Informatics in Education, 2018, Vol. 17, No. 2, 151–166

DOI: 10.15388/infedu.2018.09

Integrating Computational Thinking with a Music Education Context

Judith BELL[1], Tim BELL[2]
[1]*Chisnallwood Intermediate School, Christchurch, New Zealand*
[2]*University of Canterbury, New Zealand*
e-mail: jbell@chisnallwood.school.nz, tim.bell@canterbury.ac.nz

Received: Mach 2018

Abstract. Computational thinking is becoming common in K-12 curricula, and at the same time there is interest in how STEM subjects can be integrated with the Arts (referred to as STEAM). There are some obvious connections between music and computation, but the idea of engaging with genuine computational thinking while also having authentic music learning experiences for students provides new opportunities.

In this paper we consider ways to explore computational thinking ideas such as decomposition, patterns, abstraction and algorithms in a meaningful way while also exploring key concepts that a music educator would expect to work with. We review some existing ideas for doing this, and also provide novel approaches that connect computational thinking and music. This is done through a series of vignettes that describe creative ways to connect the two subjects using approaches that have been used successfully with school students.

The first approach is based on the use of comparisons in sorting, which can be used to have students physically compare musical elements such as note pitches. The second uses simple programming on physical devices to represent music notation. Further examples include exploring binary representations using sound, writing programs for musical scales, understanding musical phrases in the context of programming, and using programming for music composition.

Integrating the opportunity to learn about computational thinking and music at the same time has the benefit that some efficiency can be gained in teaching, but more importantly, students are able to experience the relevance of these two subjects to each other, when they might otherwise pigeonhole them into separate areas of their lives.

Keywords: computational thinking, music theory, curriculum integration, STEAM, parallel sorting network, binary representation, steganography, MIDI, scales.

Introduction

As school curricula in many countries are finding ways to include "computational thinking", under titles such as "Computing", "Digital Technologies", and "Computer Science" (Hubwieser, Giannakos, Berges, *et al*. 2015; Duncan and Bell, 2015; Heintz, Mannila,

& Färnqvist, 2016), we have the challenge of fitting it into what is usually an already full school curriculum. This raises the concern that STEM subjects will overshadow traditional subjects that are important for a rounded education. However, the arts have an important role in STEM subjects because they encourage creativity, communication and teamwork, and this has been articulated through the use of the term "STEAM", which includes Arts in STEM.

In this paper we provide several vignettes of novel ways to address these issues in music education by integrating aspects of music education with computational thinking. Integration means that both subjects are being taught at the same time, which not only provides efficiency in the use of class time, but also helps students to avoid seeing subjects in isolation, and can potentially engage students who are more attracted to one of the integrated elements than the other. The contrast may seem particularly strong when music and computer science are juxtaposed, particularly based on stereotypes of music as a creative art and computer science appearing to be a machine-centric science. In fact, they have more in common than might be expected from the stereotypes, both in terms of skills needed to be effective (such as creativity, teamwork, communication, working with notation), and also the opportunities to use one subject to engage and even enhance learning in the other. Furthermore, in practical situations both music and computer science involve constructing something that is intended for an audience; in computing the audience might be the users of an app or web page, or an organisation collecting data, while in music the audience might be at a concert, watching a film, or listening to an advertisement.

The challenge is to design activities that include genuine music learning as well as genuine computational thinking. Simply using computers in a music class (such as making digital recordings or using online resources) is unlikely to help students engage at a deep level with computational thinking. Similarly, using music in a computer class (such as playing background music in a game or learning how to compress audio files) is unlikely to be teaching many key elements of music.

In this paper we will reflect on what the key elements of the two subjects are, and then provide a series of specific ideas for ways to genuinely integrate these elements in activities and lessons that can be used in a school setting. This paper is an extension of a version presented in the Constructionism 2018 conference (Bell and Bell, 2018).

The Elements of Computational Thinking and Music

In order to be sure that an exercise is teaching both computational thinking (CT) and music, we need to be aware of the kinds of concepts that are covered by these two areas of learning.

Curricula based on CT generally include learning to create new artefacts on digital devices, as well as using the devices. This often includes working with and for others, which has been argued to be an important part of this discipline (Kafai, 2016). There are many definitions of CT, and even some debate about what should be included (Tedre and Denning, 2016), but there are common elements that appear in most definitions. For

example, Selby and Woollard (2013) use the following list: algorithmic thinking; abstraction; decomposition; generalization and evaluation. Computer programming isn't listed explicitly here, although learning to program does cover these concepts well, and some would argue that it defines the scope of CT (Denning, 2017). In this paper we evaluate the relevance of each activity against these CT criteria, including programming, since this gives concrete evidence that they are likely to genuinely support CT in a computing curriculum. Our examples also provide broader contexts in which students can engage with CT.

A music curriculum, at a high level, will typically cover creating, performing, responding and connecting (Kaschub and Smith, 2016), and the elements of music that define the scope of curriculum are often articulated as a list such as pitch, timbre, texture, dynamics, duration, tempo, and structure (Burton, 2015). As with CT, the elements do not provide a curriculum, but they help us to identify if something belongs in a music curriculum. For a general classroom course these are likely to be made accessible through the use of basic notation, accessible instruments, and meaningful contexts such as popular music, film music, and local culturally relevant music. We will evaluate activities against these criteria to ensure they match the concepts that are likely to appear in school music curricula. We note that these are very atomic components of music, and are likely to be covered by broader descriptions. For example, pitch might come up in relation to understanding the range of a musical instrument, which in turn might be in the context of the instruments of an orchestra or other ensemble.

There are common elements between music and CT. In practical situations, both rely on notations in formal languages (for music that could include Common Music Notation on 5-line staves, tablature, solfège notation or other notations; for computing it includes programming languages as well as markup languages and protocols); and both use concepts around sequence (the order in which notes appear in time; and the order of statements in a computer program) and repetition (in music this includes repeats, as well as forms such as rondo or the structure of popular songs; in computing loops and recursion provide this).

However, we are not advocating that these related ideas should be used directly as common examples, as sometimes analogies and differences might add to confusion, but we do note that there are already related forms of thinking in both subjects.

In the remainder of this paper we give some vignettes of ways that music and CT can be combined in a way that genuinely engages students with both subjects. Some are built on activities from CS Unplugged (`csunplugged.org`), and the others are based on programming in a simple block-based language. The exercises mainly cover music theory, but also some composition.

Parallel Sorting Network

A parallel sorting network is an abstract concept for parallelising the task of ordering data into a sequence, usually in increasing order of value. It is a quintessential activity from the CS Unplugged material (Bell, Rosamond, & Casey, 2012), where the network

is marked out on the ground (or floor) using chalk or tape, and students traverse the network and come out in sorted order.

Fig. 1 shows the layout of a 6-way sorting network. Six students enter the network at the left, each holding a card with a numeric value on it. When they meet at a node, they compare numbers, with the smaller value leaving to the left, and the larger one to the right. This simple rule is repeated each time they meet, and at the end the students emerge with their cards in ascending order. A key point is that only a very small amount of instruction is needed to explain the rules for students, and then they are able to explore and experiment with the structure they have been given.

Since sorting can be applied to any values that have a binary relation that is a total order, sorting networks can be used for keys other than numbers, including putting words or names into alphabetical order, or putting elements of a well-known story into the sequence in which they should occur. This opens up a number of possibilities in music.

One valuable approach is to have students compare written notes in Common Music Notation (Fig. 2, left). Initially this can be simple notes that are on the same clef and only use lines and spaces (so the simple rule is that the note that is further up the stave vertically should go to the right). This can be extended by adding accidentals, so that two notes on the same line might be distinguished by being sharp, natural, or

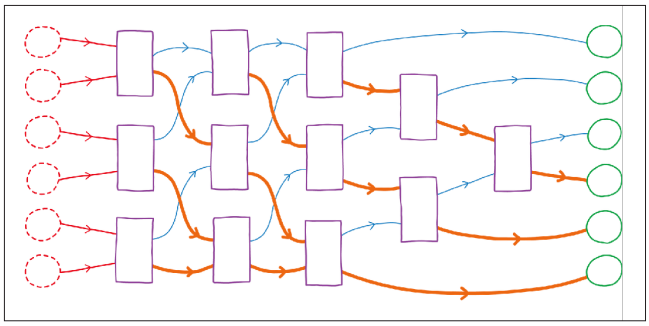

Fig. 1. A 6-way parallel sorting network.

Fig. 2. Comparing musical notes (left) and bell pitches (right) in a sorting network.

flat. If the students have mixed levels of understanding, the "harder" notes (with accidentals) can be given to a more advanced student, so that if such a comparison is needed, the student holding the more difficult card will be able to explain it to the student that they meet each time, leading to a natural form of peer instruction that is repeated every time they meet a student at a node. The next step up is to add ledger lines and different clefs (bass, alto, tenor). This approach could be used with a variety of notations, such as solfège, tablature or numbered music notation, and even mixing up different notations!

The teacher can add notes that appear to be the same (such as G flat and F sharp), without giving instruction to the students. This quickly leads to a discussion on which of the two is higher; on the piano they are the same note, in which case either order could be used, but on other instruments such as violin they are technically slightly different, and an ordering can be found. The computational process is being used to raise a number of such questions for student to explore.

Another extension is to compare note and rest lengths (students are given quavers/crotchets/minims and so on, with tied and dotted notes used as an extension). It can also apply to dynamics (***ff, f, mf, mp*** etc.) and tempo indications (Allegro, Andante, Presto etc.). Aural skills can be exercised if the students are given instruments that make a sound; we have found tuned bells to be particularly effective, as it is possible to get bells that are the same size but have different pitches (Fig. 2, right). This forces students to make pitch comparisons over and over, and success is measured if the bells come out in ascending order. A further link with computation can be made by having students program a small app on a mobile device to play a note, and use their notes as the basis of comparison.

While a fixed sorting network itself is fairly easy to master, we have been surprised that students enjoy using it regularly. They can try out different things to compare, and reason about the meaning of the order for different musical elements. The activity has a built in gamification in that students want to complete the task as quickly as possible, but if they are inaccurate because of speed then the team doesn't achieve a successful outcome.

From a computational point of view, students can encounter the factorial number of orders that the input can start in, and the idea that two identical values end up being sorted together. It touches on every aspect of CT, but is particularly strong in abstraction (the algorithm works regardless of what is being sorted), decomposition (a sophisticated outcome is broken into very simple comparison steps), and logic (trying to explain, for example, why the smallest value will always find its way to the correct node). The CT aspect can be extended by having students design and test small sorting networks of their own for 3 or 4 inputs.

As well as touching on aspects of CT, students are repeatedly gaining experience with the notation for pitch and rhythm, and exercising their aural skills in a motivating context; and as they become comfortable with a notation, it can be extended by adding more difficult variations. When using this we have repeatedly found that students are able to meaningfully grasp music concepts beyond their current music theory level, which results in much deeper understanding of their current knowledge.

Harmonic Theory through Positioning Robots

An activity based on simple turtle-style robots is to draw a large stave on the ground (Fig. 3), and have students program simple robots to go to the location of a specified note on the stave, enabling them to represent notes and chords. The robots in Fig. 3 are "Bee-Bots", a simple device that can be programmed using only four commands (forward, back, left, right) that are entered as a sequence. These commands are then followed when the "go" button is pressed, which is equivalent to running a program. The Bee-Bot moves 15cm for each forward/back command, so with the stave lines 30cm apart, each unit of movement corresponds to one space or line on the stave.

Initially the Bee-Bot only needs to move forward to the given note (e.g. "get the Bee-Bot to go to D"), but this can be extended into notating a short tune either with the Bee-Bot pausing on each note of the tune and spinning 360 degrees, or having multiple Bee-Bots. This now requires programming the position in two dimensions based on the simple movements.

In our experience, students initially make mistakes with the programming, but are determined to get to the right note. After a while they become adept at the "language" of the Bee-Bots, and some have even taken the risk of sending the Bee-Bot on longer paths than needed to introduce some humour to the exercise. This approach is easiest using tunes based on a diatonic C major scale, although representing sharps and flats can be done by the direction the Bee-Bot is pointing (left for flat, right for sharp).

Another variation is to use two Bee-Bots, and work with intervals, either programming them to show an interval (possibly given aurally), or having other students name the interval that has been displayed. This can be extended further by having one group of students program a chord with three or four Bee-Bots either vertically aligned (harmonic) or in a horizontal pattern (melodic), which also provides the challenge of avoiding collisions. We then have another group play the chord on a piano, and/or name the chord (starting with simple triads, and extending to other types of

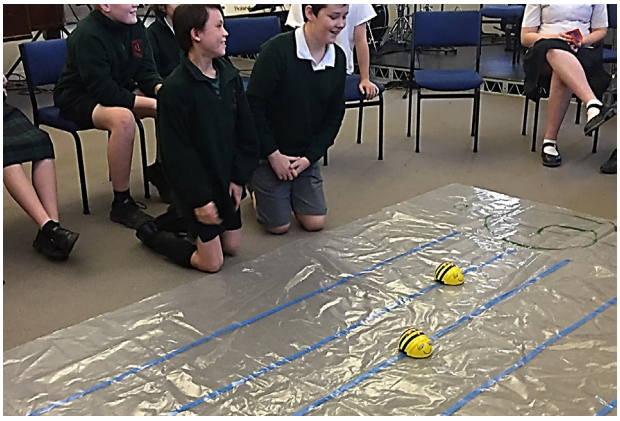

Fig. 3. Students using Bee-Bots to show a musical interval.

chords, including inversions, 7ths, *sus* chords and so on). A further extension is to change the clef.

A challenge that exercises *evaluation* from CT is to try to make all the Bee-Bots arrive at their final destination at the same time. This would require students to accurately determine the number of steps in the "program" before it is run, and pad out the steps for shorter routes to match the longest one. This exposes students to the idea that we could analyse the running time of an algorithm before it is executed.

This exercise motivates students to repeatedly use the fundamental idea in programming of sequences, with the concomitant CT elements, particularly algorithmic thinking, and decomposition of the path into steps. Musically, students are working with rudiments of music theory, particularly pitch notation, accidentals, clefs, intervals, and triads/chords. They also need to demonstrate teamwork to create "chords".

Another variation of this is to create ukulele or guitar diagram grids for students to program the Bee-Bots to land on chord shapes. In this case the grid lines are 15cm apart, representing the strings and frets.

Binary Representation Based on Sound

The idea that all data on computers (text, sound, images, numbers etc.) is represented as binary numbers is fundamental to the idea of a *digital* device. The CS Unplugged activities include a popular way to introduce binary numbers with minimal mathematical background required (in fact, the main prerequisite is being able to count up to 31). This is done by having students work with 5 cards, with 16, 8, 4, 2 and 1 dots on one side of them respectively. The CS Unplugged website gives a lesson plan for a constructivist approach to show students the relationship between decimal numbers and binary representation by flipping over the cards as a binary representation.

This segues to representing letters of the alphabet using these numbers; for the 26-letter English alphabet students will usually suggest using 1 for A, 2 for B, 3 for C, and so on. For example, the binary number 11010 represents the number 26, or the letter Z. Since the 0 and 1 digits are abstract representations, this transitions easily to the idea of using other representations that have two values, including sound.

Getting to this point can typically be achieved in about ten minutes of class time, as the CS Unplugged scaffolding makes it fairly simple to understand. In the CS Unplugged lessons, students then explore patterns in binary numbers, but in a music setting, we can look at how to encode letters and other symbols using sound. A variety of alternatives for representing binary values can be explored (loud/soft, long/short, timbre, melodic patterns, low/high), with the latter (low/high) being effectively how a telephone modem works. Once students realise that letters of the alphabet can be transmitted using only high and low notes, a tune can be used to represent some text.

As a simple example, the first exercise in the CS Unplugged "Modems" activity has recordings of a jazz singer singing a sequence that students can decode (see `https://classic.csunplugged.org/modems-unplugged/`). The tune for the first recording is shown in Fig. 4; the numbers under the high notes show the bit values that are "on";

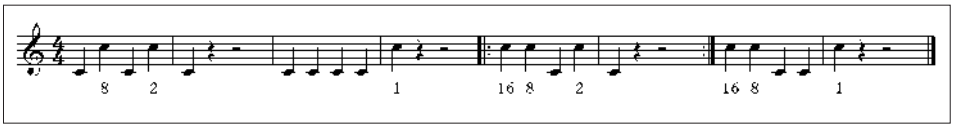

Fig. 4. Binary representation of letters expressed as music.

for example, the first five notes have a high note for the 8 and 2 bits, so it represents letter number ten in the alphabet, which is "J" in our example alphabet. The second set of five notes represent the number 1, which is "A"; the complete word is "JAZZY".

Transmitting words by singing binary values can be used to illustrate how binary representations are insensitive to inaccuracies in the physical transmission; for example, if the notes above were slightly out of tune, or out of time, or obscured by the backing, it can still be obvious which notes are high and low.

The concept of encoding messages and hiding them in an apparently unrelated format is called "steganography". The idea that this is even possible is an important opportunity to show students how innovative algorithms can achieve something that might not have been thought possible, and can motivate students to create their own hidden messages.

This leads to a constrained exercise in composition for students – they should first work out the binary representation that they would like to hide in a composition, and then write a meaningful piece of music that encodes the message. This is typically done using notes that are clearly high and low, but any other two-state representation could be used, such as whether each note is higher or lower than the previous one, a percussion part that has two sounds (such as a kick and snare drum), or a backing part that is ascending or descending. The composition could be performed live on acoustic instruments, entered into recording or notation software, or played using websites that allow students to create music using a range of graphical input formats.

As an intriguing variation, the hidden message could be a melody itself, represented with diatonic music numbers or MIDI (which maps the pitch of a note in semitones onto whole numbers from 0 to 127). If MIDI is being used, the notes can be represented in 7 bits, and this could map on to rhythms with 8 quavers (eighth notes) in a 4/4 bar (notes or rests for 1 or 0 respectively), with either the first or last quaver being chosen by the composer, and the other 7 based on the bits in the MIDI note. In this case, a melody is being represented by a rhythm.

This activity exercises a variety of CT concepts, including algorithms for converting between decimal and binary, abstraction to the level that even representations may have different representations (such as representing a tune as MIDI, representing MIDI in binary, and then representing the binary values using music!), and evaluation of the number of bits needed to represent a range of symbols (e.g. for alphabets of more than 26 characters).

Musically, it provides motivation for composition, as well as some constraints. Students listening to steganographic recordings will be exercising their aural skills to recognise the binary digits that make up the message. The idea of steganography can also be a motivation for students since it raises the idea of using artistic media to communicate hidden digital messages.

Music Theory through Programming Note Sequences

Music theory contains a lot of rules around sequences of notes, particularly scales, which are fundamental to many genres of music. These scales are based on repeating patterns, so they are amenable to being programmed. Most programming languages are able to play MIDI notes, which opens up the possibility of writing programs to play scales. Parameters could be added to determine aspects like the key, range and speed.

For example, Fig. 5 shows ways that a simple one-octave chromatic scale can be programmed in the Scratch language, starting on a C (MIDI note number 60). Each note played is 1 semitone higher than the previous one (the "change degree by 1" command). Students can be given this as an example, and asked to identify the scale it plays. They can then "remix" the program (edit it) to make the note number change by 2 each time, creating a whole-tone scale, which has a distinct sound to it, and can be linked to musical examples, including Debussy's work and (as a specific example) the introduction to Stevie Wonder's "You are the sunshine of my life". Changing it to steps of 3 produces diminished 7th arpeggios, which can be linked to the idea of building tension through music, and (for example), the classic silent movie accompaniment to signal impending disaster, typically associated with a villain.

Students can explore other intervals; using steps of 4 semitones produces augmented triads; 5 semitones produces quartal harmony and a cycle of fourths that go through every note; 6 semitones produce tritones, which divide an octave in half and raise interesting harmonic implications; and 7 semitones produces a cycle of fifths. Using 8 and 9 semitones produces widely spread arpeggios, but the notes the inversions of augmented and diminished triads respectively (since 8+4 and 9+3 semitones add up to an octave), and this pattern continues for 10 and 11 semitones, which are the inversions of the 2 and 1 semitone scales that we started with.

Students could also be challenged to make the notes ascend in octaves (12 steps, which explores the idea that there are 12 semitones in an octave), to write descending scales or arpeggios (negative steps), and identify them aurally. When the notes increase in octaves they can quickly go out of audible range, although in practice if MIDI is being used, the maximum value is 127 (7 bits), which is more than an octave past the end of a piano keyboard and higher than any conventional orchestral instrument, but (just) within the limits of human hearing. These issues all provide valuable discussion points around binary representation, human hearing, and the ranges of musical instruments.

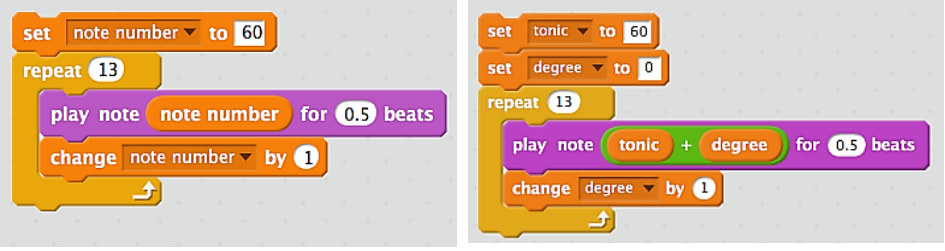

Fig. 5. First attempts at a Scratch program for a scale.

Fig. 5 also shows how the choice of variable names provides a valuable form of integrated learning. The terminology in the second version (tonic and degree) for how the note is calculated articulates how the logic of the program works. In this example, the term "degree" is not quite musically accurate as it is the number of semitones above the tonic, rather than the degree. While such discussions might seem pedantic, they give the opportunity to work with the precise terminology in music, and to recognise the importance of accurately-named variables to make a program understandable.

The next challenge is to have students adapt the program for other scales and arpeggios (such as major, minor, and modes). These will involve a mixture of intervals between notes for each octave (for example, a major scale increases the pitch of each note by a tone, tone, semitone, tone, tone, tone, semitone respectively) to give 8 notes. Programming this forces students to think about each degree of the scale, and they can debug their program by listening to it and checking that the scale sounds like a conventional major scale. They could also write a program to play a sequence of scales or arpeggios (e.g. C major, then C# major, D major, and so on), which provides opportunities to work with nested loops. For more advanced programmers, this also raises the possibility of storing the degrees of the scale in a list, and looking up the degree in the list, which makes it easy to move up and down a scale, or implement modal scales.

Students can then use these tools to create their own music using programming as the "instrument". This raises new possibilities because the complexity of the performance isn't limited by the dexterity of the performer. One student extended the scales into creating a composition that had very fast chromatic movement in it (reminiscent of the Rimsky-Korsakov's "Flight of the Bumble Bee"), while another student extended the program to play a range of major scales in different keys. Arpeggios could also be used as they were in early electronic games that had limited access to voices; fast arpeggios could approximate chords using just one voice.

These exercises give students a direct application of the programming they have learned to the subject of music, which they may be passionate about. Working with scales engages students with loops and variables in a way that there are precise desired outcomes (such as playing a particular scale). The need for accuracy playing scales can transfer from musical training to the way the program does precisely what is specified. As well as working with algorithms, students are using abstract representations of notes, and are also using decomposition to break scales into their elements. Musically, students are developing their knowledge of music theory (such as the intervals between notes in each scale and use of musical phrases) as well as their aural skills.

Linking Musical Phrases to Procedures in Programs

Another valuable activity is to have students write programs to play a tune, and use musical phrases that can be represented as computational functions or procedures ("More Blocks" in Scratch), so that they are aware of both the structure of the music and using the CT concept of decomposition (Greher and Heines, 2014).

Here we will illustrate it with the song "Twinkle, Twinkle little star", shown in Fig. 6.

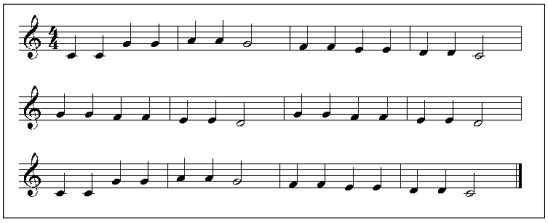

Fig. 6. Twinkle Twinkle Little Star in Common Music Notation.

An initial observation is that there are many ways to notate music. Fig. 7 shows a range of notations are essentially equivalent, and these notations themselves are closely related to the formal languages that appear in Computer Science; each of the notations conveys the same tune, although some may be more suited for particular contexts than others. One difference from computational languages is that music notation doesn't necessarily show every detail of the music; for example, cultural norms for the emphasis of beats (such as 1 and 3 in classical music, 2 and 4 in many forms of contemporary music), and rhythmic interpretation (such as swing in jazz and "notes inégales" such as the double-dot convention in Baroque and Classical music) mean that the written notation requires interpretation with regard to the context in which it was written and will be performed.

Putting aside the issue of interpretation, the song "Twinkle Twinkle" can be written in a programming language as a sequence of notes as shown in Fig. 8. This program uses a sequence of 42 instructions, one for each note in the tune. This approach produces the correct output and students will naturally use this to express a tune, but it is not an ideal representation of the melody both computationally and musically.

In music, a melody like this can be broken up into melodic phrases, where a phrase is a "musical sentence". In the case of "Twinkle, Twinkle", the phrases would most likely be seen as 4 bars long each, although a case could also be made for two-bar phrases. The phrases can be represented explicitly in Scratch by creating a "new block" for each one (this is the equivalent of a procedure or function in other programming languages) as shown in Fig. 9. This reveals the ternary ABA structure of the piece. Phrase B also uses a

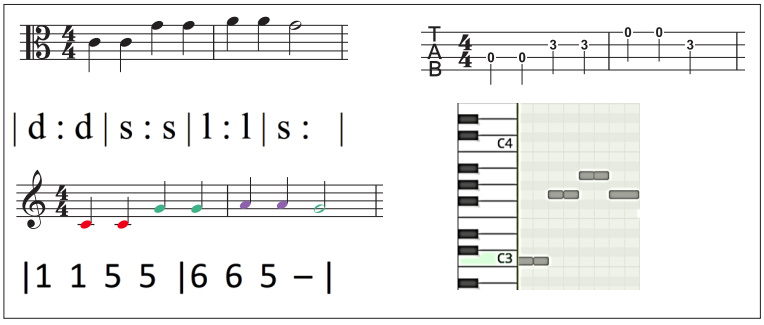

Fig. 7. The first phrase of Twinkle Twinkle Little Star in a variety of notations (Alto clef, Ukulele tab, Solfège, Boomwhacker notation, Jiănpǔ (numbered musical notation), piano roll.

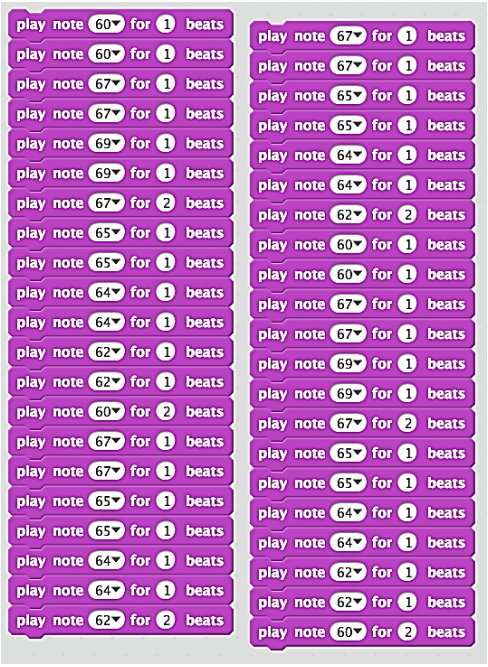

Fig. 8. Twinkle Twinkle Little Star implemented in the Scratch programming language as one sequence.

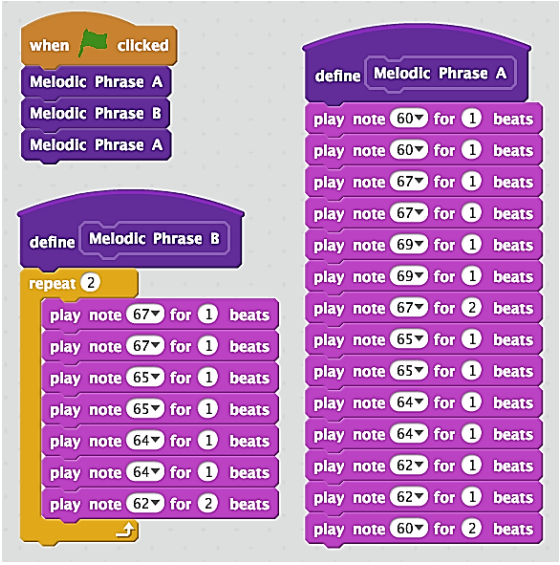

Fig. 9. Twinkle Twinkle Little Star using four-bar phrases.

Scratch "repeat" command that reflects the structure of that phrase. The new version has only 28 blocks instead of 42 in the linear version, and although this represents a minor saving in space, the main point is that it makes the program considerably easier to write, read and debug (for example, if one note was wrong in a phrase, correcting it will fix it in all uses of that phrase). In Scratch, some students may be more familiar with events that can be triggered by messages, and may use these to implement the phrase structure, which leads to an interesting discussion about the relative merits of each approach, and grappling with the idea that there will always be multiple ways to achieve a given outcome when programming.

The structure shown in Fig. 9 isn't the only way that the music might be represented. For example, Phrase B could be broken into two sub-phrases in a separate block, as shown in Fig. 10; or even into two-bar phrases, leading to the structure in Fig. 11. The key point is that considering how best to do this engages students in musical thinking, analysing the structure of the music, and discussions on alternative representations are more important than getting some "correct" solution. If different approaches are used, they can be tested for accuracy by playing the song. Working out how to decompose a program into modules like this is important in computational thinking, and it is also

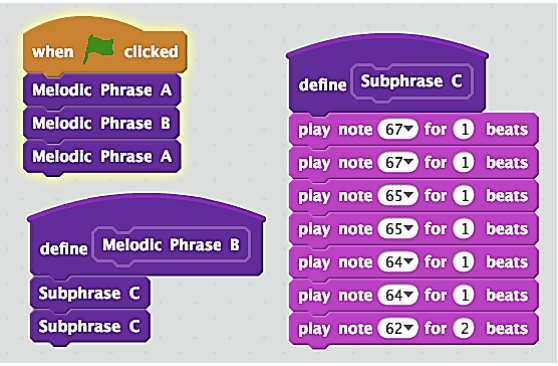

Fig. 10. Twinkle Twinkle Little Star using four-bar phrases, with sub-phrases.

Fig. 11. Twinkle Twinkle Little Star using two-bar phrases.

valuable musically to reflect on the structure of the piece of music. It also opens the possibility of re-ordering phrases to experiment with how they can be repurposed to create a new tune.

Greher and Heines (2014) use a similar idea to identify common riffs in contemporary music, or to play phrases as a round (canon) in traditional songs.

This idea also comes up in musical forms such as ternary (ABA) and rondo (ABACA…), although sometimes there are subtle variations between repeated segments of the music that can make these examples more challenging to implement in a computational setting. However, simple examples such a folk songs that fit this well can be used to reinforce the patterns that appear in music, and the value of decomposition and modularity in computational thinking.

Programming for Music Composition

The idea of using algorithms to generate music is well-established, and dates back to Mozart's "dice music" and earlier (Simoni, 2003). Implementing compositional algorithms as computer programs to create music directly has also been done for some time (Fitch & Leach, 1995), but there are now easily accessed systems available for school students to generate music algorithmically in a context that is intended to teach programming while at the same time building on their interest in music. This is the general principle of "Media computation" (Guzdial, 2003), which takes everyday media such as sound, images, video and text, and teaches computer programming through writing code to manipulate these familiar forms of data.

The book by Greher and Heines (2014) on "Computational Thinking in Sound" explores the idea of creativity in the contexts of both music and computer science, including finding projects around creating music in a computing environment that are meaningful in both disciplines. They start by using found instruments and invented notations for them, then move through audio editing software (which reflects sequential notation), to notating music in Scratch (as described in the previous section), then creating input devices to control sounds in the program and therefore building new musical instruments. They also reflect on how to assess work in a course that crosses the two disciplines.

Another approach is taken by Engelman *et al*. (2017), who have developed EarSketch, an environment where students can manipulate audio *samples* using Python programming to generate music that is meaningful to them. This approach can generate a form of music that is able to connect with the culture that the students are in, using a programming language that is commonly used in computer science courses. Engleman and colleagues reported that the creative learning environment resulted in students showing an improvement in attitude and persistence.

The Sonic Pi system (Aaron, Blackwell, & Burnard 2016) also provides ways for students to express themselves musically through a programming language based on the Ruby language. Sonic Pi is suited to "live coding", where the system continues to generate sound while it is being modified by the programmer. This makes it more akin to a musical instrument that is used for live performance, and also supports contemporary

genres that will be meaningful to school students. It also combines composition with performance (i.e. improvisation), since the program is both the notation and the tool for generating the sound. Sonic Pi will run on conventional computers, but was specifically targeted at the Raspberry Pi computer, which provides a physically different context for making music. and yet another motivation to master programming techniques.

Both EarSketch and Sonic Pi are based on conventional text-based languages that are used in industry, and this gives students a sense that they are doing something more substantial than a programming exercise, since the context is using professional tools to generate music that could be performed professionally. All of the systems discussed in this section provide connections between the elements of programming that students will benefit from learning, and genuine elements of music education.

Conclusion

The examples given above are intended to seed ideas for meaningful ways to integrate computational thinking and music, and to show how arts can have a primary role in CT learning. They provide ways for a student with a passion for one subject to engage with the other, and provide a way for a teacher who is strong in one area to engage with teachers who have expertise in the other. This can encourage collaborations between teachers in the different disciplines, which can lead to more benefits if the subject is explored with guidance from experts in both subjects.

Of course, the examples above only touch on aspects of each curriculum. Since computational thinking is new in schools, we hope that opportunities for integration will only grow as teachers open their eyes to the possibilities, and as creative ways to bring about collaborations with a variety of subjects emerge.

References

Aaron, S., Blackwell, A.F., Burnard, P. (2016). The development of Sonic Pi and its use in educational partnerships: Co-creating pedagogies for learning computer programming. *Journal of Music, Technology & Education*, 9(1), 75–94.

Bell, J., Bell, T. (2018). CS Unplugged and Computational thinking. In: V. Dagienė & E. Jasutė (Eds.), *Constructionism 2018*. Vilnius, Lithuania. 21–28.

Bell, T., Rosamond, F., Casey, N. (2012). Computer Science Unplugged and related projects in math and computer science popularization. In: H.L. Bodlaender, R. Downey, F.V Fomin, & D. Marx (Eds.), *The Multivariate Algorithmic Revolution and Beyond: Essays Dedicated to Michael R. Fellows on the occasion of his 60th birthday* (Vol. LNCS 7370). Heidelberg: Springer-Verlag, Berlin, Heidelberg, 398–456.

Burton, R.L. (2015). The elements of music: what are they, and who cares? In: *The Australian Society for Music Education National XXth Conference Proceedings*. 22–28.

Denning, P.J. (2017). Remaining trouble spots with computational thinking. *Communications of the ACM*, 60(6), 33–39.

Duncan, C., Bell, T. (2015). A Pilot Computer Science and Programming Course for Primary School students. In: *WIPSCE 2015* (pp. 39–48). `http://doi.org/10.1145/2818314.2818328`

Engelman, S., Magerko, B., McKlin, T., Miller, M., Edwards, D., Freeman, J. (2017, March). Creativity in Authentic STEAM Education with EarSketch. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. ACM, 183–188.

Fitch, J.P., Leach, J.L. (1995). Nature Music and Algorithmic Composition. *Computer Music Journal*, 19(2), 23–33.

Greher, G.R., Heines, J.M. (2014). *Computational thinking in sound: Teaching the art and science of music and technology*. Oxford University Press.

Guzdial, M. (2003). A media computation course for non-majors. *ACM SIGCSE Bulletin,* 35(3), 104–108.

Heintz, F., Mannila, L., Färnqvist, T. (2016). A review of models for introducing computational thinking, computer science and computing in K-12 education. In: *Proceedings – Frontiers in Education Conference (FIE)*. 1–9 . http://doi.org/10.1109/FIE.2016.7757410

Hubwieser, P., Giannakos, M.N., Berges, M., Brinda, T., Diethelm, I., Magenheim, J., Pal, Y., Jackova, J., & Jasute, E. (2015). A global snapshot of computer science education in K-12 schools. In: *Proceedings of the 2015 ITiCSE on Working Group Reports – ITICSE-WGR '15*. New York, New York, USA: ACM Press, 65–83.

Kafai, Y.B. (2016). From computational thinking to computational participation in K–12 education. *Commun. ACM*, 59(8), 26–27.

Kaschub, M., Smith, J.P. (2016). The big picture: Developing musical capacities. *Music Educators Journal*, 102(3), 33–40.

Selby, C., Woollard, J. (2013). Computational thinking: the developing definition. Available from http://eprints.soton.ac.uk/356481

Simoni, M. (2003). Algorithmic composition: a gentle introduction to music composition using common LISP and common music. *SPO Scholarly Monograph Series*.

Tedre, M., & Denning, P.J. (2016). The Long Quest for Computational Thinking. In: *Proceedings of the 16th Koli Calling Conference on Computing Education Research*. 120–129.

J. Bell is the Specialist Music Teacher at Chisnallwood Intermediate School in Christchurch, New Zealand. She has a MusB (Hons) in composition from the University of Canterbury, is a registered music teacher (LAIRMT) in flute, clarinet and saxophone, and also performs on trombone and guitar. She is involved in teaching, conducting, performing, arranging, and recording music and presents music workshops nationally and internationally.

T. Bell is a professor in the Department of Computer Science and Software Engineering at the University of Canterbury, New Zealand, where he leads the Computer Science Education Research Group. His "Computer Science Unplugged" project is being widely used internationally. He has received awards for his work in computing education including the 2018 ACM SIGCSE Outstanding Contribution to Computer Science Education award. He has been actively involved in the design and deployment of the new digital technologies curriculum in New Zealand schools.