An Online Interactive Course on Computer Music *

Roger B. Dannenberg
Carnegie Mellon University
rbd@cs.cmu.edu

Jesse Stiles
Carnegie Mellon University
jessestiles@cmu.edu

Yuezhang Li
Peking University
lyzmusic@pku.edu.cn

Qiao Zhang
Tsinghua University
qiaozhang0429@gmail.com

ABSTRACT

We have developed a highly interactive, online course to teach Introduction to Computer Music. The course is intended for students with a basic knowledge of programming and teaches principles and techniques of sound synthesis and algorithmic composition using a hands-on, project-oriented approach. The course contains an extensive set of short video lectures complemented by about 80 interactive, web-based exercises. Data indicates that students are able to complete the online exercises on their own and that students are highly motivated to perfect their online solutions, which offer immediate feedback through automatic grading. The online course is complemented by in-class music listening and discussion, and our intention is to offer a completely online version of the course in the near future.

1. INTRODUCTION

Computer Music is an increasingly popular subject among college engineering and science students. Students are attracted by an almost universal interest in music, technical fascination with computing and signal processing, popular applications for novice music making, and the increasing use of laptops and electronics in popular music making. Many students are excited by the chance to engage in a more formal study of Computer Music.

At Carnegie Mellon, enrollment in our Introduction to Computer Music class has reached nearly 100 students, and a new concentration area in Integrative Design, Arts, and Technology (IDeATe) promises to attract even more students. In addition, the IDeATe curriculum needs a course offering every year rather then every other year as in the past. With limited staff to teach growing numbers of students more frequently, we have turned toward a technological solution: creating an online course on Computer Music.

The new course combines online lectures, numerous interactive exercises that are graded automatically, and a set of open-ended projects with classroom discussion, listening and analysis sessions, and some manual assessment. Our goal is to offer the course remotely in the future to interested students and to make the material available to other instructors.

In the following section, we describe the prerequisites and learning objectives and content for the class. In Section 3, we describe the technological components including interactive exercise generation and grading. In Section 4, we present our current structure for delivering the course. Then, we describe some preliminary results and experience in Section 5. In Section 6, we describe future work and present conclusions.

2. COURSE PREREQUISITES, LEARNING OBJECTIVES AND CONTENT

Our Introduction to Computer Music is offered by the Computer Science Department and is taught as a computer science subject. The goals of the course include learning

- "what every computer scientist should know" about audio signal representation and sampling theory,
- how unit generators are combined to create and control musical sound,
- basic synthesis methods including AM, FM, granular synthesis, voice synthesis and subtractive models, physical models, and spectral processing,
- algorithmic composition techniques including random generation of events, control strategies, score representation and manipulation, and pattern-based parameter generation.

In addition to these goals, we try hard to encourage students to be more open-minded about sound and music, exposing them to new ways of thinking about and listening to music. Contemporary music practice might be well understood or at least familiar to any musician or composer attending an International Computer Music Conference, but is often quite foreign to a typical engineering and science student, even those with some traditional musical training. Since one of the attractions of computation in music is the ability to explore new sonic worlds, we feel it is important to point out that these worlds exist. Without this frame of mind, the techniques that we teach are difficult to motivate

*Published as: Roger B. Dannenberg, Jesse Stiles, Yuezhang Li, and Qiao Zhang, "An Online Interactive Course on Computer Music," in *Proceedings of Looking Back, Looking Forward: 41st International Computer Music Conference* Denton, Texas USA, September 2015, pp. 28-33.

and even confusing to someone who only knows popular music.

2.1 Programming, Not Music, as Prerequisite

Because we offer Introduction to Computer Music as a computer science subject, we assume that students know at least one programming language at the level of an introductory programming course. Most of our students have had at least a second semester learning about computer science, programming, and algorithms.

On the other hand, we do not assume any music theory or other music prerequisite, which would be useful but impractical in our case. As mentioned above, we devote considerable time to introduce students to works of contemporary computer music under the assumption that the contemporary music vocabulary will be quite foreign.

2.2 The Nyquist Programming Language

The course is based on the sound synthesis and music composition language Nyquist. There are of course many alternatives and certainly more popular languages and systems for work in computer music, so we offer our perspective and rationale for Nyquist.

Nyquist is a text-based algorithmic language. Students who have been programming in Python, C, or Java should find much of the syntax and semantics at least recognizable. Nyquist also offers a full-featured language and environment where motivated students can create complex programs without limitation. Nyquist is free, open-source, cross-platform, and includes an interactive development environment.

A major advantage of Nyquist for our online course is that we can easily run Nyquist in a web server to evaluate student exercises. When running on the web server, we have the option of disabling audio output or even capturing intended output for analysis and grading purposes. We can also place limits on file access, memory usage, and run time to prevent "rogue" or malicious code submissions from hogging resources or even compromising the server's files and software.

An alternative to Nyquist might be a visual programming language, i.e. Max [4] or Pd [9]. These are popular platforms, especially for real-time sound processing. For musicians and artists lacking a background in computer science the graphical environment presents a paradigm that may be more readable and intuitive than a text-based environment. To develop an online course with automatic grading for these platforms, however, would be an enormous undertaking. Furthermore, working in a visual programming language would be a poor choice for computer science students who wish to advance their text-based programming skills.

Supercollider [7], a text-based platform that is partly influenced by Nyquist, is another popular language for work in computer music. It would be a good choice for us, but we think Nyquist has a simpler and easier-to-learn syntax. ChucK, another text-based language emphasizes real-time programming, which is a strong point, but does so at the expense of providing almost no support for data structures [12]. In contrast, Nyquist is a more open-ended and general-purpose language (it has an entire Lisp programming environment as a subset), but has minimal support for real-time interaction.

Nyquist was originally designed to bring the concepts of Music *N* languages (unit generators, scores, orchestras, instantiation of instruments at given times and durations) into a more general programming language framework. Since then, csound has introduced many linguistic improvements such as expression parsing and function definition, but it is still a long way from offering a general programming environment where the user can explore algorithmic composition, score generation, synthesis algorithms, and manipulating sound files all within one language.

Any choice of language is going to be new to most students and will have to be taught. We use Nyquist to teach general principles such as music representations, algorithms, signal processing, and unit generators in the hope that these principles can be applied later, whether the student is writing low-level C code in an embedded system or working with some other high-level music system.

2.3 Course Content

The course covers the subjects listed in Table 1, devoting roughly one week to each unit. The subjects are approached through a combination of theory and practice. On the theory side, we introduce concepts, explaining for example the basics of sampling theory or the model of unit generators. On the practical side, students implement and apply synthesis models and algorithmic composition techniques using Nyquist, producing short but musical compositions.

Unit 1: Introduction and Nyquist
Unit 2: Unit Generators, Score Processing
Unit 3: Sampling Theory, Amplitude Modulation
Unit 4: Frequency Modulation
Unit 5: Algorithmic Composition
Unit 6: Granular Synthesis
Unit 7: Sampling and Filters
Unit 8: FFT and Spectral Processing
Unit 9: Source-Filter Models and Voice Synthesis
Unit 10: Audio Perception, Effects, Reverberation
Unit 11: Physical Models
Unit 12: Spectral Models, Spatial Sound
Unit 13: Audio Compression
Unit 14: Music Understanding

Table 1. Overview of the content of Introduction to Computer Music.

We currently use Roads' *The Computer Music Tutorial* [10] for material on sound synthesis and other topics, and Simoni and Dannenberg's *Algorithmic Composition* [11] for its material on Nyquist and algorithmic composition. Students are also directed to the Nyquist Reference Manual [5], over 200 pages of reference material on Nyquist.

3. ONLINE INSTRUCTIONAL TECHNOLOGY

The online component of the course consists mainly of video lectures and interactive exercises. The video lectures are based on slides used previously in a more typical classroom lecture setting, but the lectures are mostly divided into short presentations lasting 5 to 10 minutes. At the conclusion of each lecture, the student is asked to demonstrate understanding through programming exercises or multiple-choice questions. These must receive a passing grade before the next lecture becomes accessible.

3.1 Programming Exercises

We extended ATutor [1], a learning management system (LMS), with the ability to present and automatically grade Nyquist-based exercises. We will describe how the exercises are implemented because they are the most innovative aspect of the course technology. Then we will describe how ATutor is used. The implementation of exercises consists of question generation, code execution and question grading. These three aspects are described in sequence.

3.2 Question Generation

Developing interactive exercises (typically small programming problems) requires a significant effort. Once problems are presented to students, it seems inevitable that some problems and answers will be copied and shared. We want to discourage this kind of cheating, so we *generate* questions, or at least the details, resulting in dozens of potential questions. Even if an answer is published, a student hoping to simply copy the answer will find it necessary instead to at least understand and adapt the answer by changing variable names, function names, and parameter values.

Questions are designed by filling in an online form available only to ATutor "instructors." The form has a field for instructions (the problem statement), and a field for "starter code" – typically a partial solution containing Nyquist code that the student should modify or complete. These fields can use parameters denoted by `%1`, `%2`, etc. which are replaced by randomly selected values. Values for these parameters are computed or selected according to expressions that are entered into additional fields of the online form. A small library is included to perform common functions such as selecting a value from a list, generating a random value within some range and selecting a random pitch.

3.3 Code Execution

When presented with an exercise, students can edit Nyquist code directly in a browser form. The code can be evaluated by clicking an "Evaluate" button. This sends the code to the server, which copies the code into a file and interprets the code with Nyquist. The text output of Nyquist is captured and returned to the browser. If Nyquist generates a sound, the sound is stored in a file, and the result page is created with a sound file player object to play the sound.

Nyquist is a general purpose programming language that is perfectly capable of erasing files, entering an infinite loop, or launching malicious attacks on the server, so we must be careful to place limits on what student code can do while running on the server. We have extended Nyquist with a number of security features as follows:

- File access: Nyquist can be given a set of paths from which all file reads must take place (typically restricted to Nyquist libraries and a per-account directory for student-submitted code),
- File writes: Nyquist can be given a set of paths where file writes may occur (typically the per-account directory only),
- CPU time: Nyquist will terminate if the interpreter runs too long. The interpreter normally polls for interrupt characters every few milliseconds, so we count the number of times the polling routine is called, and if the number is exceeded, the Nyquist process exits,
- Memory usage: The Nyquist garbage collector is instrumented to keep track of memory usage. If the usage exceeds a limit, the process exits.

Nyquist is run in the web server using these security features to limit the damage that can be done by a programming error or a malicious attack. Students can overwrite their own exercises but cannot affect other users or hog resources to any great extent.

3.4 Question Grading

In addition to the "Evaluate" button, which simply runs Nyquist code in the web server and returns results, the student can use a "Submit" button, which directs the server to automatically grade the exercise. Grading is done by Nyquist programs using a simple protocol to communicate with the learning management system (LMS).

To grade an exercise, the system creates two files. One contains the code submitted by the student. The other contains the parameters that were used to generate the exercise as described in Section 3.2. Each exercise has a name (e.g. "waveform"). The LMS uses the name to generate a grading program name (e.g. `waveform-grade.sal`), which is executed. The grading program typically reads the parameters and constructs a correct solution. The student submis-

sion is then read, stripping out comments and looking for expected code patterns. The student submission can also be run, sometimes after redefining key functions to allow the grader to capture parameters, generated sounds, and other information, which are then compared to expected values from the correct solution.

After grading the student submission, the grader program writes a score and text feedback to a file and exits Nyquist. The LMS then reads the score and text feedback, storing the information into a database and also constructing a results page to be delivered to and displayed on the student's browser.

4. COURSE DELIVERY

4.1 Online Instruction

In our current instance of the course, the main technical content is delivered through our online system. There are 81 lecture videos with a mean length of 11 minutes and 97 interactive problems. Figure 1 shows a typical lecture, which includes slides, video of the lecturer, and sound examples, and screen capture from the Nyquist IDE.

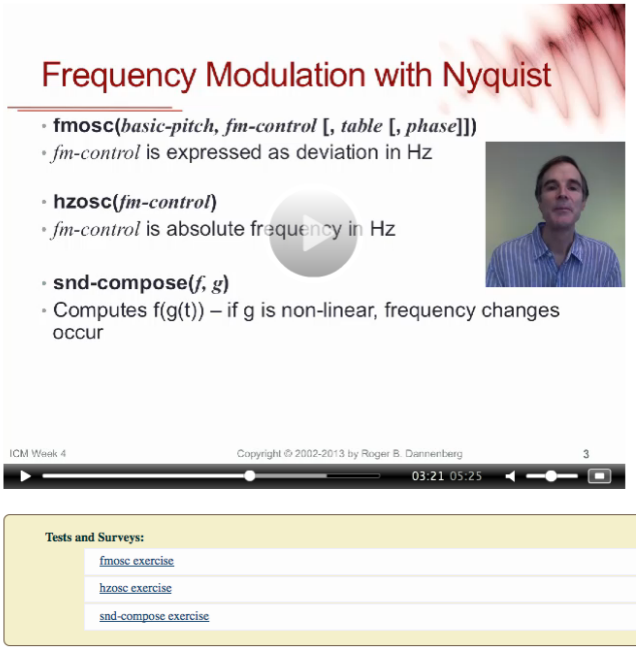

Figure 1. A portion of a browser window illustrating a typical lecture video and links (under "Tests and Surveys") to exercises.

The interactive problems include 23 quizzes containing multiple-choice questions covering more theoretical concepts such as sampling theory or music perception. The other problems consist of 74 short programming exercises using Nyquist. Exercises are designed to reinforce lecture content and to create a more active learning experience for students. To make sure students actually take time to do the exercises, successfully passing an exercise following a lecture video is a prerequisite to viewing the next lecture video. To avoid students getting "stuck" on one exercise, all exercises are designed so that minimal work is required to get a passing grade. Typically, all that is needed to pass is a program that runs to completion and that includes some expected code. For example, if the instructions say to use the `fmosc` function, the student need only write a program that calls `fmosc` in order to pass. However, students can submit exercises multiple times to improve their scores, and most students will work on exercises until they have close to perfect grades.

Figure 2 illustrates an interactive exercise programmed in Nyquist in the browser. The text box in the middle of the figure is used to create a solution. After entering Nyquist code, the student can click on the "Evaluate" button to run the code. The program is run on the web server, compiler and run-time output are displayed on the web page, and if the program produces audio, an audio player object is displayed as well, allowing students to listen to the audio output of their programs.

The student may also click a "Submit" button to grade the program. The results of a submission are shown in Figure 2. Here, the student has entered a valid program, but the modulation signal, `lfo(5)`, has an amplitude of 1 rather than the specified 14 (which gives a 14 Hz depth of modulation.) The feedback gives hints to fix the problem. The score of 70%, while sufficient to allow the student to move on, provides some incentive to find and fix the problem.

Students can get help from their peers and from course staff through Piazza [8], an online forum supporting university instruction.

4.2 Classroom and "Off-Line" Instruction

We meet our students in a traditional classroom once per week (vs. the normal 2 or 3 classes per week). We use these sessions to:

- preview upcoming topics and projects,
- answer general questions,
- listen to music, discussing the technical approaches, but perhaps more importantly discussing the esthetics, approaches to composition, and links between recent popular music and historical masterworks of computer music.

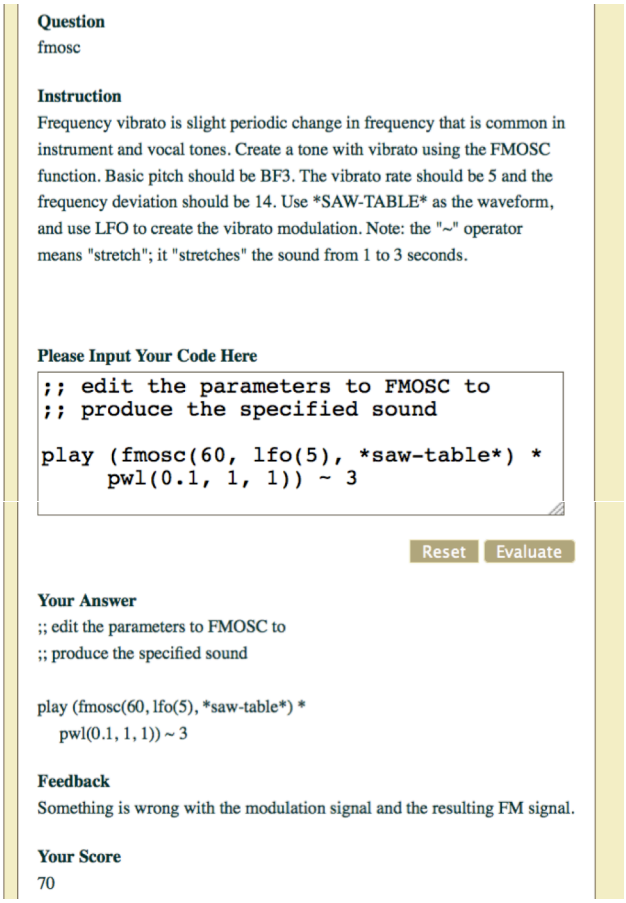

Figure 3. An online exercise to create vibrato using frequency modulation, programmed in Nyquist.

- Students are also assigned projects that take more time than the online programming exercises. These projects are implemented using the Nyquist IDE running locally on students' personal computers. Currently, these projects are graded by teaching assistants because they include a short composition and some creative programming that cannot be reduced (in their current form) to a simple checklist of attributes or a single right answer for automatic grading.

5. EXPERIENCE AND RESULTS

In the spring semester of 2015, 70 students completed the full semester and started work on a quiz or programming exercise nearly 20,000 times. Since multiple-choice questions are not novel, we will consider only the online programming exercises, of which 16,358 were started. Of those, 11,248 were submitted for grading and feedback. The remainder (about 30%) were abandoned, possibly because the student ran out of time or simply decided to begin with a fresh copy of the exercise. One student reported they sometimes look at exercises *before* watching the lecture so they know what to take notes on in order to solve the problems more easily. In any case, these incomplete sessions are ignored in our analysis because, without any submission to the server, we have no information about what the student did or even how much time was spent.

One finding is that nearly every student completed nearly every exercise. The average rate of completion was 98.4% even though the online problems account for only 8% of the overall course grade. The average final grade on submitted exercises was 98.5 (out of 100), meaning that virtually every student is making multiple attempts if necessary to produce essentially perfect scores. This indicates that

- Exercises can be completed by every student (perhaps after seeking help),
- Students are motivated to resolve all problems reported by the automated grading software.

One could imagine that the exercises are trivial and students nearly always get grades of 100%. To test this hypothesis, we can compare grades from initial submissions to the best grades achieved on each exercise. The mean *initial* grade taken over all students and all exercises (not including cases where students did not submit anything) was 82.7, significantly below the final mean of 98.5. Figure 3 shows the distribution of initial scores in white and final scores in gray.

The mean number of submissions per exercise was 2.2 with a maximum of 54. The mean time spent per exercise is difficult to estimate because it appears that sometimes students returned to "abandoned" exercises and completed them. Perhaps some students also started exercises to read the problems and then watched the corresponding lecture, thus increasing the apparent time on task for the exercise. In

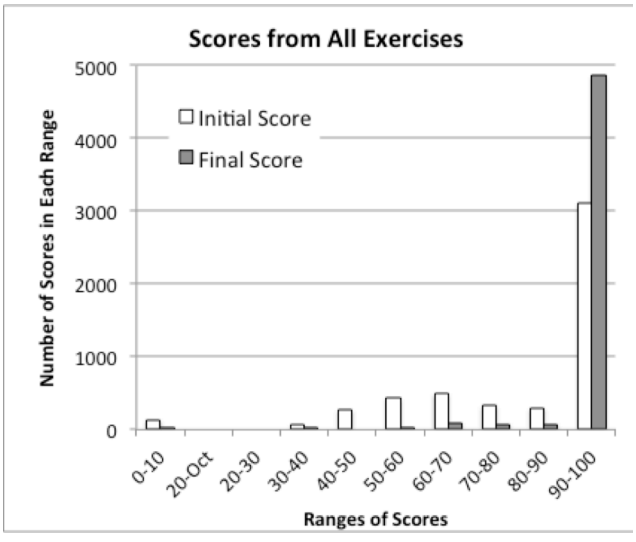

Figure 2. Histogram of scores. About 40% of all 5098 initial submissions scored below 90%, while less than 5% of all final submissions scored below 90%.

any case, measured times between starting an exercise and submitting a solution range from seconds to weeks. Considering only times of 10 minutes or less, the mean time per submission was 127s, and the mean time per exercise, allowing for multiple submissions, was 244s. The mean total time spent on all 74 programming exercises was 4.9 hours, not counting lecture viewing time (about 15 hours) or submissions that took longer than 10 minutes. Overall, it seems that the problems are perhaps easy, but not trivial, and students working online are able to master the material and solve the problems.

It should also be mentioned that grading over 11,000 exercises in one semester would be well beyond the capabilities of our teaching staff. The evidence indicates that the short exercises and rapid feedback help to engage students and help them to learn step-by-step as content is introduced through short lectures.

We expected that an end-of-semester evaluation would show some differences between the online version of the course in 2015 and the previous standard course offering in 2013. However, the overall course evaluation was not significantly different. Student comments pointed to the lack of coordinated reading material to augment the online lectures, and we are working in this direction.

6. FUTURE WORK AND CONCLUSIONS

Our goal is to offer Introduction to Computer Music as a free and open course. The online system is almost ready for wider use, but we would like to augment the material with open, online readings, especially for students who cannot visit an instructor at office hours for help. The online books *Music and Computers* [3] and *Linking Art, Science, and Practice through Digital Sound* [2] are excellent and highly compatible resources that might be utilized.

Another issue is giving students feedback on more open-ended projects. One approach is peer grading, where students assess the works of other students [6]. Another possibility is to automate grading of these more open-ended creative projects. While a reliable measure of "musicality" or "creativity" is very ambitious, a near-term goal might be to evaluate submissions with the same reliability as student teaching assistants. Features that reflect code complexity, use of appropriate functions, spectral richness and balance, rhythmic variety, clipping, and task-specific attributes might be enough to generate credible feedback. Further design is needed to offer projects with feedback in a remote version of the curriculum.

We also want to encourage listening and discussion of computer music. We publish a playlist with most of the pieces we listen to in class using the Google Play Music service.

In conclusion, we have developed an online course that teaches "Introduction to Computer Music." The course is designed for science and engineering students, or at least for students with a basic programming background. The course offers highly interactive instruction and exercises emphasizing a hands-on, computing-intensive curriculum. We plan to make the course available outside of our university in the near future and welcome collaborators who might want to integrate our material into their courses.

Acknowledgments

The authors would like to thank the School of Computer Science and the Carnegie Mellon IDeATe network for their support.

7. REFERENCES

[1] ATutor, *ATutor Handbook*, http://help.atutor.ca, 2015.

[2] J. Burg, J. Romney, and E. Schwartz, *Linking Science, Art and Practice through Digital Sound,* http://csweb.cs.wfu.edu/~burg/CCLI/Templates/curriculum_index.php, 2015.

[3] P. Burk, L. Polansky, D. Repetto, M. Roberts, and D. Rockmore, *Music and Computers: A Theoretical and Historical Approach*, http://music.columbia.edu/cmc/musicandcomputers, 2011.

[4] Cycling74, Inc., *Max*, https://cycling74.com/products/max/, 2015.

[5] R. Dannenberg, *Nyquist Reference Manual,* http://www.cs.cmu.edu/~rbd/doc/nyquist/, 2013.

[6] H. Luo, A. Robinson, J.-Y. Park, "Peer Grading in a MOOC: Reliability, Validity, and Perceived Effects," *Online Learning,* Vol. 18, No. 2, 2014.

[7] J. McCartney, "Rethinking the computer music language: SuperCollider," *Computer Music Journal*, Vol. 26, No. 4, 2002, pp. 61–68.

[8] Piazza Technologies, *Piazza*, http://piazz.com/signup, 2015.

[9] M. Puckette, "Pure Data," *Proceedings, International Computer Music Conference*. San Francisco: International Computer Music Association, 1996, pp. 224-227.

[10] C. Roads, *The Computer Music Tutorial*, MIT Press, 1996.

[11] M. Simoni and R. Dannenberg, *Algorithmic Composition: A Guide to Composing Music with Nyquist*, University of Michigan Press, 2013.

[12] G. Wang and P. Cook, "ChucK: A Concurrent, On-the-fly, Audio Programming Language," in *Proceedings of the 2003 International Computer Music Conference,* 2003.