Fostering Computational Thinking in Secondary School through Music
An Educational Experience based on Google Blockly

Adriano Baratè, Andrea Formica, Luca A. Ludovico and Dario Malchiodi
Department of Computer Science, Università degli Studi di Milano, Via Comelico 39, Milan, Italy
{*adriano.barate, luca.ludovico, dario.malchiodi*}*@unimi.it*

Keywords: Music, Education, Visual Programming.

Abstract: We propose a methodology especially conceived to exploit the musical media in order to vehiculate some aspects in the realm of computational thinking to pupils of the lower secondary school (6^{th} to 8^{th} grades). The related activities are based on a visual programming language whose execution generates a melody shown using both its traditionally-notated musical score and its audio reproduction. This language provides the basic programming tools, such as simple and structured variables, iterations and so on. The learning activities are based on challenging small groups of students to solve programming exercises of increasing difficulty.

1 INTRODUCTION

A significant effort in educational research has been recently spent focusing on the study of specific methodologies for teaching concepts from the informatics field since the early school stages. There are two main fronts in this research area:

1. Selecting subjects apt to be explained since the lower secondary or even the primary school levels. See for instance (Hubwieser, 2006; Hromkovič and Steffen, 2011; Saeli et al., 2011);
2. Providing technological tools specifically designed in order to support teachers in their activities. Also in this case the literature offers several cues, such as (Burbaitė et al., 2013; Meerbaum-Salant et al., 2013; Hadjerrouit, 2015)[1].

In this work, explicitly referring to the *algomotricity* learning methodology (Bellettini et al., 2014a), we propose both an informatics topic to be investigated in the lower secondary school, that is from 6^{th} to 8^{th} grade, and a computer-based tool expressly conceived in order to support pupils in learning this topic. Namely, we focus on the description of musical information using a programming language. More precisely, we consider the possibility of teaching basic computational thinking concepts through engaging pupils in writing programs whose output is a prefixed melody. The tool supporting this process is a customization of the Blockly visual programming language, extended in order to deal with the main objects and operators in the realm of traditional musical notation.

The paper is structured as follows: Section 2 briefly summarizes the recent research results in the research area of informatics teaching, with special emphasis on the proposed methodologies. Section 3 illustrates the proposed approach, both detailing its musical and computational aspects and describing the developed software tool to be used in the learning activities, which in turn are depicted in Section 4. Some concluding remarks end the paper.

[1]Note that in this vein we consider only tools expressly conceived for teaching informatics, and not computer-based tools which *generally* support teaching of non-informatics concepts, for instance in the STEM field.

2 RELATED WORKS

After having described the two main educational references of computational thinking and algomotricity, this section highlights the intersection of musical and computational education characterizing the proposed learning activity.

2.1 Computational Thinking

Current research is focusing on those psychopedagogic practices that emphasize the potential of new technologies and their use to motivate young students through active learning. With reference to early education stages, some experts underline the need to define new and engaging learning experiences

Baratè, A., Formica, A., Ludovico, L. and Malchiodi, D.
Fostering Computational Thinking in Secondary School through Music - An Educational Experience based on Google Blockly.
In *Proceedings of the 9th International Conference on Computer Supported Education (CSEDU 2017) - Volume 2*, pages 117-124
ISBN: 978-989-758-240-0

for children, addressing in particular those future job opportunities related to information technology and computing. Many scientific papers that propose coding activities recall the need to develop computational thinking even in primary schools (Sabitzer et al., 2014; Yadav et al., 2014; Kalelioğlu, 2015). An interesting debate on this subject is presented in (Duncan et al., 2014), a paper that impartially collects arguments in favor and against learning coding at a young age.

During a coding activity, students are exposed to *computational thinking* (Wing, 2006), namely a form of reasoning oriented to problem solving which involves abstraction, debugging, remixing and iteration processes (Wing, 2008; Brennan and Resnick, 2012). Computational thinking is in line with the skills expected for 21st-century students, such as creativity, critical thinking and problem solving (Ananiadou and Claro, 2009; Binkley et al., 2012).

The challenge is that young students - while approaching the basic concepts and the cognitive strategies related to coding - can develop logical-cognitive abilities, with positive future effects in terms of meta-knowledge (self-regulation, peer-seeking, problem posing and solving, etc.) and digital skills obtained through playful learning processes (Lillard, 2013).

2.2 Algomotricity

The term *algomotricity* (Bellettini et al., 2014a) refers to an active/kinesthetic teaching methodology (Begel et al., 2004) grounded on the experiential learning theory (Kolb et al., 2001). This methodology has been introduced with the aim of teaching through laboratorial activities some core concepts of informatics also at lower educational grades, meanwhile avoiding common misconceptions identifying the discipline with operational abilities in using software tools such as Web browsers or spreadsheets (Bellettini et al., 2014b).

Each laboratory is typically composed by three phases. In the first one, with reference to the PBL methodology (Hmelo-Silver, 2004), the facilitator[2] describes a problem which pupils should address to, working together in small groups. These groups are engaged in playful activities based for instance on object manipulation, in order to informally introduce the topic under investigation. A second abstract learning phase gives each pupil the possibility to build *its* mental model of that topic. Finally, a computer-based phase ends the activities, also closing the loop with the previous acquaintance of students with applications, thus meeting at least in part their expectations.

The algomotricity approach has been already applied to several informatics concepts, including reasoning on meta-information (Bellettini et al., 2012), programming (Lonati et al., 2015), or recursion (Lonati et al., 2016). In Section 4 we will propose a learning activity based on music operators in an algomotricity context, in order to develop computational-thinking skills.

2.3 Music and Computational Thinking

Many experts agree that music can influence the construction of the young students' personality since it promotes the integration of perceptual, motor, affective, social and cognitive dimensions (Willems, 2011) by relating the basic aspects of human life (e.g., physiological, emotional, and mental spheres) with the basic elements of music (e.g., rhythm, melody, and harmony). The abilities of listening, exploration and analysis are fundamental for the development of general meta-cognitive skills, such as attention, concentration, and control. Through music, young students can develop the aspects of analysis and synthesis, problem posing, argumentation, evaluation, and application of rules (Berkley, 2004; Burnard and Younker, 2004; Kaschub and Smith, 2009; Major and Cottle, 2010).

In the digital era, new technologies and computer-based approaches can influence music learning and teaching processes. A recent and comprehensive review of this subject can be found in (Finney and Burnard, 2010), a work that discusses a range of innovative practices in order to highlight the changing nature of schooling and the transformation of music education. Many researchers, experts and music teachers feel a pressing need to provide new ways of thinking about the application of music and technology in schools. It is necessary to explore teaching strategies and approaches able to stimulate different forms of musical experience, meaningful engagement, creativity, and teacher-learner interactions.

Music can be an effective way to develop computational thinking. A comparison between coding environments and music-oriented programming frameworks unveils similarities and differences. According to (Lye and Koh, 2014), coding environments for young students should foster the development of three dimensions of computational thinking: computational concepts, practices, and perspectives. *Computational concepts* are the conceptual entities that programmers

[2]this term typically designates the equivalent of a teacher in an active learning environment, underlining the fact that learning takes place without *direct* transmission of knowledge, and pupils should be just oriented in their autonomous discovery of concepts.

use, such as variables and loops, in order to solve a problem algorithmically. *Computational practices* are problem-solving practices that occur during the process of programming (e.g., iteration, reuse and remix, abstraction, and modularization). Finally, *Computational perspectives* refer to students' understanding of themselves, their relationship to others, and the world around them.

A suitably designed music-coding environment can achieve all the mentioned goals. Computational concepts – mapped onto musical operators and constructs to manipulate them – will be discussed in Section 3. As it regards computational practices, such an environment allows the exploration of music contents through abstraction processes, as the learning-activity case study discussed in Section 4 will clarify. Moreover, the availability of ad-hoc operators (e.g., iteration) and storage tools (e.g., memory drawers) encourages reusing and remixing of music materials.

Finally, as it regards computational perspectives, an immediate feedback in terms of graphical rendering and audio reproduction improves self-consciousness. Besides, the lesson can be structured to foster cooperative discussion and peer as well as expert review about the coding activity, in order to strengthen the relationship among students and with the teacher. Moreover, the coding environment itself could implement features to allow self-assessment, peer seeking (i.e. searching for the help of a colleague), and peer reviewing (i.e. asking for other students' opinions).

A music-based approach to develop computational thinking has been already explored in other experimentations. Concerning the efforts to promote coding to kids, we can mention programmable robots. In this context we are not generically interested in Music Information Robotics, a branch that has been explored in a number of scientific works, such as (Kapur, 2005), (Solis and Takanishi, 2007) and (Ness et al., 2011). Rather, we are addressing those initiatives aiming to foster the development of computational thinking through music at an early age. For instance, *Play-I* [3] is a robot that can move around, express character using sounds and lights, and respond to a number of external stimuli. Using a tablet with the app, a child can program the robot to play music and dance during the performance. Another example is *Wigl* [4], an interactive educational robot able to hear notes from any instrument and to respond with real-world movements and lights.

Another relevant initiative is *Note Code*, a music programming puzzle game designed as a tangible device coupled with a graphical user interface (Kumar et al., 2015). Tapping patterns and placing boxes in proximity enables programming these entities to store sets of notes, play them back and activate different sub-components or neighboring boxes.

Finally, Ruthmann et al. discussed the experience of an interdisciplinary general education course called Sound Thinking offered to university students (Ruthmann et al., 2010). Even if the recipients and the pedagogical goals are different from our proposal, the adopted approach and technological tools are quite similar.

Consequently, a music coding framework can improve music skills while fostering, but above all it can convey computational thinking even better than other "traditional" coding environments, thanks to the intrinsic characteristics of the musical media.

3 THE PROPOSED APPROACH

This section illustrates the approach guiding the proposed learning activity, in terms of its goals, contents, and also describing an ad hoc software to be used as a support for pupils.

3.1 Goals and Design Choices

As shown by great composers and theorists of the past, a formal and algorithmic approach to music composition and analysis is possible. In recent times, a milestone is the generative theory of tonal music proposed in (Jackendoff, 1985), aiming to uncover a grammar that could explain the human musical mind in a scientific manner comparable to Noam Chomsky's transformational or generative grammar (Chomsky, 2002).

In our framework we introduced both music operators – which can be seen as atomic steps of a generative algorithm to build a music tune – and constructs to manipulate and combine the result of operators.

Due to the educational goal of this initiative, the application domain has been intentionally simplified. First, the score is considered as composed by a single melody, presenting no chords. This is sufficient to catch the melodic and rhythmic aspects of music, not the harmonic ones. Besides, every note (and rest) is quantized: the *quantum* represents the smallest rhythmical value that can be encoded. Longer values can be obtained by tying notes together, thus joining quanta, or adding augmentation dots. This approach is valid for simple tunes, where a coarse quantization is sufficient, but it would show its limits in the representation of tuplets and other complex rhythmic layouts.

[3] https://www.makewonder.com/

[4] http://wiglbot.com/

Finally, as a design choice, all supported operators should be easily understandable and simple to use. In fact, the goal is not to cover all possible compositional processes, but to provide also musically-untrained students with an intuitive tool set to build basic scores.

3.2 Musical Operators

In order to achieve our educational goal, we designed a basic tool set of *music operators* which can be used to create a logic representation of music scores. Consequently, the purpose of such statements is not to draw music symbols on a staff or to play notes, but to generate a sequence of music events which can be parsed by ad hoc applications in order to produce a traditional staff notation or an audio performance.

Music operators are organized in the following categories: 1. *Melodic Operators* (*MO*s) working on pitches; 2. *Rhythmic Operators* (*RO*s) acting on rhythm-related aspects of music events; 3. *Iteration Operators* (*IO*s) implementing loops; 4. *List Operators* (*LO*s) building and manipulating lists of music events. Operators in the first three families are the following:

- *AddElementToScore(*m*)* – A *MO* that adds music event m at the end of the score, where m is either a pitched note or a rest. A pitch can be set in two ways, i.e. by choosing a note name in the range [A ... G] or providing a more accurate definition in terms of note name, octave and accidental;
- *AddElementToScore(*$\mathcal{L}$*)* – A *MO* that appends a list of music events called $\mathcal{L}$, made of any non-null combination of pitches and rests (see the end of this section);
- *AddTiedElementsToScore(*n, m*)* – A variant of *AddElementToScore(*m*)* aiming to insert n consecutive music events, all set to the same pitch and tied together. In music notation, a tie is a curved line connecting the heads of two notes of the same pitch and name, indicating that they are to be played as a single note with a duration equal to the sum of the individual notes' values. In a quantized context, this operator is intended to input longer rhythmical values, and – from this point of view – it belongs both to *MO* and to *RO* families;
- *AddAugmentationDot()* – A *RO* that adds an augmentation dot (i.e. half of the current duration) to the event it is applied to;
- *DiatonicallyTranspose(*m, g*)* – A *MO* that aims to transpose the pitch of music event m by g grades, with $g \in \mathbb{Z}$. For the sake of simplicity, the reference key is C major. Consequently, the diatonic transposition of a D-pitched note by $g = +2$ produces an F-pitched note, and the transposition of a G♯-pitched note by $g = -2$ produces an E♯-pitched note. Please note that many pitched instruments for young students are diatonic, such as most toy xylophones and whistles;
- *ChromaticallyTranspose(*m, h*)* – A *MO* that aims to transpose the pitch of m by h halftones, with $h \in \mathbb{Z}$. The chromatic transposition of a D-pitched note by $h = +1$ produces a D♯-pitched note, and the transposition of a G♯-pitched note by $h = -3$ produces an F-pitched note;
- *Repeat(*X, n*)* – This first case of operator belonging to the *IO* family allows to repeat X for n times, where X can be a single music event m (either a note or a rest) or a list $\mathcal{L}$;
- *RepeatAndDiatonicallyTranspose(*X, n, g*)* – This variant repeats X for n times, applying a diatonic transposition by g grades at each iteration;
- *RepeatAndChromaticallyTranspose(*X, n, h*)* – This variant repeats X for n times, applying a chromatic transposition by h halftones at each iteration.

Lists and related operators require a more detailed discussion. In order to be a type of parameter interchangeable with respect to the single music event, a list must be built, manipulated and stored through ad hoc operators. After saving the list, this object can be passed to some of the above listed music operators like a single pitch or a rest. The set of *LO*s includes, but is not limited to:

- *CreateList(*$\mathcal{L}, X$*)* – A *LO* that creates list $\mathcal{L}$ and initialize it to object X, namely a music event m or an already existing list $\mathcal{M}$;
- *AddElementToList(*$m, \mathcal{L}$*)* – A *LO* that appends music event m to the already existing list $\mathcal{L}$;
- *AddTiedElementsToList(*$n, m, \mathcal{L}$*)* – A *LO* that appends n occurrences tied together of music event m to the already existing list $\mathcal{L}$.

The proposed language contains other aggregated *LO*s which have not been described here for sake of brevity, being their meaning either very easy to understand, or similar to one of the three described *LO*s.

3.3 Music Coding

The proposed framework aims to provide pupils with the opportunity to learn a variety of computational concepts, including variables, functions, and iteration, while they engage in analyzing, composing, and playing music.

First, the idea itself of music event – namely an entity that can be set to a given pitch – recalls the basic concept of variable (in its simplest form) or structure/object (in its more detailed form).

The range of expected values that a music event can take in terms of pitch, octave number and accidental status can be linked to the concept of data type. Specifically, the octave number can be intuitively mapped onto the range of integers, whereas the limited set of values available for note name and accidental status can recall the concept of enumeration. From this point of view, lists of music events can be seen as arrays or dynamic lists.

The use of music operators taking $[0 \dots n]$ parameters is similar to function calls in traditional programming. Please note that the availability of music operators sharing the same name but taking a different number or different types of parameters recalls the concept of function overloading, namely the possibility to create multiple methods with different implementations whose call is disambiguated on the base of the context. It is the case of *AddElementToScore(𝔪)* and *AddElementToScore(𝕃)*, acting on a single music event or a list respectively.

Moreover, in order to foster computational thinking, iteration is a fundamental concept to convey. In this sense, the functions belonging to the *IO* family allow to choose a sequence of generative steps to be repeated for a given number of times. Please note that in music notation there are symbols that recall this approach (e.g., n-bar repeat signs, repeat barlines, and text indications such as “Da capo”).

Finally, the possibility to create, manipulate and rework entire code sections pushes towards the well-known practice of code reuse, presenting a positive impact on time and resources consumption, cost effectiveness, code length and redundancy reduction (Goguen, 1986).

3.4 The Software Tool

In the context of the algomotricity approach discussed in Section 2.2, computer activities can be conducted through a software tool expressly designed and implemented. This tool is based on Google Blockly [5], a visual programming platform. In this environment, the programmer is freed from syntax checking: all language elements are denoted by blocks of different shapes, and the systems allow them to be composed together only if the resulting structure denotes a well-formed language statement. The execution of a program gives directions to one or more interacting *sprites* living on a *stage*, thus the overall system is not dissimilar in its philosophy to Seymour Papert’s LOGO (Papert, 1980).

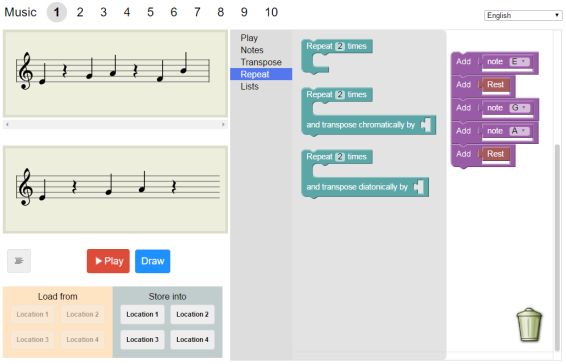

Figure 1: The Blockly-based software tool designed for music coding.

This visual environment has been rethought and applied to music coding, as shown in Figure 1. Now the problem to solve is how to reconstruct a given music tune through a suitable combination of available music operators, and success is achieved when the user’s music score (displayed below) is equal to the one originally proposed (displayed above). The blocks presented in the middle column correspond to music operators described in Section 3.2.

This customized Blockly game is publicly available for free. Even if Web technologies should guarantee multi-platform compatibility, audio rendering of music scores – a fundamental feature to provide feedback – requires non-trivial software installations, including a virtual MIDI synthesizer and a browser compatible with the Web MIDI API (Wilson and Kalliokoski, 2015), which is still a working draft by W3C [6]. For this reason, the tool will be both published as a Web-available framework and distributed to schools and any other interested subject as a Docker package [7]. The latter solution highly simplifies the tool deployment in schools. Indeed, Docker is an open-source project wrapping up in a so-called *container* all the resources needed in order to run one or more processes. This isolation concerns both hardware (CPU, memory, file system, etc.) and software (libraries, tools, code, and so on). The deployment is further simplified since container descriptions can be automatically downloaded from publicly available repositories.

The software is already available for teachers who request it, and it will be published shortly on a repository with GPL-like license.

[5] https://developers.google.com/blockly/

[6] https://www.w3.org/

[7] https://www.docker.com/

4 AN EXAMPLE

As mentioned above, the final goal is to push students towards computational thinking through a music-oriented activity. In practical terms, this assignment is to reconstruct increasingly complex music tunes through a proper combination of available music operators. Achieving this goal requires a mix of analytical skills to be applied to music, knowledge of the available tools (non only their function, but also their limitations and the possibilities to combine them), and even a good amount of creativity. All these ingredients are part of computational thinking.

Algomotricity, a methodology introduced in Section 2.2, fosters multiple skills through an experiential learning approach occurring in three phases: 1) definition of the problem, 2) self-reflection and construction of a mental map of the solution, and 3) hands-on experience. Consequently, even if Blockly games are conceived for self-exploration of programming concepts (and this is virtually possible also for our proposal), the idea is to build a learning activity composed by the mentioned stages and supervised by an expert.

The solutions of the proposed exercises are not unique: some are easier to find (but often require a greater number of steps), some other are smarter (and more compact). The advantages of this approach include:

- The possibility for teachers to modulate the difficulty of an exercise by enabling/disabling some music operators;
- The reduction of students' level of frustration, since a simple solution, roughly based on an imperative programming paradigm, always exists;
- The intrinsic push towards peer confrontation and discussion of different solution strategies, which is an integral part of the educational process;
- The presence of different (although all correct) solutions is a key aspect in active learning, because students can explore the solution space without feeling constrained in following a fixed learning path.

Now an example of assignment is called for, together with a discussion of multiple solution strategies. The music tunes to produce is a *rigaudon* – a form of French baroque dance – composed by Henry Purcell, and its score is shown in Figure 2. A first solution consists in explicitly declaring all the notes to be played. Recalling the music operators listed in Section 3.2, code listing would be:

```
AddElementToScore(C)
AddElementToScore(C)
AddElementToScore(B)
AddElementToScore(A)
AddTiedElementsToScore(2,G)
AddElementToScore(G)
AddElementToScore(G)
AddElementToScore(A)
AddElementToScore(A)
AddElementToScore(B)
AddElementToScore(G)
AddTiedElementsToScore(2,C)
AddTiedElementsToScore(2,G)
AddElementToScore(C)
AddElementToScore(C)
AddElementToScore(B)
AddElementToScore(A)
AddTiedElementsToScore(2,G)
AddElementToScore(G)
AddElementToScore(G)
AddElementToScore(A)
AddElementToScore(A)
AddElementToScore(B)
AddElementToScore(G)
AddTiedElementsToScore(4,C)
```

This strategy, based on invoking as many MOs as notes, is the simplest one, but it fosters computational thinking to a very small extent. For instance, redundancy in program listing is evident, since the first eleven operators are repeated later.

Another way to solve the assignment could be based on nested iterations and diatonic transposition. This proposal requires analytical skills and the development of an algorithmic approach in order to use a smaller number of operators and to make code more compact. The resulting listing would be:

```
AddElementToScore(C)
RepeatAndDiatonicallyTranspose(C,3,-1)
AddTiedElementsToScore(2,G)
RepeatAndDiatonicallyTranspose(
   Repeat(G,2),2,1)
AddElementToScore(B)
AddElementToScore(G)
RepeatAndDiatonicallyTranspose(
   AddTiedElementsToScore(2,C),2,-3)
AddElementToScore(C)
RepeatAndDiatonicallyTranspose(C,3,-1)
AddTiedElementsToScore(2,G)
RepeatAndDiatonicallyTranspose(
   Repeat(G,2),2,1)
AddElementToScore(B)
AddElementToScore(G)
AddTiedElementsToScore(4,C)
```

Finally, another approach which is based on the recognition of redundancy and fully exploits code reuse could employ lists and LOs, as shown in the following listing:

```
CreateList(List1,C)
AddElementToList(C,List1)
AddElementToList(B,List1)
AddElementToList(A,List1)
```

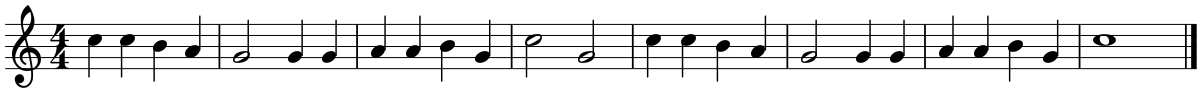

Figure 2: A *rigaudon* composed by Henry Purcell.

```
AddTiedElementsToList(2,G,List1)
AddElementToList(G,List1)
AddElementToList(G,List1)
AddElementToList(A,List1)
AddElementToList(A,List1)
AddElementToList(B,List1)
AddElementToList(G,List1)
AddElementToScore(List1)
AddTiedElementsToScore(2,C)
AddTiedElementsToScore(2,G)
AddElementToScore(List1)
AddTiedElementsToScore(4,C)
```

These solutions for the same assignment are only some of the possible alternatives that may emerge. Please note that peer collaboration during the activity, peer discussion after the activity and a final review phase supervised by experts may be integral parts of the educational path.

The functions that the framework offers to graphically render score symbols on staff and to listen to notes, also during a step-by-step search for solution, provide prompt feedback and may represent effective reinforcement techniques.

5 CONCLUSIONS

In this work we proposed a learning activity for lower secondary schools with the mixed learning objective of focusing both on musical and computational aspects. The activity, rooting on the algomotricity learning methodology, has a twofold purpose: it can be used in order to vehiculate some of the main ingredients of computational thinking (with special emphasis on programming) through the use of traditional musical notation, or conversely it can focus on learning the traditional musical notation from the point of view of formal descriptions. The activity is coupled with an expressly designed software tool, whose pedagogical value lies in the presence of a modular set of musical operators (to be enabled in function of the learning acitivity stage), on immediate visual and audio feedback, and on its customizability (teachers can easily add new learning stages to the existing ones).

In the next future we plan to start a research-action experimentation in schools, in order to validate the methodology and to tune both the learning activities and the proposed software tool.

Authors would like to thank prof. M. Monga for his valuable suggestions in the definition of the virtual architecture used to implement the software tool described in this work.

REFERENCES

Ananiadou, K. and Claro, M. (2009). *21st century skills and competences for new millennium learners in OECD countries*. OECD Publishing.

Begel, A., Garcia, D. D., and Wolfman, S. A. (2004). Kinesthetic learning in the classroom. In *Proc. of the 35th SIGCSE TSCSE*, pages 183–184, New York, USA. ACM.

Bellettini, C., Lonati, V., Malchiodi, D., Monga, M., Morpurgo, A., and Torelli, M. (2012). Exploring the processing of formatted texts by a kynesthetic approach. In *WiPSCE'12 Proceedings of the 7th Workshop in Primary and Secondary Computing Education*, pages 143–144. ACM.

Bellettini, C., Lonati, V., Malchiodi, D., Monga, M., Morpurgo, A., Torelli, M., and Zecca, L. (2014a). Extracurricular activities for improving the perception of informatics in secondary schools. In *Informatics in Schools. Teaching and Learning Perspectives 7th Int. Conf. on Informatics in Schools: Situation, Evolution, and Perspectives, ISSEP 2014. Proceedings*, Lecture Notes in Computer Science, page 161172. Springer International Publishing.

Bellettini, C., Lonati, V., Malchiodi, D., Monga, M., Morpurgo, A., Torelli, M., and Zecca, L. (2014b). Informatics education in italian secondary school. *ACM Transactions on Computing Education (TOCE) Special Issue on Computing Education in (K-12) Schools*, 14(2):15.115.6.

Berkley, R. (2004). Teaching composing as creative problem solving: conceptualising composing pedagogy. *British Journal of Music Education*, 21(03):239–263.

Binkley, M., Erstad, O., Herman, J., Raizen, S., Ripley, M., Miller-Ricci, M., and Rumble, M. (2012). Defining twenty-first century skills. In *Assessment and teaching of 21st century skills*, pages 17–66. Springer.

Brennan, K. and Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American Educational Research Association, Vancouver, Canada*, pages 1–25.

Burbaitė, R., Damaševičius, R., Štuikys, V., et al. (2013). Teaching of computer science topics using meta-programming-based glos and lego robots. *Informatics in Education-An International Journal*, 12_1:125–142.

Burnard, P. and Younker, B. A. (2004). Problem-solving and creativity: Insights from students individual com-

posing pathways. *International Journal of Music Education*, 22(1):59–76.

Chomsky, N. (2002). *Syntactic structures*. Walter de Gruyter.

Duncan, C., Bell, T., and Tanimoto, S. (2014). Should your 8-year-old learn coding? In *Proceedings of the 9th Workshop in Primary and Secondary Computing Education*, pages 60–69. ACM.

Finney, J. and Burnard, P. (2010). *Music education with digital technology*. Bloomsbury Publishing.

Goguen, J. A. (1986). Reusing and interconnecting software components. *Computer*, 19(2).

Hadjerrouit, S. (2015). Exploring the effect of teaching methods on students learning of school informatics. In *Proceedings of Informing Science & IT Education Conference (InSITE)*, volume 201, page 219.

Hmelo-Silver, C. E. (2004). Problem-based learning: What and how do students learn? *Educational Psychology Review*, 16(3):235–266.

Hromkovič, J. and Steffen, B. (2011). *Why Teaching Informatics in Schools Is as Important as Teaching Mathematics and Natural Sciences*, pages 21–30. Springer Berlin Heidelberg, Berlin, Heidelberg.

Hubwieser, P. (2006). *Functions, Objects and States: Teaching Informatics in Secondary Schools*, pages 104–116. Springer, Berlin, Heidelberg.

Jackendoff, R. (1985). *A generative theory of tonal music*. MIT Press.

Kalelioğlu, F. (2015). A new way of teaching programming skills to K-12 students: Code.org. *Computers in Human Behavior*, 52:200–210.

Kapur, A. (2005). A history of robotic musical instruments. In *Proceedings of the International Computer Music Conference (ICMC 2005)*, pages 21–28.

Kaschub, M. and Smith, J. (2009). *Minds on music: Composition for creative and critical thinking*. R&L Education.

Kolb, D. A., Boyatzis, R. E., and Mainemelis, C. *et al.*. (2001). Experiential learning theory: Previous research and new directions. *Perspectives on thinking, learning, and cognitive styles*, 1:227–247.

Kumar, V., Dargan, T., Dwivedi, U., and Vijay, P. (2015). Note code: A tangible music programming puzzle tool. In *Proceedi of the Ninth Int. Conf. on Tangible, Embedded, and Embodied Interaction*, TEI '15, pages 625–629, New York, USA. ACM.

Lillard, A. S. (2013). Playful learning and montessori education. *American journal of play*, 5(2):157.

Lonati, V., Malchiodi, D., Monga, M., and Morpurgo, A. (2015). Is coding the way to go? In *Informatics in Schools. Teaching and Learning Perspectives 8th International Conference on Informatics in Schools: Situation, Evolution, and Perspectives, ISSEP 2015. Proceedings*.

Lonati, V., Malchiodi, D., Monga, M., Morpurgo, A., and Previtali, M. (2016). A playful tool to introduce lower secondary school pupils to recursive thinking. In *Proceedings of 9th International Conference on Informatics in Schools: Situation, Evolution, and Perspectives, ISSEP 2016*, pages 51–52.

Lye, S. Y. and Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for k-12? *Computers in Human Behavior*, 41:51–61.

Major, A. E. and Cottle, M. (2010). Learning and teaching through talk: Music composing in the classroom with children aged six to seven years. *British Journal of Music Education*, 27(03):289–304.

Meerbaum-Salant, O., Armoni, M., and Ben-Ari, M. (2013). Learning computer science concepts with scratch. *Computer Science Education*, 23(3):239–264.

Ness, S. R., Trail, S., Driessen, P. F., Schloss, W. A., and Tzanetakis, G. (2011). Music information robotics: Coping strategies for musically challenged robots. In *Proc. of the 12th Int. Society for Music Information Retrieval Conference (ISMIR 2011)*, pages 567–572.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc.

Ruthmann, A., Heines, J. M., Greher, G. R., Laidler, P., and Saulters II, C. (2010). Teaching computational thinking through musical live coding in scratch. In *Proceedings of the 41st ACM technical symposium on Computer science education*, pages 351–355. ACM.

Sabitzer, B., Antonitsch, P. K., and Pasterk, S. (2014). Informatics concepts for primary education: preparing children for computational thinking. In *Proceedings of the 9th Workshop in Primary and Secondary Computing Education*, pages 108–111. ACM.

Saeli, M., Perrenet, J., Jochems, W. M., Zwaneveld, B., et al. (2011). Teaching programming in secondary school: a pedagogical content knowledge perspective. *Informatics in Education-An International Journal*, 10_1:73–88.

Solis, J. and Takanishi, A. (2007). An overview of the research approaches on musical performance robots. In *International Computer Music Association*.

Willems, E. (2011). *Las bases psicológicas de la educación musical*. Grupo Planeta (GBS).

Wilson, C. and Kalliokoski, J. (2015). Web MIDI API. working draft, W3C. https://www.w3.org/TR/webmidi/.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3):33–35.

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical transactions of the royal society of London A: mathematical, physical and engineering sciences*, 366(1881):3717–3725.

Yadav, A., Mayfield, C., Zhou, N., Hambrusch, S., and Korb, J. T. (2014). Computational thinking in elementary and secondary teacher education. *ACM Transactions on Computing Education (TOCE)*, 14(1):5.