INDERSCIENCE PUBLISHERS
Linking academia, business and industry through research

International Journal of Learning Technology

ISSN online: 1741-8119 - ISSN print: 1477-8386
https://www.inderscience.com/ijlt

Visual programming and computational thinking environments for K-9 education: a systematic literature review

Dimitrios Trakosas, Christina Tikva, Efthimios Tambouris

DOI: 10.1504/IJLT.2023.10056591

Article History:

Received:	21 January 2022
Last revised:	26 April 2022
Accepted:	18 July 2022
Published online:	06 June 2023

Visual programming and computational thinking environments for K-9 education: a systematic literature review

Dimitrios Trakosas, Christina Tikva* and Efthimios Tambouris

University of Macedonia,
156 Egnatia Street, GR-546 36 Thessaloniki, Greece
Email: dimitristrakosas@gmail.com
Email: ch.tikva@uom.edu.gr
Email: tambouris@uom.edu.gr
*Corresponding author

Abstract: Teaching programming and computational thinking to young students has gained increasing attention in recent years. This great attention is attributed partially to the emergence of easy-to-use visual programming environments. These environments help students focus on the logic and concepts of programming and at the same time enhance their engagement. It has been shown that the characteristics of visual programming environments influence students' engagement with programming. However, there is still no systematic investigation of these characteristics. This study aims to provide insights on the characteristics of visual programming environments for K-9 education based on a systematic literature review of 83 empirical studies on K-9 teaching and learning programming. These characteristics are analysed based on the following four levels: a) functional features; b) student experience; c) teacher experience; d) disadvantages. Finally, herein we discuss the features that a programming environment for K-9 education could have to improve the experience of students and teachers.

Keywords: visual programming environments; computational thinking environments; K-9 education.

Reference to this paper should be made as follows: Trakosas, D., Tikva, C. and Tambouris, E. (2023) 'Visual programming and computational thinking environments for K-9 education: a systematic literature review', *Int. J. Learning Technology*, Vol. 18, No. 1, pp.94–121.

Biographical notes: Dimitrios Trakosas is a graphic designer and a children's book, comic and application illustrator in Thessaloniki, Greece for the last nine years. He holds a Bachelor's degree in Fine Arts from the University of Ioannina, an MSc in Graphic Design from the University of Lancaster, UK and an MSc in Information Systems from the University of Macedonia, Thessaloniki, Greece. He is a PhD candidate at the University of Ioannina and his research interests include graphical interfaces, picture books and computer and arts education in K-12.

Christina Tikva is a Computer Science Teacher at Secondary Education in Thessaloniki, Greece, for the last 13 years. She holds a Bachelor's degree in Applied Informatics from the University of Macedonia and an MSc from

Aristotle University of Thessaloniki, Greece. She is a PhD candidate at the University of Macedonia and her research interests include computational thinking and programming in K-12 and higher education.

Efthimios Tambouris is an Assistant Professor at the Applied Informatics Department at the University of Macedonia, Thessaloniki, Greece. Before that, he served at Research Centers CERTH/ITI and NCSR 'Demokritos' as well as the IT industry. He holds a Diploma in Electrical Engineering from the National Technical University of Athens, Greece, and an MSc and PhD from Brunel University, UK. During the last 20 years, he has initiated and managed several research projects, e.g., LLP EATrain, Erasmus+ ODEdu, PBL_LA and flip2G. He has also participated in numerous research projects, service contracts, and standardisation activities. He has more than 150 publications in e-government, e-participation, e-learning and information systems.

1 Introduction

Recently, teaching of programming to young students has gained increasing interest worldwide (Moreno-León et al., 2016). This is evident from the growing number of studies (e.g., Keane et al., 2019) focusing on teaching and learning programming to young students (Zhang and Nouri, 2019). Researchers in these studies (e.g., Fokides, 2017b; Kalelioğlu, 2015) argue that programming and algorithmic skills should be developed at an early age to lay the foundation for the future, as they enhance the scientific skills of K-9 students (Weintrop et al., 2015). To this end, many countries around the world have reformed their curricula to teach programming to younger students (Kalelioğlu, 2015).

Lye and Koh (2014) attribute the recent interest in teaching and learning programming for young students to the availability of easy-to-use visual programming environments. Their easy-to-use design is due to the visual characteristics, the provision of phrase library, the drag-and-drop mechanism and the fact that the blocks are more legible than plain text (Lin and Weintrop, 2021). These tools also prevent syntax errors and encourage students to focus on programming logic and concepts (Fronza et al., 2017; Lye and Koh, 2014; Sengupta et al., 2013; Tikva and Tambouris, 2021). In addition, visual programming environments focus on design and creation (Grover and Pea, 2013a), enhancing student engagement in programming by providing fun, motivation, enthusiasm (Lazarinis et al., 2018; Sáez-López et al., 2016). As an interactive technology, they can also enhance expressiveness (Papadakis, 2022). They are based on the constructionist tradition that emphasises on the self-directed learning through art, games and interactive stories (Weintrop and Wilensky, 2015). Such engaging and fun environments can lead young students to want to learn text-based programming in the future (Lin and Weintrop, 2021). In contrast, text-based programming environments are not considered developmentally appropriate for younger students in K-9 education who are in the early stages of learning to read (Papadakis, 2021). Due to the aforementioned characteristics, visual programming environments are often exploited in studies involving young learners (e.g., Kalelioğlu 2015; Moreno-León et al., 2016; Sáez-López et al., 2016; Zhong et al., 2016).

However, previous research reveals that specific characteristics of programming environments such as gaming, provision of interaction and reward systems may affect students' engagement (Hershkovitz et al., 2019). Therefore, it is important to investigate the characteristics of these environments, as well as the user experience they offer, in order to provide a holistic overview of the programming and computational thinking tools aimed at young students.

The development of relevant research has led to the publication of reviews that investigate programming tools and technologies used in early years education. These efforts (e.g., Ching et al., 2018; Tang et al., 2019) mainly provide an overview of available tools and technologies by classifying them into general categories. However, apart from documenting and classifying these tools, there is a lack of investigation into the specific characteristics of visual programming environments used for teaching and learning programming in early years of education.

This work aims to provide insights into the characteristics of visual programming environments used in K-9 education, by reviewing empirical studies focusing on K-9 teaching and learning programming and computational thinking. The characteristics of visual programming environments are analysed based on the following four levels:

a functional features

b student experience

c teacher experience

d disadvantages.

2 Background work

The accumulation of previous research has led to the publication of relevant reviews (e.g., Ching et al., 2018) with the aim of providing an overview of programming and computational thinking tools and environments aimed at young students. Additionally, several literature reviews (e.g., Buitrago Flórez et al., 2017; Grover and Pea, 2013a; Hsu et al., 2018; Lye and Koh, 2014) include topics such as programming tools and environments, among others. Most of these reviews focus on issues such as the role of these tools in learning programming and developing computational thinking, as well as recording the tools used for this purpose in previous studies.

Hsu et al. (2018) found that the tools used for teaching and learning computational thinking include programming environments, computer games, robots, board games, instant response system and videos. Ching et al. (2018) reviewed technologies aimed at developing computational thinking for young students by analysing them based on:

a the design of the tools (agents' type and manipulatives occurrence)

b their pedagogical possibilities regarding the concepts (sequence, loops, conditionals, operators) that students can learn.

To this end, they classify the aforementioned technologies in programming toys, board games, robot kits, programming applications/websites and animation/game development tools. In the same line, Buitrago Flórez et al. (2017) described the most powerful and useful tools for novice programming learners and classify them in Virtual approaches

(Scratch, Logo, Python), Real life experiences (LEGO®, LEGO® Mindstorms) and Game programming. Lin and Weintrop (2021) defined the approaches used in transitioning young learners from block-based to text-based programming environments.

Grover and Pea (2013a) highlighted some of the features that effective computational thinking tools should have, including low floor and high ceiling. This means that tools should enable novice learners to easily create working programs; while at the same time must be powerful enough to create advanced programming projects. The low floor feature, as pointed out by Lin and Weintrop (2021), should be the main goal of all visual programming environments. Other studies (e.g., Papadakis, 2022; Resnick and Silverman, 2005) also emphasise the importance of low floor-high ceiling. Other significant features are the iterative exploration of computational thinking, the elimination of syntax errors, the motivation and engagement they offer and the ability to be used to bridge the gender gap in the programming field.

Several studies support that visual programming tools allow students to better understand the principles of programming and computational thinking as they avoid unnecessary syntax such as the use of parentheses, questionmarks, etc. Additionally, students in most cases place code in the editor using drag and drop functionality. In these ways, students are helped to focus on the logic and concepts of programming (Grover and Pea, 2013a; Lye and Koh, 2014) and consequently develop new expressive and emotional ways (Papadakis, 2021). Several studies (e.g., Fokides, 2017a; Grover and Pea, 2013a; Hsu et al., 2018) also emphasise that teachers find visual programming more appealing to students. This is especially true for young students involved in programming.

Visual programming environments can provide significant opportunities for learning programming and developing computational thinking to young students. However, investigation of their characteristics is still missing. This study aims to provide insight into visual programming environments' characteristics and therefore could help educators identify the most appropriate environment in relation to their and their students' needs.

3 Method

3.1 Study goal and research questions

This study aims to provide insight into visual programming environments' characteristics aimed at K-9 education. To this end, the following research questions are answered.

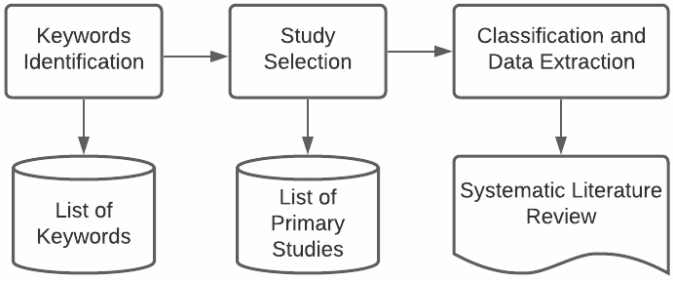
RQ1 What are the visual programming environments aimed at K-9 education?

RQ2 What are the characteristics of the visual programming environments aimed at K-9 education?

3.2 Procedure

A systematic literature review based on Webster and Watson's (2002) methodology was applied. The steps and results of the three phases of the procedure are presented in Figure 1.

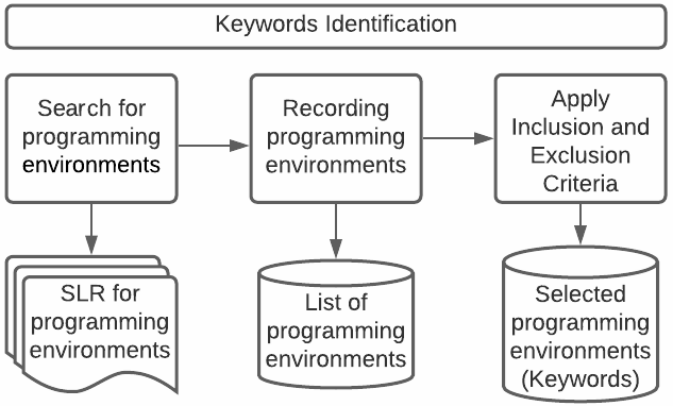
Figure 1 Procedure



3.2.1 Keywords identification

During this phase, we identify the programming environments applied in K-9 education that serve as keywords for our search strategy. Figure 2 presents the steps of the Keywords Identification phase and the related results.

Figure 2 Steps of the keywords identification phase and related results



We started by doing a manual search in journals for reviews regarding programming and computational thinking in K-9 education. We searched for reviews dating back to 2006 when Wing (2006) re-introduced 'computational thinking'. We then applied the following inclusion criteria to the search results:

a the main educational focus of the review is K-9 education

b the scope of the review includes tools and environments for programming and computational thinking.

Since only one review met the first criterion while did not meet the second, we replaced the first inclusion criterion with the following: the main educational focus of the review is K-12 education. Subsequently, we listed the programming environments included in the selected reviews. Finally, we applied the following inclusion and exclusion criteria to the recorded programming environments.

Inclusion criteria:

- *Active user:* programming environments that are designed to encourage students to be creators of programming artefacts and not just consumers. This means that the final derivative of the process of learning is a functional program, a game, an animation or a programming artefact that a third person can use as a consumer (Dunjohn, 2013).
- Visual programming: programming environments that include visual programming.
- *Exclusion criteria:* programming environments that include an electronic physical agent and manipulatives for programming, robot kits, microcontrollers and board games.

3.2.2 Study selection

We searched for studies in the following scientific databases: ACM digital library, IEEE Xplore, ScienceDirect, ResearchGate and Google Scholar using as keywords the selected programming environments derived from the Keywords Identification phase described in Section 3.1.1. We then checked the results by reading all the titles and abstracts and removed studies that were not written in English language or we could not have full access to them. Finally, we scrutinised the remaining studies against inclusion and exclusion criteria presented in Table 1.

Table 1 Inclusion and exclusion criteria

Inclusion criteria	*Exclusion criteria*
Journals, conferences or books.	Theoretical research and opinion articles.
Empirical research where participants are K-9 students or teachers.	Studies that focus on grades greater than 9th grade, higher or adult education.

3.2.3 Classification and data extraction

During this step, we applied content analysis to the list of selected primary studies, as well as inspected the programming environments and their official websites. For each case we recorded the characteristics of the programming environments included in the case. Subsequently, we grouped them into the following four levels:

a functional features

b advantages in relation to the user experience for students

c advantages in relation to the user experience for teachers

d disadvantages.

4 Results

4.1 Visual programming environments targeting K-9 education

In order to identify the programming environments targeting K-9 education we followed the procedure describe in Section 3.2.1. The search resulted in 12 reviews from which the following three (Ching et al., 2018; Grover and Pea, 2013a; Lye and Koh, 2014) met the

inclusion and exclusion criteria. We then, recorded 39 programming environments listed in the aforementioned reviews and removed three duplicates.

Finally, the following 11 programming environments met the inclusion and exclusion criteria: AgentCubes, AgentSheets, Alice, Code.org, Game Maker, Hopscotch, Kodu, MIT App Inventor, Scratch, Snap!, Tynker

4.2 Characteristics of visual programming environments

4.2.1 Overview

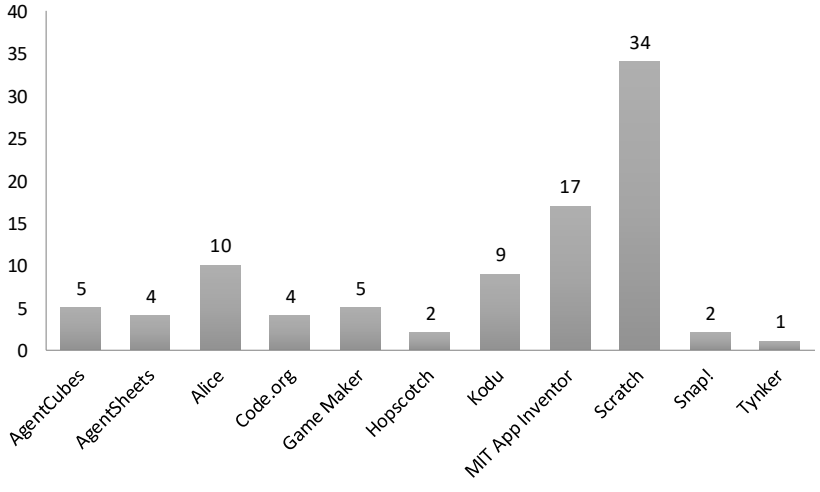
Study selection phase is described in Section 3.2.2. Initially, the searches resulted in 225 studies. Finally, the review includes 83 primary studies that met the inclusion and exclusion criteria. In addition, the official websites of the programming environments are also included in the review. The total cases are presented in Appendix.

Most of the selected studies (67.47%) apply qualitative research methods, but without much difference with those that use quantitative research methods (59.03%). A 26.5% percentage of the selected studies use both qualitative and quantitative methods. Table 2 presents the classification of studies by followed method.

Table 2 Method followed

Research method	*Case (id)*	*Sum*
Qualitative research	C1, C4, C6, C9, C12, C13, C15, C16, C20, C21, C25, C27, C28, C31, C32, C34, C38, C40, C41, C46, C47, C50, C55, C56, C58, C59, C62, C63, C66, C68, C70, C76, C77, C78	34
Quantitative research	C2, C5, C7, C8, C10, C11, C17, C23, C24, C33, C37, C44, C45, C48, C49, C52, C60, C61, C64, C65, C67, C69, C71, C73, C74, C80, C82	27
Both	C3, C14, C18, C19, C22, C26, C29, C30, C35, C36, C39, C42, C43, C51, C53, C54, C57, C72, C75, C79, C81, C83	22

Figure 3 Interventions per programming environment



The programming environment most used in the selected studies is Scratch (34 interventions) followed by MIT App Inventor (17 interventions), Alice (ten interventions) and Kodu (nine interventions). The remaining programming environments appear in less of five interventions in the selected studies. The number of interventions per programming environment is presented in Figure 3.

4.2.2 Functional advantages of programming environments

Investigation of the selected studies reveals the following functional advantages of visual programming environments targeting K-9 students: object-oriented programming, visual to text-based language translator, open-source code, online project library, connectivity with other programming environments, and connectivity with external agents. Table 3 presents the classification of programming environments in each functional advantage.

Table 3 Functional advantages

Advantages	*Programming environment*	*Case (id)*
Object-oriented programming	AgentCubes, AgentSheets, Alice, Game Maker, Hopscotch, Kodu, MIT App Inventor, Scratch, Snap!, Tynker	C1, C3, C6, C9, C10, C11, C23, C24, C26, C27, C36, C41, C47, C51, C53, C55, C58, C60, C61, C68, C71, C73, C74, C75, C76, C77, W2, W4, W10
Visual to text-based language translator	Alice, Code.org, Game Maker, Tynker	C38, C41, W2, W3, W4, W10
Open-source code	Code.org, MIT App Inventor, Scratch, Snap!	W3, W7, W8, W9
Online project library	AgentCubes, AgentSheets, Hopscotch, Kodu, MIT App Inventor, Scratch, Snap!	C6, C13, C19, C52, C55, C61, C63, C73, W1, W5, W6, W9
Internet security	Hopscotch	W5
Connectivity with other programming environments	Hopscotch, Tynker	W5, W10
Connection with external agents	Hopscotch, MIT App Inventor, Scratch	C19, C25, C31, C50, C63, C70, C71, C72, W5, W7, W8

Ten of the 11 programming environments follow the principles of object-orientated programming by allowing users to apply classes, abstraction, inheritance and encapsulation. Only five of them have methods that are either ready-made or allow users to make their own (AgentCubes, AgentSheets, Alice, MIT App Inventor, Snap!).

Four of the listed programming environments include visual to text code translation. This feature allows translation of visual programming blocks into text programming languages as follows: Java (Alice), JavaScript (Code.org, Tynker), Python (Tynker),GameMaker (Game Maker Language).

Only four of the selected programming environments are open-source. Open-source allows programming environments to be improved and customised according to existing needs, thus making a programming environment useful to almost everyone, as new or existing features can be added or changed.

Seven programming environments include an online project library. Allowing students to upload their projects, which can then be downloaded by other students and

worked on or improved upon can provide additional motivation and inspiration. It can also increase collaboration between students in the classroom and the wider online community (Papadakis et al., 2014; Resnick et al., 2009) as well as give an open-source character to students' creations (Perdikuri, 2014). However, only Hopscotch provides Internet Security features for young primary school students. Internet Security applies to cases where an online project accepts feedback from other users, checking whether there is a risk of cyber bullying.

Connectivity with other programming environments allows easy transfer of projects between environments. For example, Hopscotch and Tynker provide the ability to import projects made with Scratch (Dunjohn, 2013). This function, of course, is difficult to achieve for all environments as they do not have similar functions, or even the operating systems they run on are different.

4.2.3 Advantages related to the user experience of teachers

The investigation of the selected studies reveals the following advantages related to the user experience of teachers: easy to learn/teach, multidisciplinary/intersectionality, classroom management tool, real-time project analysis or grading, online community, and support material. Table 4 presents the classification of programming environments in each advantage regarding the user experience of teachers.

Table 4 Advantages related to the user experience of teachers

Advantages	*Programming Environment*	*Case (id)*
Easy to learn / teach	AgentSheets, Game Maker, Kodu, MIT App Inventor, Scratch, Snap!	C6, C7, C12, C18, C20, C24, C26, C38, C42, C47, C48, C49, C51, C56, C57, C58, C59, C63, C73, C76
Multidisciplinary / intersectionality	Alice, Code.org, Game Maker, Hopscotch, Kodu, MIT App Inventor, Scratch, Snap!, Tynker	C11, C12, C14, C15, C16, C17, C18, C20, C24, C26, C32, C41, C43, C50, C51, C52, C57, C58, C59, C62, C68, C69, C73, C80, C83, W2, W5, W10
Classroom management tool	Code.org, Hopscotch, Tynker	C43, W3, W5, W10
Real-time project analysis or grading	AgentCubes[1], Tynker	C9, W10
Online community	Code.org, Game Maker, Hopscotch, Kodu, Scratch, Snap!, Tynker	C13, C60, C68, C82, C83, W3, W4, W5, W6, W8, W9, W10
Supporting material	Alice, Code.org, Game Maker, Hopscotch, Kodu, MIT App Inventor, Scratch, Snap!, Tynker	C2, C43, C56, C68, C83, W2, W4, W5, W6, W8, W9, W10

Note: This feature is only supported in the online version of AgentCubes

Source: Basawapatna et al. (2015)

For six of the 11 programming environments there are reports of ease of learning by the teachers themselves. Consequently, when teachers can easily learn a programming environment, they can teach their students more easily. It is easier to learn and teach a programming environment when it avoids the need to memorise complex syntax

(Kalelioğlu and Gülbahar, 2014; Lazarinis et al., 2018; Morelli et al., 2010). Additional ease can be provided by the hypertonicity of the blocks while the code is executed as it happens in Code.org and Scratch (Papadakis et al., 2014).

Interdisciplinary features are observed in nine programming environments. Interdisciplinarity can help introduce programming to students through courses that are not directly related to programming (Lewis, 2010; Rodger et al., 2010).

Only three out of ten programming environments have a class management tool. Creating an online classroom and having it managed by teachers is a very powerful tool for both teachers and students. Important functions are the possibility of enrolling and accepting students, receiving detailed reports on student progress (Kalelioğlu, 2015) and creating courses and exercises in addition to the ready-made ones offered. Another function considered important is the automatic analysis and advice to teachers (Basawapatna et al., 2015) or, as in Tynker, the automatic correction and grading of students' projects. In addition, a classroom management tool feature can increase collaboration between students and enhance work outside the classroom (Kalelioğlu, 2015).

Seven programming environments have an official online community. An active online community can help teachers exchange opinions and ideas, lesson plans, solve questions with the help of other teachers who are currently online, and also find tutorials and educational materials (Resnick et al., 2009).

Finally, nine programming environments provide teachers with supporting material, such as exercise booklets, instructional videos and syllabi for programming or other courses.

4.2.4 Advantages related to the user experience of students

The investigation of the selected studies reveals the following advantages related to the user experience of students: Focus on programming concepts, computational thinking development, creativity, development of problem-solving skills, code blocks categorisation by colour, ability to create new blocks of code, immediate explanation of code blocks, instant code feedback, hypertension of corresponding blocks of code during execution, automatically prevent incorrect code input, translated into other languages, low floor/high ceiling, create or import of custom graphics, motivation for teamwork, additional incentives for use. Table 5 shows the classification of programming environments in each advantage in terms of students' user experience.

Ten of the 11 programming environments have code-generating mechanisms designed in such a way that students can focus on the concepts and logic of programming and not get stuck in writing code that can frustrate them (Cooper, 2010; Cooper et al., 2003; Costa and Miranda, 2016; Sykes, 2007). As Repenning (2017) mentions, in the complex mechanisms of constructing a function, there must be a way to simplify it so that students do not get bored.

All programming environments focus on the development of computational thinking. Its development through a programming environment can be achieved through interactive simulations, testing of the theories taught, visualisation of the actions taken to solve a problem and more generally through the exploration of other scientific fields through programming (Basawapatna et al., 2013a; Lazarinis et al., 2018). Ten of them particularly focus on the development of students' problem-solving abilities.

Table 5 Advantages related to the user experience of students

Advantages	*Programming Environment*	*Case (id)*
Focus on programming concepts	AgentCubes, AgentSheets, Alice, Code.org, Game Maker, Hopscotch, MIT App Inventor, Scratch, Snap!, Tynker	C2, C4, C5, C7, C11, C13, C16, C17, C18, C24, C25, C30, C33, C35, C36, C38, C41, C43, C44, C45, C47, C48, C49, C50, C51, C52, C53, C55, C56, C58, C61, C64, C67, C69, C70, C71, C72, C73, C74, C75, C83, W2, W10
Computational thinking development	AgentCubes, AgentSheets, Alice, Code.org, Game Maker, Hopscotch, Kodu, MIT App Inventor, Scratch, Snap!, Tynker	C1, C3, C4, C7, C9, C10, C13, C17, C18, C20, C23, C24, C29, C30, C32, C35, C36, C38, C39, C41, C43, C44, C47, C48, C53, C55, C56, C58, C61, C62, C64, C67, C69, C70, C71, C72, C74, C76, C77, C81, C82, W2, W10
Creativity	AgentCubes, AgentSheets, Alice, Code.org, Game Maker, Hopscotch, Kodu, MIT App Inventor, Scratch, Snap!, Tynker	C3, C4, C11, C13, C24, C31, C36, C38, C41, C42, C43, C47, C51, C53, C56, C62, C63, C72, C73, C75, C76, C79, C80, C81, C82, W1
Development of problem-solving skills	AgentCubes, AgentSheets, Alice, Game Maker, Hopscotch, Kodu, MIT App Inventor, Scratch, Snap!, Tynker	C1, C3, C4, C5, C7, C12, C17, C19, C23, C24, C25, C30, C32, C36, C41, C42, C44, C47, C48, C54, C56, C57, C58, C61, C63, C72, C74, C76, C77, C79, C81, C82
Code blocks categorisation by colour	Code.org, Hopscotch, Kodu, MIT App Inventor, Scratch, Snap!, Tynker	C13, C47, C53, C60, C61, C71, C72, C73, C76, C78, W3, W5, W6, W8, W9, W10
Ability to create new blocks of code	AgentCubes, AgentSheets, Scratch, Snap!	C33, C51, W1, W8, W9
Immediate explanation of code blocks	Kodu, Scratch, Snap!, Tynker	C51, C53, C76, W9, W10
Instant code feedback	Code.org, Game Maker, Tynker	C41, W3, W4, W10
Hypertension of corresponding blocks of code during execution	Code.org, Kodu, Scratch, Snap!	C60, W3, W6, W9
Automatically prevent incorrect code input	Alice, Kodu, Scratch	C18, C49, C60, C68, C75, C76, C77, C78, C82
Translated into other languages	Code.org, Hopscotch, MIT App Inventor, Scratch, Snap!	W3, W5, W7, W8, W9
Low floor / high ceiling	AgentCubes, AgentSheets, Alice, Code.org, Game Maker, Hopscotch, Kodu, MIT App Inventor, Scratch, Snap!, Tynker	C6, C18, C31, C34, C38, C45, C47, C48, C58, C63, C68, C74, C76, C82, W1, W2, W3, W9, W10

Table 5 Advantages related to the user experience of students (continued)

Advantages	*Programming Environment*	*Case (id)*
Create or import of custom graphics	AgentCubes, AgentSheets, Alice, Code.org, Game Maker, Hopscotch, MIT App Inventor, Scratch, Snap!, Tynker	C11, C12, C21, C25, C36, C51, C53, C60, C62, C68, W1, W2, W3, W5, W9, W10
Motivation for teamwork	AgentSheets, Alice, Code.org, Game Maker, Kodu, MIT App Inventor, Scratch, Snap!	C6, C14, C23, C24, C25, C26, C28, C29, C32, C37, C38, C39, C41, C42, C43, C44, C54, C58, C60, C68, C80
Additional incentives for use	AgentCubes, AgentSheets, Alice, Code.org, Game Maker, Hopscotch, Kodu, MIT App Inventor, Scratch, Snap!, Tynker	C1, C2, C3, C4, C5, C6, C7, C8, C11, C12, C13, C14, C15, C16, C20, C24, C25, C26, C33, C34, C35, C36, C38, C40, C41, C43, C46, C47, C48, C50, C52, C53, C56, C58, C59, C60, C62, C64, C65, C67, C69, C70, C72, C74, C75, C76, C79, C80, W10

Focusing on some of the more technical features that aid learning, seven programming environments were found to provide block categorisation by colour according to their type of functionality.

Only four of the 11 programming environments allow students to create their own blocks of code and add them to the code library according to their needs. Therefore, students are not limited to using only predefined blocks. At the same time, by creating new solutions and blocks, students' creativity is enhanced (Repenning and Sumner, 1995).

Instant explanation of the ready-made blocks of code in the library is offered by four programming environments. In Scratch, when students click on the desired block, they discover its function as it is executed on the screen (Maloney et al., 2008) while in Tynker, an explanatory video is displayed. Kodu has descriptive tiles (WHEN-DO), so practically students can immediately see what a piece of code can do (Aggarwal et al., 2018). In addition, three programming environments give direct feedback on the generated code after its execution.

In addition, highlighting the code snippet while it is running can help students understand it better. This feature, which is only available at Code.org, Kodu, Scratch and Snap!, can be very useful as it facilitates the control of large programs (Papadakis et al., 2014).

Three programming environments do not allow students to enter a piece of code that is incorrect (Cooper, 2010; Cooper et al., 2003; Costa and Miranda, 2016; Resnick et al., 2009; Sykes, 2007). However, this function is controversial, as according to some studies, (e.g., Cooper et al., 2003; Sykes, 2007) it does not allow students to learn from their mistakes.

All of the selected programming environments are designed to be equally interesting to both beginners and advanced learners, as they allow beginners to learn relatively easily, while providing experienced users with features and mechanisms that allow them to create more complex programs. Thus, the risk of frustration for beginners and the boredom for students with previous programming experience can be avoided (Basawapatna et al., 2013b; Ioannidou et al., 2009; Repenning, 2012).

In ten programming environments students can design or import custom graphics based on their own needs. This process can increase their interest (Resnick et al., 2009). However, it should not be too complicated and difficult to avoid the opposite effect (Remshagen et al., 2018; Rodger et al., 2010).

Eight programming environments provide some motivation to increase students' teamwork and collaboration. For example, Alice encourages students to unite their worlds and create larger or online libraries with comments received from other students. Other environments (e.g., Scratch, Tynker) encourage students to create group projects (Chuter, 2016; Cooper et al., 2003; Padlipsky, 2018; Werner et al., 2012).

In addition, all 11 programming environments provide additional incentives for students. Some examples are the transfer and execution of student projects on everyday objects, such as smartphones (Grover and Pea, 2013b; Papadakis et al., 2014; Perdikuri, 2014), the use of the reward system with the distribution of points or stars after the completion of the project, the use of familiar objects in projects (Cooper, 2010; Empson, 2014; Kalelioğlu, 2015; Sykes, 2007) and the provision of everyday life problems to be solved (Trory et al., 2018). In addition, the combination of the reward system with the simultaneous existence of multiple solutions where the optimal solution gets more points, can increase motivation (Kalelioğlu, 2015) but also competitiveness in the classroom (Gedik et al., 2017).

4.2.5 Disadvantages of the studied programming environments

Examination of the selected studies reveals the following disadvantages: future difficulty in logical connection with real (text-based) programming, hard to understand programming language, difficult to detect errors or relations between objects, repetition, disorientation from programming, limited graphics, mandatory Google account, bugs/problems, quite complicated, especially at younger ages, feedback error messages can discourage the students. Table 6 presents the classification of programming environments in each disadvantage.

Particular attention should be given to how programming is introduced in K-9 education. If too much emphasis is placed on non-programming elements, such as animation, students may in the future find it difficult to relate the knowledge they have acquired to text programming, as observed in Alice (Lewis, 2010). In the same line, Maloney et al. (2008) point out that the students who participated in their study were distracted from learning programming and its concepts by things that are too impressive to the child's eye, such animations in Scratch. On the other hand, in the studies of Kalelioğlu and Gülbahar (2014) and Fokides (2017a), students and teachers complained about the limited graphics provided in Scratch.

The level of difficulty of the respective programming environment should be also taken into account. For example, Repenning (2017) found that the programming language of AgentCubes and AgentSheets was extremely difficult for the elementary school students who participated in his study, even the most eager of them. In addition, in the aforementioned programming environments and App Inventor, debugging proved to be very difficult especially in large programs (Monteiro et al., 2017; Papadakis et al., 2014).

Table 6 Disadvantages

Disadvantages	*Programming environment*	*Case (id)*
Future difficulty in logical connection with real (text-based) programming	Alice	C49
Hard to understand programming language	AgentCubes, AgentSheets	C55
Difficult to detect errors or relations between objects	AgentCubes, AgentSheets, MIT App Inventor	C55, C60
Forgotten hidden objects can result to underperforming programs	AgentCubes, AgentSheets	C55
Repetition	Scratch	C60
Disorientation from programming	Scratch	C53
Limited graphics	Kodu, Scratch	C3, C26, C42, C66
Mandatory Google account	MIT App Inventor, Tynker	C63, W10
Bugs / Problems	AgentCubes, AgentSheets, Alice, MIT App Inventor, Tynker	C19, C55, C63, C75
Quite complicated, especially at younger ages	Game Maker	C41
Feedback error messages can discourage the students	Game Maker	C41
Available only on one platform	Hopscotch	C34, C74, C83
Mandatory Software Installation	AgentCubes, AgentSheets, Kodu	C59, W1
Working with multiple screens	MIT App Inventor	C57

Some other problems regarding the use of the selected programming environments include:

- Slow or sudden loss of connection to the main server (MIT App Inventor).
- Sudden program errors leading to project loss (Alice).
- Malfunction of the recycling bin (MIT App Inventor).
- Unable to save large projects (Alice).
- Missing ‘undo’ function (MIT App Inventor).
- Limited version on mobile devices (Tynker).
- Overlay elements without indication, so only the top element is visible (AgentCubes, AgentSheets).
- Problems with the simultaneous use and real time updating of projects by two or more students. This was observed during the simultaneous operation of Scratch and specifically during saving, where the user who saves last does the work of the previous one on the same project, to be lost (Remshagen et al., 2018).

5 Discussion and recommendations

The analysis of the selected programming environments targeting K-9 education highlights important features that could enhance the learning and teaching of computational thinking and programming at K-9 education but also shortcomings. In the following paragraphs we discuss these features as well as their presence in the selected environments.

The selected programming environments use visual programming. This type of programming helps beginners become familiar with programming and easily develop the skills needed to create complex programs. In addition, these environments are suitable for the entire age range of students in K-9 education with the aim of developing problem-solving skills, creativity, collaboration and participation. They also enable students to develop their own projects involving emotional and cognitive processes. In this way, learning programming becomes more accessible and exciting for students. This is consistent with studies (e.g., Dunjohn, 2013, Sáez-López et al., 2016) showing that students get excited when they create their own games. However, if the tools are too complicated, then the risk of abandonment and frustration with programming increases.

The results suggest that increasing student motivation is of significant importance. All 11 programming environments include incentive features for students. Besides the interest in the final project being created, rewards and unlocking additional features could be additional ways to increase motivation. Additional characters could also be used to engage students. Other ways include using high scores and leader boards. The most important motivation, however, is for students to be involved in problem solving with examples from their own reality that are important to them.

Open-source environments can be customised to the students' and teachers' needs in order to offer specialised motivation. In this way, an open-source environment has the potential to be constantly improved with new features. However, only four of the selected programming environments are open source, with two of them being the most popular among researchers in the selected primary studies (MIT App Inventor and Scratch). Alice which is the third most popular application in the selected studies has only a few open-source modules. This suggests the need for open-source programming environments that can be configured with the aim of further improving them and consequently improving the learning process.

Regarding the functional characteristics of the environments, a gap in compatibility between different environments was observed. In addition, although visual block programming favours the use of these environments on mobile devices, few of them include the feature of connectivity with external devices such as GPS and sensors.

The use of these environments from an early age increases the need for security. However, this is the biggest gap that has been observed, especially with regard to the creations that students upload to online libraries. There is a need for greater control over how students upload their creations, who has access to these online libraries and to what extent each user can access them.

From the teacher's point of view, the analysis revealed important characteristics found in the majority of the selected environments (9 out of 11). Based on this, it is proposed that programming environments aimed at K-9 education should include the following features. First, they should be easy to use even by teachers with no previous programming experience. Second, they must be able to be used in subjects besides Computer Science, especially in lower grades where the goal is to develop computational

thinking and problem-solving skills rather than coding skills. Third, the environments should provide various supporting materials for teacher training and use in the classroom.

A classroom management tool is an additional feature that helps teachers to better organise their lessons and monitor students' progress, identifying where they are having difficulty and providing them with more focused assistance. The existence of this feature did not prove to be very widespread in the selected environments, while its individual functions such as automatic grading and student feedback are even more limited.

Various factors, such as educational objectives, gender and age of the students, determine which of the characteristics described in the results section can be considered of major importance. Therefore, teachers are likely to consider different characteristics as important based on the specific purposes of their educational process. For example, teachers who aim to prepare their students for text programming languages are expected to focus on programming environments that include an optical to text-based language translator. The present study can therefore facilitate teachers in selecting the appropriate environments for them by presenting and categorising their characteristics. Future research could focus on investigating the relationships between the characteristics presented herein with specific educational objectives and students' engagement and motivation, providing more insights in this direction.

6 Limitations

The limitations of the study are reported in this section. The limitations concern the non-inclusion of all relevant research based on specific criteria applied by the authors. In the present study, only research written in English and published after 2006 was selected. In addition, searches were contacted for programming environments targeting K-9 education based on keywords derived from the analysis of other reviews. This has led to the inclusion of certain programming environments listed in these reviews. Therefore, the study does not include all of the visual programming environments that may be targeted at K-9 education because they have not been studied in scientific papers. Lastly, the inclusion of only visual programming environments where students are active users could be an additional limitation.

References

Aggarwal, A., Touretzky, D. S., and Gardner-McCune, C. (2018) 'Demonstrating the ability of elementary school students to reason about programs', *SIGCSE '18: Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, Association for Computing Machinery, New York, NY, USA, pp.735–740.

Basawapatna, A., Repenning, A., Koh, K.H. and Nickerson, H. (2013a) 'The zones of proximal flow: guiding students through a space of computational thinking skills and challenges', *ICER 2013 – Proceedings of the 2013 ACM Conference on International Computing Education Research*, Association for Computing Machinery, New York, NY, USA, pp.67–74.

Basawapatna, A., Repenning, A. and Lewis, C. (2013b) 'The simulation creation toolkit: an initial exploration into making programming accessible while preserving computational thinking', *SIGCSE 2013 – Proceedings of the 44th ACM Technical Symposium on Computer Science Education*, Association for Computing Machinery, New York, NY, USA, pp.501–506.

Basawapatna, A.R., Repenning, A. and Koh, K.H. (2015) 'Closing the cyberlearning loop: enabling teachers to formatively assess student programming projects', *SIGCSE '15 Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, Association for Computing Machinery, New York, NY, USA, pp.12–17.

Buitrago Flórez, F., Casallas, R., Hernández, M., Reyes, A., Restrepo, S. and Danies, G. (2017) 'Changing a generation's way of thinking: teaching computational thinking through programming', *Review of Educational Research*, Vol. 87 No. 4, pp.834–860.

Ching, Y-H., Hsu, Y-C. and Baldwin, S. (2018) 'Developing computational thinking with educational technologies for young learners', *TechTrends*, Vol. 62, No. 1, pp.563–573.

Chuter, A. (2016) *Can You Code on a Mobile Device? – Critically Examining mLearning Tools for K-12 Programmers and Coders* [online] https://ict4kids.ca/2016/08/24/can-you-code-on-a-mobile-device-critically-examining-mlearning-tools-for-k-12-programmers-and-coders/#respond (accessed 17 January 2019).

Cooper, S. (2010) 'The design of Alice', *ACM Transactions on Computing Education*, Vol. 10, No. 4, pp.1–16.

Cooper, S., Dann, W. and Pausch, R. (2003) 'Teaching objects-first in introductory computer science', *SIGCSE '03 Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, Association for Computing Machinery, New York, NY, USA, pp.191–195.

Costa, J. and Miranda, G. (2016) 'Relation between Alice software and programming learning: a systematic review of the literature and meta-analysis', *British Journal of Educational Technology,* Vol. 48, No. 6, pp.1464–1474.

Dunjohn, C. (2013) *Tynker Introduces your Kids to Programming Code Either at Home or at School* [online] https://newatlas.com/tynker-programming-for-kids/28598/ (accessed 17 January 2019).

Empson, R. (2014) *With 5M Users Already On Board, Tynker Goes Mobile To Help Kids Learn To Code On The iPad* [online] https://techcrunch.com/2014/03/12/with-5m-users-already-on-board-tynker-goes-mobile-to-help-kids-learn-to-code-on-the-ipad/ (accessed 17 January 2019).

Fokides, E. (2017a) 'Digital educational games and mathematics. Results of a case study in primary school settings', *Education and Information Technologies*, Vol. 23, No. 2, pp.851–867.

Fokides, E. (2017b) 'Tablets, very young primary school students, and basic programming concepts', *Asian Journal of Education and e-Learning*, Vol. 5, No. 3, pp.86–94.

Fronza, I., El Ioini, N. and Corral, L. (2017) 'Teaching computational thinking using agile software engineering methods: a framework for middle schools', *ACM Transactions on Computing Education*, Vol. 17, No. 4, pp.1–28.

Gedik, N., Cetin, M. and Koca, C. (2017) 'Examining the experiences of preschoolers on programming via tablet computers', *Mediterranean Journal of Humanities*, Vol. 7, No. 1, pp.193–203.

Grover, S. and Pea, R. (2013a) 'Computational thinking in K-12: a review of the state of the field', *Educational Researcher*, Vol. 42, No. 1, pp.38–43.

Grover, S. and Pea, R. (2013b) 'Using a discourse-intensive pedagogy and android's App Inventor for introducing computational concepts to middle school students', *SIGCSE 2013 – Proceedings of the 44th ACM Technical Symposium on Computer Science Education*, Association for Computing Machinery, New York, NY, USA, pp.723–728.

Hershkovitz, A., Sitman, R., Israel-Fishelson, R., Eguíluz, A., Garaizar, P., and Guenaga, M. (2019) 'Creativity in the acquisition of computational thinking', *Interactive Learning Environments*, Vol. 27, Nos. 5–6, pp.628–644.

Hsu, T-C., Chang, S-C. and Hung, Y-T. (2018) 'How to learn and how to teach computational thinking: suggestions based on a review of the literature', *Computers and Education*, Vol. 126, pp.296–310.

Ioannidou, A., Repenning, A. and Webb, D.C. (2009) 'AgentCubes: incremental 3D end-user development', *Journal of Visual Languages and Computing*, Vol. 20, No. 4, pp.236–251.

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

Kalelioğlu, F. and Gülbahar, Y. (2014) 'The effects of teaching programming via Scratch on problem solving skills: a discussion from learners' perspective', *Informatics in Education*, Vol. 13, No. 1, pp.33–50.

Keane, T., Chalmers, C., Boden, M. and Williams, M. (2019) 'Humanoid robots: learning a programming language to learn a traditional language', *Technology, Pedagogy and Education*, Vol. 28, No. 5, pp.533–546.

Lazarinis, F., Karachristos, C.V., Stavropoulos, E.C. and Verykios, V.S. (2018) 'A blended learning course for playfully teaching programming concepts to school teachers', *Education and Information Technologies*, Vol. 24, No. 1, pp.1237–1249.

Lewis, C.M. (2010) 'How programming environment shapes perception, learning and goals: Logo vs. Scratch', *SIGCSE'10 – Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, Association for Computing Machinery, New York, NY, USA, pp.346–350.

Lin, Y. and Weintrop, D. (2021) 'The landscape of Block-based programming: characteristics of block-based environments and how they support the transition to text-based programming', *Journal of Computer Languages*, Vol. 67, No. 4, pp.1–18.

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*, Vol. 41, pp.51–61.

Maloney, J., Peppler, K., Kafai, Y., Resnick, M. and Rusk, N. (2008) 'Programming by choice: urban youth learning programming with Scratch', *SIGCSE '08 Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, Association for Computing Machinery, New York, NY, USA, pp.367–371.

Monteiro, I.T., Salgado, L.C.D., Mota, M.P., Sampaio, A.L. and de Souza, C.S. (2017) 'Signifying software engineering to computational thinking learners with AgentSheets and PoliFacets', *Journal of Visual Languages and Computing*, Vol. 40, pp.91–112.

Morelli, R., de Lanerolle, T., Lake, P., Limardo, N., Tamotsu, E. and Uche, C. (2010) 'Can Android App Inventor bring computational thinking to K-12?', *The Humanitarian Foss Project* [online] http://www.hfoss.org/index.php/publications-etc (accessed 14 April 2019).

Moreno-León, J., Robles, G. and Román-González, M. (2016) 'Code to learn: where does it belong in the K-12 curriculum?', *Journal of Information Technology Education: Research*, Vol. 15, No. 1, pp.283–303.

Padlipsky, S. (2018) *Using Offline Activities to Enhance Online Cybersecurity Education*, Unpublished Master Thesis, Faculty of California Polyethnic State University, San Luis Obispo, USA.

Papadakis, S. (2021) 'The impact of coding apps to support young children in computational thinking and computational fluency. A literature review', *Frontiers in Education*, Vol. 6 [online] https://www.frontiersin.org/articles/10.3389/feduc.2021.657895/full (accessed 07 April 2022).

Papadakis, S. (2022) 'Can preschoolers learn computational thinking and coding skills with ScratchJr? A systematic literature review', *International Journal of Educational Reform*, pp.1–34 [online] https://journals.sagepub.com/doi/pdf/10.1177/10567879221076077 (accessed 14 April 2022).

Papadakis, S., Kalogiannakis, M., Orfanakis, V. and Zaranis, N. (2014) 'Novice programming environments. Scratch and App Inventor: a first comparison', *IDEE '14 Proceedings of the 2014 Workshop on Interaction Design in Educational Environments*, Association for Computing Machinery, New York, NY, USA, pp.1–7.

Perdikuri, K. (2014) 'Students' experiences from the use of MIT App Inventor in classroom', *PCI '14 Proceedings of the 18th Panhellenic Conference on Informatics*, Association for Computing Machinery, New York, NY, USA, pp.1–6.

Remshagen, A., Gray, K. and Lee, T. (2018) 'A Scratch hackathon for teens', *Proceedings of the 2018 International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS'18)*, CSREA Press, pp.136–140.

Repenning, A. (2012) 'Education: programming goes back to school', *Communications of the ACM*, Vol. 55, No. 5, pp.38–40.

Repenning, A. (2017) 'Moving beyond syntax: lessons from 20 years of blocks programing in AgentSheets', *CU Experts*, Vol. 3, No. 1, pp.68–91.

Repenning, A. and Sumner, T. (1995) 'Agentsheets: a medium for creating domain-oriented visual languages', *Computer*, Vol. 28, No. 3, pp.17–25.

Resnick, M. and Silverman, B. (2005) 'Some reflections on designing construction kits for kids', *Proceedings of the 2005 Conference on Interaction Design and Children*, Association for Computing Machinery, New York, NY, USA, pp.117–122.

Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B. and Kafai, Y. (2009) 'Scatch: programming for all', *Communications of the ACM*, Vol. 52, No. 11, pp.60–67.

Rodger, S.H., Bashford, M., Dyck, L., Hayes, J., Liang, L., Nelson, D. and Qin, H. (2010) 'Enhancing K-12 education with Alice programming adventures', *ITiCSE'10 – Proceedings of the 2010 ACM SIGCSE Annual Conference on Innovation and Technology in Computer Science Education*, Association for Computing Machinery, New York, NY, USA, pp.234–238.

Sáez-López, J.M., Román-González, M. and Vázquez-Cano, E. (2016) 'Visual programming languages integrated across the curriculum in elementary school: a two year case study using "Scratch" in five schools', *Computers and Education*, Vol. 97, No. 3, pp.129–141.

Sengupta, P., Kinnebrew, J.S., Basu, S., Biswas, G. and Clark, D. (2013) 'Integrating computational thinking with K-12 science education using agent-based computation: a theoretical framework', *Education and Information Technologies*, Vol. 18, No. 2, pp.351–380.

Sykes, E.R. (2007) 'Determining the effectiveness of the 3D Alice programming environment at the computer science I level', *Journal of Educational Computing Research*, Vol. 36, No. 2, pp.223–244.

Tang, K-Y., Chou, T-L. and Tsai, C-C. (2019) 'A content analysis of computational thinking research: an international publication trends and research typology', *Asia-Pacific Education Researcher*, Vol. 29, No. 4, pp.9–19.

Tikva, C. and Tambouris, E. (2021) 'Mapping computational thinking through programming in K-12 education: a conceptual model based on a systematic literature review', *Computers and Education*, Vol. 162, article 104083.

Trory, A., Howland, K. and Good, J. (2018) 'Designing for concreteness fading in primary computing', *IDC 2018 – Proceedings of the 2018 ACM Conference on Interaction Design and Children*, Association for Computing Machinery, New York, NY, USA, pp.278–288.

Webster, J. and Watson, R. (2002) 'Analyzing the past to prepare for the future: writing a literature review', *MIS Quarterly*, Vol. 26, No. 2, pp.13–23.

Weintrop, D. and Wilensky, U. (2015) 'To block or not to block, that is the question: students' perceptions of blocks-based programming', *IDC '15: Proceedings of the 14th International Conference on Interaction Design and Children*, 21–24 June, pp.199–208, doi: 10.1145/2771839.2771860.

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L. and Wilensky, U. (2015) 'Defining computational thinking for mathematics and science classrooms', *Journal of Science Education and Technology*, Vol. 25, No. 1, pp.127–147.

Werner, L., Campe, S. and Denner, J. (2012) 'Children learning computer science concepts via alice game-programming', *SIGCSE '12 Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, Association for Computing Machinery, New York, NY, USA, pp.427–432.

Wing, J.M. (2006) 'Computational thinking', *Communications of the ACM*, Vol. 49, No. 3, pp.33–35.

Zhang, L. and Nouri, J. (2019) 'A systematic review of learning computational thinking through Scratch in K-9', *Computers and Education*, Vol. 141, article 103607.

Zhong, B., Wang, Q., Chen, J. and Li, Y. (2016) 'An exploration of three-dimensional integrated assessment for computational thinking', *Journal of Educational Computing Research*, Vol. 53, No. 4, pp.562–590.

Appendix

Table A1 Selected cases

id	*Source*	*Environment*
C1	Aggarwal, A., Touretzky, D. S., and Gardner-McCune, C. (2018) 'Demonstrating the ability of elementary school students to reason about programs', *SIGCSE '18: Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, Association for Computing Machinery, New York, NY, USA, pp.735–740.	Kodu
C2	Aivaloglou, H. and Hermans, F. (2019) 'Early programming education and career orientation: the effects of gender, self-efficacy, motivation and stereotypes', *SIGCSE '19: Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, Association for Computing Machinery, New York, NY, USA, pp.679–685.	Scratch
C3	Akcaoglu, M., and Santos Green, L. (2018) 'Teaching systems thinking through game design', *Educational Technology Research and Development*, Vol. 67, No. 1, pp.1–19.	Kodu
C4	Allsop, Y. (2018) 'Assessing computational thinking process using a multiple evaluation approach', *International Journal of Child-Computer Interaction*, Vol. 19, pp.30–55.	Alice, Scratch
C5	Barmpoutis, A. and Huynh, K. (2019) 'Name tags and pipes: assessing the role of metaphors in students' early exposure to computer programming using emoticoding', *Advances in Human Factors in Training, Education, and Learning Sciences*, Vol. 785, pp.194–202.	Tynker
C6	Basawapatna, A., Koh, K.H. and Repenning, A. (2010) 'Using scalable game design to teach computer science from middle school to graduate school', *ITiCSE'10 Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education*, Association for Computing Machinery, New York, NY, USA, pp.224–228.	AgentSheets
C7	Basawapatna, A., Repenning, A., Koh, K.H. and Nickerson, H. (2013) 'The zones of proximal flow: guiding students through a space of computational thinking skills and challenges', *ICER 2013 – Proceedings of the 2013 ACM Conference on International Computing Education Research*, Association for Computing Machinery, New York, NY, USA, pp.67–74.	AgentSheets

Table A1 Selected cases (continued)

id	*Source*	*Environment*
C8	Basawapatna, A., Repenning, A. and Lewis, C. (2013) 'The simulation creation toolkit: an initial exploration into making programming accessible while preserving computational thinking', *SIGCSE 2013 – Proceedings of the 44th ACM Technical Symposium on Computer Science Education*, Association for Computing Machinery, New York, NY, USA, pp.501–506.	AgentCubes, AgentSheets
C9	Basawapatna, A.R., Repenning, A. and Koh, K.H. (2015) 'Closing the cyberlearning loop: enabling teachers to formatively assess student programming projects', *SIGCSE '15 Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, Association for Computing Machinery, New York, NY, USA, pp.12–17.	AgentCubes
C10	Basawapatna, A.R., Repenning, A. and Savignano, M. (2019) 'The zones of proximal flow tutorial: Designing computational thinking cliffhangers', *SIGCSE '19: Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, Association for Computing Machinery, New York, NY, USA, pp.428–434.	AgentCubes
C11	Basu, S. (2019) 'Using rubrics integrating design and coding to assess middle school students' open-ended block-based programming projects', *SIGCSE '19: Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, Association for Computing Machinery, New York, NY, USA, pp.1211–1217.	MIT App Inventor, Scratch
C12	Bell, S., Frey, T. and Vasserman, E. (2014) 'Spreading the word: introducing pre-service teachers to programming in the K-12 classroom', *SIGCSE '14 Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, Association for Computing Machinery, New York, NY, USA, pp.187–192.	Scratch
C13	Brennan, K. and Resnick, M. (2012) 'New frameworks for studying and assessing the development of computational thinking', *Proceedings of the 2012 Annual Meeting of the American Educational Research Association*, AERA, Washington, DC, USA, pp.1–25.	Scratch
C14	Buelin-Biesecker, J.K. (2012) *Fostering and Assessing Creativity in Technology Education*, Unpublished PhD thesis, North Carolina State University, Raleigh, NC, USA.	Game Maker
C15	Burke, Q. and Kafai, Y.B. (2010) 'Programming and storytelling: opportunities for learning about coding and composition', *IDC'10: Proceedings of the 9th International Conference on International Design and Children*, Association for Computing Machinery, New York, NY, USA, pp.348–351.	Scratch
C16	Burke, Q. and Kafai, Y. B. (2012) 'The writer's workshop for youth programmers: digital storytelling with Scratch in middle school classrooms', *SIGCSE'12 Proccedings of the 43rd ACM Technical Symposium on Computer Science Education*, Association for Computing Machinery, New York, NY, USA, pp.433–438.	Scratch
C17	Calao, L.A., Moreno-Leon, J., Correa, H.E. and Robles, G. (2015) 'Developing mathematical thinking with Scratch: an experiment with 6th grade students', *EC-TEL 2015: Design for Teaching and Learning in a Networked World*, Springer-Verlag, Berlin, Heidelberg, pp.17–27.	Scratch

Table A1 Selected cases (continued)

id	*Source*	*Environment*
C18	Cetin, I. (2016) 'Preservice teachers' introduction to computing: exploring utilization of Scratch', *Journal of Education Computing Research*, Vol. 54, No. 7, pp.997–1021.	Scratch
C19	Chatzinikolakis, G. and Papadakis, S. (2014) 'Motivating K-12 students learning fundamental computer science concepts with App Inventor', *2014 International Conference on Interactive Mobile Communication Technologies and Learning (IMCL2014)*, IEEE Publications, pp.152–159.	MIT App Inventor
C20	Chiazzese, G., Fulantelli, G., Pipitone, V. and Taibi, D. (2018) 'Engaging primary school children in computational thinking: Designing and developing videogames', *Education in th Knowledge Society*, Vol. 19, No. 2, pp.63–81.	Kodu
C21	Cooper, S., Rodger, S.H., Schep, M., Stalvey, R.H. and Dann W. (2015) 'Growing a K-12 community of practice', *SIGCSE '15 Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, Association for Computing Machinery, New York, NY, USA, pp.290–295.	Alice
C22	Dag, F. (2019) 'Prepare pre-service teachers to teach computer programming skills at K-12 level: experiences in a course', *Journal of Computers in Education*, Vol. 6, No. 2, pp.277–313.	Alice, Scratch
C23	Denner, J., Werner, L., Campe, S. and Ortiz, E. (2014) 'Pair programming: under what conditions is it advantageous for middle school students', *Journal of Research on Technology in Education*, Vol. 46, No. 3, pp.277–296.	Alice
C24	Dong, Y., Catete, V., Lytle, N., Isvik, A., Barnes, T., Jocius, R., Albert, J., Joshi, D., Robinson, R. and Andrews, A. (2019) 'Infusing computing: analyzing teacher programming products in K-12 computational thinking professional development', *ITiCSE '19: Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, Association for Computing Machinery, New York, NY, USA, pp.278–284.	Snap!
C25	Ferreira, M.N.F., Gresse von Wangenheim, C., Missfeldt Filho, R., Pinheiro, F. and Hauck, J.C.R. (2019) 'Learning user interface design and the development of mobile applications in middle school', *Interactions*, Vol. 26 No. 4, pp.66–69.	MIT App Inventor
C26	Fokides, E. (2017) 'Digital educational games and mathematics. Results of a case study in primary school settings', *Education and Information Technologies*, Vol. 23, No. 2, pp.851–867.	Kodu
C27	Gedawy, H., Razak, S., and Alshikhabobakr, H. (2019) 'The effectiveness of creating localized content for middle school computing curriculum', *ITiCSE '19: Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, Association for Computing Machinery, New York, NY, USA, pp.478–484.	Alice
C28	Gestwicki, P. and Ahmad, K. (2011) 'App Inventor for Android with studio-based learning', *Journal of Computing Sciences in Colleges*, Vol. 27, No. 1, pp.55–63.	MIT App Inventor

Table A1 Selected cases (continued)

id	*Source*	*Environment*
C29	Grover, S. (2015) '"Systems of assessments" for deeper learning of computational thinking in K-12', *Annual Meeting of the American Educational Research Association*, AERA, Washington, DC, USA.	Scratch
C30	Grover, S. and Basu, S. (2017) 'Measuring student learning in introductory block-based programming: examining misconceptions loops, variables, and boolean logic', *SIGCSE '17: Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, Association for Computing Machinery, New York, NY, USA, pp.267–272.	Scratch
C31	Grover, S. and Pea, R. (2013) 'Using a discourse-intensive pedagogy and android's App Inventor for introducing computational concepts to middle school students', *SIGCSE 2013 – Proceedings of the 44th ACM Technical Symposium on Computer Science Education*, Association for Computing Machinery, New York, NY, USA, pp.723–728.	MIT App Inventor
C32	Hart, M., Early, J.P. and Brylow, D. (2008) 'A novel approach to K-12 CS education: linking mathematics and computer science', *ACM SIGCSE Bulletin*, Vol. 40, No. 1, pp.286–290.	Alice
C33	Hermans, F. and Aivaloglou, E. (2017) 'Teaching software engineering principles to K-12 students: a MOOC on Scratch', *IEEE/AMC 39th International Conference on Software Engineering: Software Engineering Education and Training Track*, IEEE Publications, pp.13–22.	Scratch
C34	Hill, C. (2015) *Programming Environments for Children: Creating a Language that Grows with You*, Unpublished Master thesis, University of California, Santa Barbara, CA, USA.	Scratch
C35	Hu, Y., Li, Y.H. and Su, C.Y. (2019) 'Perceptions of teachers toward game-based programming tools in K-12 classrooms', *International Journal on Computer Science and Information Systems*, Vol. 14, No. 1, pp.17–30.	Code.org
C36	Ioannidou, A., Repenning, A. and Webb, D.C. (2009) 'AgentCubes: incremental 3D end-user development', *Journal of Visual Languages and Computing*, Vol. 20, No. 4, pp.236–251.	AgentCubes
C37	Iskrenovic-Momcilovic, O. (2019) 'Pair programming with Scratch', *Education and Information Technologies*, Vol. 24, No. 6, pp.2943–2952.	Scratch
C38	Jenson, J. and Droumeva, M. (2016) 'Exploring media literacy and computational thinking: a game maker curriculum study', *Electronic Journal of e-Learning*, Vol. 14, No. 2, pp.111–121.	Game Maker
C39	Jenson, J., Black, K. and de Castell, S. (2018) 'Digital game-design: effects of single sex groups on student success', in Ciussi, M. (Ed.): *ECGBL 2018- Proceedings of the 12th European Conference on Game-Based Learning*, pp.258–265.	Game Maker
C40	Jimenez, Y. and Gardner-McCune, C. (2015) 'Using App Inventor and history as a gateway to engage African American students in computer science', *2015 Research in Equity and Sustained Participation in Engineering, Computing, and Technology (RESPECT)*, IEEE Publications, pp.1–2.	MIT App Inventor

Table A1 Selected cases (continued)

id	*Source*	*Environment*
C41	Johnson, C. (2017) 'Learning basic programming concepts with game maker', *International Journal of Computer Science Education in Schools*, Vol. 1, No. 2, pp.1–20.	Game Maker
C42	Kalelioglu, F. and Gulbahar, Y. (2014) 'The effects of teaching programming via Scratch on problem solving skills: a discussion from learners' perspective', *Informatics in Education*, Vol. 13, No. 1, pp.33–50.	Scratch
C43	Kalelioglu, F. (2015) 'A new way of teaching programming skills to K-12 students: Code.org', *Computers in Human Behavior*, Vol. 52, No. 3, pp.200–210.	Code.org
C44	Ketenci, T., Calandra, B., Margulieux, L. and Cohen, J. (2019) 'The relationship between learner characteristics and student outcomes in a middle school computing course: An exploratory analysis using structural equation modeling', *Journal of Research on Technology in Education*, Vol. 51, No. 1, pp.63–76.	MIT App Inventor
C45	Kozuh, I., Krajnc, R., Hadjileontiadis, L. and Debevc, M. (2018) 'Assessment of problem solving ability in novice programmers', *PLoS ONE*, Vol. 13, No. 9, pp.1–21.	MIT App Inventor, Scratch
C46	Kraleva, R., Kralev, V. and Kostadinova, D. (2019) 'A methodology for the analysis of block-based programming languages appropriate for children', *Journal of Computing Science and Engineering*, Vol. 13 No. 1, pp.1–10.	Code.org, Scratch
C47	Kwon, K. and Cheon, J. (2019) 'Exploring problem decomposition and program development through block-based programs', *International Journal of Computer Science Education in Schools*, Vol. 3 ,No. 1, pp.1–14.	Scratch
C48	Lazarinis, F., Karachristos, C.V., Stavropoulos, E.C. and Verykios, V.S. (2018) 'A blended learning course for playfully teaching programming concepts to school teachers', *Education and Information Technologies*, Vol. 24, No. 1, pp.1237–1249.	Scratch
C49	Lewis, C.M. (2010) 'How programming environment shapes perception, learning and goals: Logo vs. scratch', *SIGCSE'10 – Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, Association for Computing Machinery, New York, NY, USA, pp.346–350.	Scratch
C50	Liu, J., Lin, C.H. Potter, P., Hasson, E.P., Barnett, Z.D. and Singleton, M. (2013) 'Going mobile with App Inventor for Android: a one-week computing workshop for K-12 teachers', *SIGCSE '13 Proceedings of the 44th ACM Technical Symposium on Computer Science Education*, Association for Computing Machinery, New York, NY, USA, pp.433–438.	MIT App Inventor
C51	Liu, J., Wilson, J., Hemmenway, D., Xu, Y. and Lin, C.H. (2015) 'Oh Snap! A one-week summer computing workshop for K-12 teachers', *2015 10th International Conference on Computer Science and Education (ICCSE)*, IEEE Publications, pp.663–668.	Snap!

Table A1 Selected cases (continued)

id	*Source*	*Environment*
C52	Makris, D., Euaggelopoulos, K., Chorianopoulos, K. and Giannakos, M.N. (2013) 'Could you help me to change the variables?: Comparing instruction to encouragement for teaching programming', *WiPSE '13: Proceedings of the 8th Workshop in Primary and Secondary Computing Education*, Association for Computing Machinery, New York, NY, USA, pp.79–82.	Scratch
C53	Maloney, J., Peppler, K., Kafai, Y., Resnick, M. and Rusk, N. (2008) 'Programming by choice: urban youth learning programming with scratch', *SIGCSE '08 Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, Association for Computing Machinery, New York, NY, USA, pp.367–371.	Scratch
C54	Marcelino, M., Pessoa, T., Vieira, C., Salvador, T. and Mendes, A. (2017) 'Learning computational thinking and Scratch at distance', *Computers in Human Behavior*, Vol. 80, No. 3, pp.470–477.	Scratch
C55	Monteiro, I.T., Salgado, L.C.D., Mota, M.P., Sampaio, A.L. and de Souza, C.S. (2017) 'Signifying software engineering to computational thinking learners with AgentSheets and PoliFacets', *Journal of Visual Languages and Computing*, Vol. 40, pp.91–112.	AgentSheets
C56	Morelli, R., de Lanerolle, T., Lake, P., Limardo, N., Tamotsu, E. and Uche, C. (2010) 'Can Android App Inventor bring computational thinking to K-12?', *The Humanitarian Foss Project* [online] http://www.hfoss.org/index.php/publications-etc (accessed 14 April 2019).	MIT App Inventor
C57	Ni, L., Schilder, D., Sherman, M. and Martin, F. (2016) 'Computing with a community focus: outcomes from an App Inventor summer camp for middle school students', *Journal of Computing Sciences in Colleges*, Vol. 31, No. 6, pp.82–89.	MIT App Inventor
C58	Nikou, S.A. and Economides, A.A. (2014) 'Transition in student motivation during a scratch and an App Inventor course', *2014 IEEE Global Engineering Education Conference (EDUCON)*, IEEE Publications, pp.1042–1045.	MIT App Inventor, Scratch
C59	Nygard, S., Kolas, L. and Sigurdardottir, H. (2015) 'Teachers' experiences using Kodu as a teaching tool', in Munkvold, R. and Kolas, L. (Eds.): *ECGBL 2018 – Proceedings of the 9th European Conference on Games Based Learning*, Academic Conferences and Publishing International Limited, Reading, pp.416–422.	Kodu
C60	Papadakis, S., Kalogiannakis, M., Zaranis, N., and Orfanakis, V. (2016) 'Using Scratch and App Inventor for teaching introductory programming in secondary education. A case study', *International Journal of Technology Enhanced Learning*, Vol. 8, Nos. 3/4, pp.217–233.	MIT App Inventor, Scratch
C61	Park, Y. and Shin, Y. (2019) 'Comparing the effectiveness of Scratch and App Inventor with regard to learning computational thinking concepts', *Electronics*, Vol. 8, No. 11, pp.1269–1281.	MIT App Inventor, Scratch
C62	Patton, R.M. (2013) 'Games as an artistic medium: investigating complexity thinking in game-based art pedagogy', *Studies in Art Education*, Vol. 55 No. 1, pp.35–50.	Game Maker

Table A1 Selected cases (continued)

id	*Source*	*Environment*
C63	Perdikuri, K. (2014) 'Students' experiences from the use of MIT App Inventor in classroom', *PCI '14 Proceedings of the 18th Panhellenic Conference on Informatics*, Association for Computing Machinery, New York, NY, USA, pp.1–6.	MIT App Inventor
C64	Perez-Marin, D., Hijon-Neira, R., Bacelo, A. and Pizarro, C. (2018) 'Can computational thinking be improved by using a methodology based on metaphors and scratch to teach computer programming to children?', *Computers in Human Behavior*, Vol. 105.	Scratch
C65	Piech, C., Sahami, M., Huang, J. and Guibas, L. (2015) 'Autonomously generating hints by infering problem solving policies', *L@S '15 Proceedings of the Second (2015) ACM Conference on Learning @ Scale*, Association for Computing Machinery, New York, NY, USA, pp.195–204.	Code.org
C66	Remshagen, A., Gray, K. and Lee, T. (2018) 'A Scratch hackathon for teens', *Proceedings of the 2018 International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS'18)*, CSREA Press, pp.136–140.	Scratch
C67	Repenning, A., Lamprou, A., Petralito, S., and Basawapatna, A. (2019) 'Making computer science education mandatory: Exploring a demographic shift in Switzerland', *ITiCSE '19: Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, Association for Computing Machinery, New York, NY, USA, pp.422–428.	AgentCubes
C68	Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B. and Kafai, Y. (2009) 'Scatch: programming for all', *Communications of the ACM*, Vol. 52, No. 11, pp.60–67.	Scratch
C69	Rodger, S., Hayes, J., Lezin, G., Qin, H., Deborah, N. and Tucker, R. (2009) 'Engaging middle school teachers and students with Alice in a diverse set of subjects', *SIGCSE '09 Proceedings of the 40th ACM technical symposium on Computer Science Education*, Association for Computing Machinery, New York, NY, USA, pp.271–275.	Alice
C70	Roy, K. (2012) 'App Inventor for Android: report from a summer camp', *SIGCSE '12 Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, Association for Computing Machinery, New York, NY, USA, pp.283–288.	MIT App Inventor
C71	Ruan, L., Patton, E. and Tissenbaum, M. (2017) 'Evaluations of programming complexity in App Inventor', in Kong, S.C. et al. (Eds.): *CTE 2017 International Conference on Computational Thinking Education*, pp.2–5.	MIT App Inventor
C72	Saez-Lopez, J.M., Sevillano-Garcia, M.L. and Vazquez-Cano, E. (2019) 'The effect of programming on primary school students' mathematical and scientific understanding: educational use of mBot', *Educational Technology Research and Development*, Vol. 67, No. 3, pp.1405–1425.	Scratch

Table A1 Selected cases (continued)

id	*Source*	*Environment*
C73	Smith, N., Sutcliffe, C. and Sandvik, L. (2014) 'Code club: bringing programming to UK primary schools through Scratch', *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, Association for Computing Machinery, New York, NY, USA, pp.517–522.	Scratch
C74	Sung, W., Choi, A. and Black, J. (2017) 'Incorporating touch-based tablets into classroom', in Mentor, D. (Ed.): *Handbook of Research in Mobile Learning in Contemporary Classrooms*, IGI Global, Pennsylvania, pp.378–406.	Hopscotch
C75	Sykes, E.R. (2007) 'Determining the effectiveness of the 3D Alice programming environment at the computer science I level', *Journal of Educational Computing Research*, Vol. 36, No. 2, pp.223–244.	Alice
C76	Touretzky, D.S. (2014) 'Teaching Kodu with physical manipulatives', *ACM Inroads*, Vol. 5 No. 4, pp.44–51.	Kodu
C77	Touretzky, D.S., Gardner-McCune, C. and Aggarwal, A. (2016) 'Teaching "lawfulness" with Kodu', *SIGCSE'16: Proceedings of the 47th ACM Technical Symposium on Computing Science Education,* Association for Computing Machinery, New York, NY, USA, pp.621–626.	Kodu
C78	Touretzky, D.S., Gardner-McCune, C. and Aggarwal, A. (2017) 'Semantic reasoning in young programmers', *SIGCSE'17: Proceedings of the ACM SIGCSE Technical Symposium on Computer Science Education*, Association for Computing Machinery, New York, NY, USA, pp.585–590.	Kodu
C79	Werner, L., Denner, J., Bliesner, M. and Rex, P. (2009) 'Can middle-schoolers use storytelling Alice to make games? Results of a pilot study', *FDG '09 Proceedings of the 4th International Conference of Foundations of Digital Games*, Association for Computing Machinery, New York, NY, USA, pp.207–214.	Alice
C80	Werner, L., Campe, S. and Denner, J. (2012) 'Children learning computer science concepts via Alice game-programming', *SIGCSE '12 Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, Association for Computing Machinery, New York, NY, USA, pp.427–432.	Alice
C81	Wong, G.K.W. and Cheung, H.Y. (2018) 'Exploring children's perceptions of developing twenty-first century skills through computational thinking and programming', *Interactive Learning Environments*, Vol. 28, No. 4, pp.438–450.	Kodu, Scratch
C82	Yildiz Durak, H. and Guyer, T. (2019) 'Programming Scratch in primary school, indicators related to effectiveness of education process and analysis of these indicators in terms of various variables', *Gifted Education International*, Vol. 35, No. 3, pp.237–258.	Scratch
C83	Zha, S., Jin, Y., Moore, P. and Gaston, J. (2019) 'Hopscotch into coding: Introducing pre-service teachers computational thinking', *TechTrends*, Vol. 64, No. 1, pp.17–28.	Hopscotch

Table A2 Selected programming environments

id	*Official website*	*Programming environment*
W1	https://agentsheets.com	AgentCubes, AgentSheets
W2	https://www.alice.org	Alice
W3	https://code.org	Code.org
W4	https://www.yoyogames.com/gamemaker	Game Maker
W5	https://www.gethopscotch.com	Hopscotch
W6	https://www.kodugamelab.com	Kodu
W7	https://appinventor.mit.edu/	MIT App Inventor
W8	https://scratch.mit.edu/	Scratch
W9	https://snap.berkeley.edu/	Snap!
W10	https://www.tynker.com	Tynker