www.logofoundation.org

Teaching Programming with Music: An Approach to Teaching Young Students About Logo

by

Mark Guzdial

Introduction

One of the greatest teaching challenges I've faced has been to teach Logo to young students (specifically, third and fourth graders, with an occasional bright second grader). I have difficulty with teaching turtle graphics. The problem may be entirely mine: there are certainly better teachers, and there may be better techniques of teaching turtle-centeredness and drawing commands. Nevertheless, I can't seem to get these concepts across to my youngest students.

I have taught Saturday computer classes at the University of Michigan Children's Center for several years to third through sixth graders. The students I taught had already become familiar with some form of single letter Logo command (e.g., EZ Logo). The explicit goal of the six to ten week courses was to learn about procedures and the **repeat** statement.

During my second term teaching this course, I decided to experiment with music. The results were so positive that the next few courses continued to use music, and now the class is explicitly a music class. I've used the same procedures and techniques on teachers in a Summer Logo Institute at the University of Michigan with similar success but with different nuances.

This memo is about this technique of teaching programming through music. The teaching style is discussed, then how the music material is introduced and used in the class. Finally, some future plans are presented.

Teaching Style

The cliché term for the style of teaching I used is *guided discovery*, though I wasn't really aware of any particular style. What I did was simply to let the students play, with some encouragement and answering of questions. With at most ten students playing with music on a Saturday, there was little that I needed to do to keep the class moving and motivated. The students enjoyed what they were doing, so I just talked with them and suggested avenues to explore.

The music discussed in the class was simple. Our focus was strictly on composition. Tonic notes were occasionally mentioned, but keys and signatures were ignored. The only real criteria for good music for us was in the ear of the author (though sometimes the author considered the teacher's opinion).

The common questions that I offered the students were, "How does that sound? Do you like that? How would you want to change it? Have you tried this yet...? and How about combining songs like this...?"

Teaching Programming or Teaching Music?

An interesting question is whether the course was about Logo or about music. My opinion is that I was teaching music, but along the way, the students learned about Logo.

For example, I helped the students correct any Logo errors that occurred, but tended to leave the students on their own when they encountered sounds in their songs that they didn't like. I didn't want my students to be frustrated by Logo errors, while unfortunately missing opportunities for the students to learn some important Logo concepts. Instead, I let them be frustrated when trying to get a song to sound just the way they wanted. This philosophy didn't seem to hinder student learning, perhaps because the criteria for a Logo procedure running correctly is stricter than the criteria for a good-sounding song.

Though the emphasis was on the music learning, even the youngest students made frequent use of subprocedures and the **repeat** statement. Clearly, a good bit of Logo was learned. (This sort of accidental Logo learning is not uncommon. For example, E. Paul Goldenberg has spoken at **Logo 86** and the **East Coast Logo Conference** about a pre-calculus class he helped teach at the Lincoln-Sudbury Regional High School. The instructors wanted to teach Logo, but the curriculum was already too large. So, the teachers used Logo in class examples, and computers were made available to the students, but the details of Logo were never explicitly taught. At the end of the course, Paul reports that the students were some of the finest Logo programmers at Lincoln-Sudbury.)

Teaching Music instead of Graphics

I was drawn to music in part because the younger students seemed to find turtle graphics difficult. Turtle-centeredness was hard, and so were degrees. Most difficult was debugging long sequences of **fd** and **rt** commands, seemingly without differentiation. My encouragement to use procedures and subprocedures to ease the debugging task fell on mostly deaf ears.

Music was easier. Students were not frustrated debugging their music procedures, and they tended to use subprocedures. I hypothesize that the students were able to program and debug music procedures more easily than graphics procedures because there is a stronger analogy between music and programming than graphics and programming. Musical phrases correspond to commands. A song is a combination of phrases just as a program is a combination of procedures. Repeating a phrase is the same as iteration.

However, I have only my biased experience as evidence, and I must admit that this hypothesis wasn't formed before I began teaching music. The class is taught on Atari 800 computers with multiple-voice music generators, and Atari Logo provides access to those tones. Making music in the class seemed natural and fun, so we just started doing it. Now I find myself trying to understand what happened.

It may also be that programming with music is natural because both are dynamic processes. When one looks at a drawing, it's not easy to determine what order of drawing lines and coloring created that pattern. However, music has a clear order that maps directly to the procedure that created it. Perhaps animation could be used in the same way as I used music, since it too unfolds over time.

Teaching Music With a Computer

I propose, then, that this was a course on music, that incidentally used a computer as the instrument and Logo as the notation. An important question is why I used a computer in a music class. Why not just have the students learn the recorder or some other similarly easy-to-learn instrument?

One reason is that playing music on the computer requires neither fingering nor strumming nor blowing. All the students are already comfortable with the keyboard, so there are no additional skills to be learned. Further, the students are not limited by their music-playing skills. Even if I taught a particular instrument, the students would not be able to write and play such complex music as they do with the computer in our class.

A more interesting reason for me is that the students explored notation in this class. When they first started, they used the music procedure exclusively. Later, they began to use the **repeat** statement. Finally, they hardly used the given music procedure, preferring to form their songs by re-combining procedures they'd already written. The students invented their own music notation consisting of the words used to name their songs. (Brian Silverman of LCSI once suggested that programming really has two processes: inventing a new language of procedures in which it is easy to solve the problem and writing the easy solution. It is this first part that students are exploring in this class.)

Beginning With a PLAY Command

On the first day of the music course, the students were asked to type in a procedure and a set of **make** statements. The procedure was a **play** command for generating tones, and the **make** statements defined the frequencies for an octave of notes. I did tell the students what the procedure does, but I made no attempt to explain how the procedure worked. Mostly, this was an exercise in using the computer and for trying the Logo editor.

Because we used Atari 800 computers, the procedure we used wouldn't be useful unless you happen to have an Atari 800. (The corresponding Atari Logo code is identical to the APPLE and LCSI LOGO II code, except for the specification of Atari toot's voice and volume inputs. The Atari version used voice 1 and volume 7.) However, the corresponding procedures for other Logo implementations are easy to write. For LogoWriter, this is the procedure and **make** statements:

```
to startup
make "a 881
make "b 988
make "c 523
make "d 587
make "e 659
make "f 698
make "g 784
make "c2 1047
make "rest 0
end

to play :song
if empty? :song [stop]
if empty? bf :song [stop]
tone thing first :song first bf :song
play bf bf :song
end
```

(APPLE and LCSI LOGO II versions appear in Appendix I of this memo.)

The notes make up a single octave on the diatonic scale. The ascending order of pitches is C (middle C), D, E, F, G, A, B, and C2 (C above middle C). Use of the diatonic scale (i.e., no accidentals) encourages experimentation since truly bad music can't be written in only a diatonic scale.

This **play** command takes one list as input. This list identifies notes to be played and durations in an alternating sequence. For example,

```
play [c 20]
```

will play the note middle C for 20 beats.

```
play [e 20 c 40 g 30]
```

will play E above middle C for 20 beats, then middle C for 40 beats, then G for 30.

It's an interesting observation on child and adult learners that only adults were concerned about the units in which the durations are specified. None of the children asked how long a beat is. They seemed content as long as 40 was twice as long as 20. Adults wanted to know the units of the durations as sixtieths (1/60) of a second on an Apple II.

In one class, we did try to use the more traditional Logo notation for music procedures; that is, a list of notes followed by a corresponding list of durations. In this notation, the previous example would be written

```
play [e c g] [20 40 30]
```

The students found this notation confusing. Since they often miscounted one or the other of the lists, the correspondence was hard to arrange and debug.

After introducing the **play** procedure, the students began experimenting. The first exploration of many students was an attempt to determine the greatest duration allowable. (When a majority of the students in the class became interested in this question, we made it a class project with each student testing a different large number as the maximum duration.)

Students soon discovered that

```
play [c 20 c 20 c 20]
```

did not play middle C three times for a duration of 20 each time. Instead, middle C played for a duration of 60. The pauses between the notes were not discernible. A rest would then be introduced as a note. To play middle C three times for a duration of 20 each time, a student might type

```
play [c 20 rest 1 c 20 rest 1 c 20]
```

Once rest is introduced, the students can explore rhythms.

One day the cabinet in which the students' disks were kept was locked. I had no key, and there was no one else in the building but us. I didn't want the students to retype the whole **play** command, and I couldn't remember all the frequencies anyway. So, I typed a modified **play** command into each student's computer that took frequencies instead of notes. We called this version **tones**. In LogoWriter, the procedure looks like this:

```
to tones :song
if empty? :song [stop]
if empty? bf :song [stop]
tone first :song first bf :song
tones bf bf :song
end
```

A song in this version might look like this:

```
tones [100 20 320 40 400 50]
```

to play pitches at frequencies 100, 320, and 400 for durations of 20, 40, and 50, respectively.

This mishap resulted in a wonderful class. Some students had had some previous music training, so they already knew the notes we were working with, and all the students were getting a feel for what the notes were about. Shifting from notes to frequencies caused the students to restart their learning. The songs composed that day were much wilder than previous ones and sounded bad much more often, but some students preferred using **tones** to **play** in future classes.

The first experiments that students engaged in here were like the earlier ones (e.g., searching for the longest duration that Logo would allow) but changed for the new environment. Students would use **tones** to search for the highest and lowest pitches that they could hear. (Some values for pitches caused errors from Logo. Other values caused no errors, but also made no sounds. These values were beyond the capacity of the monitor's speaker to play, or our ears to hear.) That these were the first experiments should come as no surprise: students naturally test boundaries, whether they be natural, man-made, or computer-based.

At this point, I might have encouraged older students to explore some new assignments of pitches to notes. Who said that middle C had to be 523 Hz? Why not make C 100, D 200 and so on? How might this system differ from the one we commonly use? Is **play** as successful with the new **make** statements?

Group Music

Mostly, what the students did in this class was to compose music. Adults tended to begin by composing songs that they already knew like "Three Blind Mice" and "Jingle Bells." Children invented songs. Occasionally, some students in the class would have had previous music training, and these students would often begin by entering a song they had already learned, perhaps trying variations.

When ten students are exploring music in the same small room, the discovery of harmonies and disharmonies is almost guaranteed. The students soon discovered that songs played together sound different than when each is played separately. The Atari 800 supports multiple voice sounds, and in some classes we used multiple voice **play** statements. Ironically, even in those classes where we used the full capabilities of the Atari, the students seem to prefer single voices emitting from multiple computers than multiple voices emitting from a single computer.

When students remarked on the sound of multiple computers playing simultaneously, I would set up some joint music projects. First, I would explain to the students how to calculate the length of a song; that is, the sum of the durations in the song. So **play [c 20 e 30 rest 10 g 40]** has a duration of 100 (20+30+10+40).

We had various multiple-computer activities based on the length of songs in these classes. One of the first was to have each student write a song whose length was a large number, say, 500. The students were to enter their songs and the command to play them, but to wait before hitting the **Return** key. On the count of three, all the students would press **Return** at once.

The resulting song was always interesting. The younger students and those less confident in mathematics tended to play only two or three notes, each with a long duration. The older students and those wishing to show off would tend to write long songs with shorter durations (e.g., five or ten beats per note). When the group song was played, the younger students' part became a background (sometimes harmonious, mostly dischordant) for the older students' melody.

Another activity we explored was to have part of the room rest while the rest of the room plays. By using rests in their songs, the students could let 1/3 of the room play while the rest awaited their turn. So, one group might play for 200 beats, then rest for 400, while another group rested for 200, played for 200, then rested for 200, and the final group rested for 400 then played for 200. If the groups were determined by location in the room, the effect was that the song would move around the room as it played.

Learning Logo While Learning Music

The process of learning Logo in this class was shaped by student problems in their music procedures and requests for new tools to use in their music. Music was the driving force in their learning of Logo.

A problem that appeared early in the class was dealing with long **play** commands on the command line. Unlike LogoWriter, Atari Logo has the traditional toplevel command input: once a command is entered, it's lost to future editing. A long song entered as a **play** command at the toplevel prompt was lost to change or replaying unless the student laboriously reentered it. Procedures were the cure for this problem. By entering their songs as procedures the students gained the ability to edit and change their songs after execution, to save and replay their songs later, and to name their songs.

Some students would ask about repeating a song or a section of a song without having to retype the procedure name or the **play** statement. This was a natural place to introduce the **repeat** statement.

Later in the class, the students would begin tackling larger songs. Often these pieces were combinations of songs written previously, either as a showcase for their masterpieces or as a tactic for saving work. Combining subprocedures into a superprocedure made this reuse simple. Usually, I would first introduce the ability to type several procedure names at once at a single toplevel prompt, separated by spaces. For many students, that's enough for their music, but others go on to use these subprocedures in other procedures.

In one class, a bright second grader wrote three procedures **baseball, upper,** and **deck.** (He explained to me that **baseball** was the sound of a runner rounding the bases and that **upper** was the sound of the crowd cheering.) When this student learned about typing multiple procedures on one statement, he typed the statement:

```
baseball upper deck
```

After hearing these three songs play, he asked me if he could change the order. I told him that he could, so he entered:

```
upper deck baseball
```

He then asked me what all the combinations of these three procedures were, so we sat down with pencil and paper to explore combinatorics.

Once a student encountered one of these problems, I would present the Logo solution to the entire class. Rarely did I have to repeat this presentation. Most students remembered the Logo as solutions to problems they were facing, and others asked their fellow students or watched someone else's screen.

What Hasn't Worked

Students in these classes often confused song procedures with the input to the **play** procedure, especially when using **repeat** statements. They would occasionally type statements like:

```
repeat 4 [play [mysong]]
```

where **mysong** is a procedure containing **play** statements. I never came up with a good solution to this problem. I would explain that it's wrong and why, but I didn't give a very convincing explanation, so the problem would recur. Perhaps it would have been helpful to change **mysong** into a reporter that presented **play** with an input, as in **play mysong**, but that would require introducing the **output** statement to the students. Whether the tradeoff of easier syntax at the cost of an additional statement would be worthwhile might depend on the individual student.

In a few classes, students tried to develop procedures that mixed music and graphics, but this was not successful for most students. Only a few older students were able to handle the difficulties of switching between debugging **play** procedures and graphics procedures. Others found the long sequences of **fd** and **rt** confusing.

Occasionally, I tried to get students to formalize what sounds good and what sounds bad in a song. For example, I once had students fill a chart indicating whether the sound was good, bad, or eerie when one student play one note (say, a G) while another student played a different note (say, a D). The students never reached a consensus over what sounded good, bad, or eerie, and they complained that they'd prefer to try a particular combination in a song than to write it down. So much for that attempt at formalization.

Future Plans

In a future class, I hope to have students explore formal notation in a different way - by inventing their own **play** commands for a harmonica. The purpose for the harmonica is to propose a challenge: how does one notate harmonica music on a computer?

A typical harmonica has ten holes. Each of these ten holes can play two notes: one when you blow air into the hole, and another when you draw air through the hole. Only one complete scale is available on a traditional diatonic harmonica. The other holes provide parts of an upper and a lower octave.

Playing a harmonica isn't hard, though the students in this class would not be expected to achieve any sort of proficiency on the harmonica. Instead, I would invite them to determine what sort of notation would work for this instrument. Whatever they invent, I'll implement and then let them test and suggest modifications to the notation. Can they compose songs for the harmonica on the computer? Can they notate songs invented on the harmonica into the computer? A real instrument with a virtual music world seems like a terrific setting for exploration.

Appendix I
APPLE and LCSI LOGO II Procedures

For APPLE and LCSI LOGO II, this is the procedure and **make** statements:

```
make "a 881
make "b 988
make "c 523
make "d 587
make "e 659
make "f 698
make "g 784
make "c2 1047
make "rest 0

to play :song
if emptyp :song [stop]
if emptyp bf :song [stop]
tone thing first :song first bf :song
play bf bf :song
end

to tones :song
if emptyp :song [stop]
if emptyp bf :song [stop]
tone first :song first bf :song
tones bf bf :song
end
```