engineers."

5. DISCUSSION AND CONCLUSIONS

The student feedback suggests that teaching CS1 with `myro.chuck` was overall promising. 4 of the 23 students enrolled in CS2 the following semester, comparable to 6 of 22 students from a parallel CS1 section not focusing on music. The small numbers of students make it difficult to obtain reliable statistics; we would like to perform better assessment with a more targeted survey instrument and/or a longitudinal study in the future. The open-ended remarks indicate that the music module especially engaged students who already had musical interests, and broadened others' ideas of what computer science entails. It exposed creative aspects of computing but also retained the logic needed to design and implement a program that behaves as desired. The use of music in conjunction with robots and graphics was especially valuable, as it led to a more well-rounded understanding of computer science and of the underlying concepts that serve as a common factor between these diverse areas.

Specific aspects of `myro.chuck` that worked include the use of an API similar to the rest of Myro. This helped

students transition between the different components, and highlighted that the computing behind these components was essentially the same. Having access to a range of musical instruments to synthesize made the material more interesting, as well as providing an opportunity to discuss object-oriented programming concepts. The `doTogether` programming construct was also popular; students used it to translate musical scores into programs, and to bring together different components of the course in their final projects. Although the details of threads and parallelism were not discussed, we hope a familiarity with using these concepts will help students understand how they work in later classes.

Interestingly, students seemed more excited to reproduce known music than to create original music. This is, of course, acceptable in an introductory computer science class, and may have resulted from the lack of in-class focus on composing as well as from students' not seeing themselves as "composers". A related observation is that some instrument-specific operations, such as setting vibrato or stiffness, were seldom used. Future courses may benefit from more attention to these aspects, as they could lead students to a new level of exploration and discovery. A "robot orchestra" (see Section 4.2) is also an exciting topic to investigate, through a series of group projects or even an entire course.

Some improvements can be made to the system. A drawback of the current version is that it allows only one instance of each instrument type. Students have at times wanted to create multiple instances of the same instrument, which makes sense from an object-oriented programming perspective. This limitation can be removed by running a set of more complex ChucK programs in the background, instead of the current relatively simple one. Another area for improvement is to offer more sophisticated control over the synthesized sound. The current options of synthesizing different instruments and manipulating their parameters scratch the surface of computer music. ChucK also offers many audio processing modules and effects, such as reverberation, echo, and customizable digital filters. A more rounded treatment of making music with computers would include these frequently used tools. Of course, doing them justice would also require spending more time on the music unit.

`Myro.chuck` may be adapted for any CS1 class. A course without robots or graphics may use it to teach imperative programming, functions, syntax, and logic, via tasks such as exploring particular instrument parameters, programming a pre-defined score, or generating music algorithmically. Thus, it serves as a useful tool for a section on programming music. Because it is integrated in Myro, it is easy to use in conjunction with other Myro modules, resulting in an even more richly engaging introduction to computer science.

6. ACKNOWLEDGMENTS

We would like to thank Perry Cook and Melissa Lawson at Princeton, and Dianna Xu at Bryn Mawr, for their support. We would also like to thank Microsoft Research for initial funding of the Institute for Personal Robots in Education.

7. REFERENCES

[1] D. Blank. Robots make computer science personal. *Communications of the ACM*, 49(12):25–27, December 2006.

[2] R. Boulanger. *The Csound Book: Perspectives in Software Synthesis, Sound Design, Signal Processing, and Programming*. MIT Press, Cambridge, Massachusetts, 2000.

[3] M. Conway, S. Audia, T. Burnette, D. Cosgrove, K. Christiansen, R. Deline, J. Durbin, R. Gossweiler, S. Kogi, C. Long, B. Mallory, S. Miale, K. Monkaitis, J. Patten, J. Pierce, J. Schochet, D. Staak, B. Stearns, R. Stoakley, C. Sturgill, J. Viega, J. White, G. Williams, and R. Pausch. Alice: Lessons Learned from Building a 3D System for Novices. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 486–493, 2000.

[4] P. R. Cook and G. Scavone. The Synthesis ToolKit (STK). In *Proceedings of the 1999 International Computer Music Conference*, October 1999.

[5] T. J. Cortina. Using the Java MIDI package to create music in CS1. *Journal of Computing Sciences in Colleges*, 21(2):86–88, 2005.

[6] J. Fuegi and J. Francis. Lovelace & Babbage and the Creation of the 1843 'Notes'. *IEEE Annals of the History of Computing*, 25(4):16–26, October 2003.

[7] M. Guzdial. A media computation course for non-majors. In *Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education*, pages 104–108, 2003.

[8] M. Guzdial and E. Soloway. Teaching the Nintendo generation to program. *Communications of the ACM*, 45(4):17–21, April 2002.

[9] J. Hamer. An approach to teaching design patterns using musical composition. *ACM SIGCSE Bulletin*, 36(3):156–160, 2004.

[10] V. Lazzarini. Musical signal scripting with PySndObj. In *Proceedings of the 5th International Linux Audio Conference*, 2007.

[11] M. P. Rogers. Making Music in CS I. *The Journal of the Consortium of Computer Sciences in Colleges*, 20(1), October 2004.

[12] J. Summet, D. Kumar, K. O'Hara, D. Walker, L. Ni, D. Blank, and T. Balch. Personalizing CS1 with Robots. *Proceedings of SIGCSE 2009 Conference, Chattanooga, TN*, 2009.

[13] G. Tzanetakis, R. Jones, C. Castillo, L. G. Martins, L. F. Teixeira, and M. Lagrange. Interoperability and the Marsyas 0.2 Runtime. In *Proceedings of the 2008 International Computer Music Conference*, August 2008.

[14] G. Wang and P. R. Cook. ChucK: A concurrent, on-the-fly, audio programming language. In *Proceedings of the 2003 International Computer Music Conference*, September 2003.

[15] G. Wang, D. Trueman, S. Smallwood, and P. R. Cook. The laptop orchestra as classroom. *Computer Music Journal*, 32(1):26–37, 2008.

[16] M. Wright and A. Freed. Open sound control: A new protocol for communicating with sound synthesizers. In *Proceedings of the 1997 International Computer Music Conference*, pages 101–104, September 1997.

[17] D. Xu, D. Blank, and D. Kumar. Games, Robots and Robot Games: Complementary Contexts for Introductory Computing Education. In *Third Annual Microsoft Academic Days Conference on Game Development in Computer Science Education*, 2008.