Effect of SRA-programming on computational thinking through different output modalities

Fanchamps, Nardie; Slangen, Lou; Specht, Marcus; Hennissen, Paul

DOI
10.1007/s40692-022-00236-w

Publication date
2022

Document Version
Final published version

Published in
Journal of Computers in Education

J. Comput. Educ.
https://doi.org/10.1007/s40692-022-00236-w

Effect of SRA-programming on computational thinking through different output modalities

Nardie Fanchamps[1] · Lou Slangen[2] · Marcus Specht[4] · Paul Hennissen[2,3]

Received: 21 July 2021 / Revised: 26 January 2022 / Accepted: 13 June 2022

Abstract The application of sense-reason-act (SRA) programming in contemporary education can ensure the development of computational thinking (CT) at a more advanced level. SRA-programming has been identified as an instrumental way of thinking for learning to program robots and encourages the development of the more complex concepts of programming. Visual programming environments are diverse in appearance and prove to be an excellent way to teach pupils the basic ideas of programming. It is important to investigate whether the type of output has a characteristic influence on the level of development of CT in visual programming environments. In this research, we therefore explore whether characteristic differences in the development of CT can be measured when SRA-programming is applied in a visual programming environment with an on-screen output or a tangible output. It was expected that the observed effect of pupils' programming actions through the application of SRA would show that the type of output influences the understanding of complex programming concepts at a higher level. Our results indicate that SRA-programming with visual, on-screen output yields a significant increase in the development of CT, as opposed to SRA-programming with a tangible output. The development of complex programming concepts can also be demonstrated.

Keywords SRA-programming · Computational thinking · Visual programming · Visual output · Tangible output

✉ Nardie Fanchamps
nardie.fanchamps@ou.nl

[1] Open University, Valkenburgerweg 177, 6419 Heerlen, Netherlands

[2] Fontys University of Applied Science, Mgr. Claessenstraat 4, 6131 Sittard, Netherlands

[3] Zuyd University of Applied Science, Nieuw Eyckholt 300, 6419 Heerlen, Netherlands

[4] Delft University of Technology, Mekelweg 4, 2628 Delft, Netherlands

Published online: 09 July 2022

Introduction

The use of information communication technology (ICT) is increasingly widespread, and modern society demands a thorough preparation in this field at the earliest possible age (Hernandez, 2017; Kafai & Burke, 2013; Stamati, 2020; Yelland, 2005). Programming is now an integral part of everyday life (Iivari et al., 2020; Luxton-Reilly, 2016), and primary school appears to be the most designated place to teach pupils the required competencies and skills at an early stage (Atman Uslu & Usluel, 2019; Edwards, 2005). There is a variety of didactic approaches by which learners can develop computational thinking (CT), which can contribute to their problem-solving capabilities (Estapa et al., 2018; Nouri et al., 2020). However, CT involves more than the ability to solve challenging problems using skills derived from the world of computer science (Brennan & Resnick, 2012; Israel-Fishelson & Hershkovitz, 2022; Tsarava et al., 2021). It encompasses the mental skills and practices needed to design computations that can let computers perform tasks for us, and to explain and interpret the world as a complex system of information processes (Denning & Tedre, 2019; Lai, 2021). The provision of opportunities to further stimulate the development of CT, and to enable pupils to acquire the associated skills, entails demands on both the environment and the task (Kong & Abelson, 2019; Rich & Browning, 2022; Yadav et al., 2016).

A variety of programming environments are available to teach pupils the concepts of programming at the primary education level (López et al., 2021; Wahl & Thomas, 2002). Visual programming environments are currently very popular and offer a wide range of possibilities (Chao, 2016; Ray, 2017). The basic starting point for visual programming is that definable code blocks are dragged into a worksheet using the 'drag and drop' method, and are arranged in the correct sequence (Weintrop, 2019; Weintrop & Wilensky, 2015). When used in this way, pupils can construct a program and execute it. A distinction can be made between (a) visual programming with a visual on-screen output, in which virtual objects are programmed and controlled, and (b) visual programming with a tangible, physically perceptible output in which concrete artefacts are programmed and controlled (Caci et al., 2013a, 2013b; Corral et al., 2019; Horn et al., 2009). This difference in the type of output determines the way in which pupils receive feedback on their programming actions (Sapounidis et al., 2015; Zhang & Nouri, 2019).

The ability to construct a computer program that anticipates changing environmental conditions by means of sensor observations demands a different computational approach than performing programming tasks in an unchanging, predictable environment (Gomes & Mendes, 2007; Kim & Kim, 2003; Kyriazopoulos et al., 2022). By using sense-reason-act (SRA) programming, a programmed artefact or a simulation of reality can react to changes in its surroundings (Fanchamps et al., 2021). The concept of SRA, derived from the world of robotics, requires the understanding that the operating program should continuously compare the external conditions with the desired conditions (Lith, 2006). Actions then become

conditional rather than automatic. An SRA-programming application is characterised by an initial process in which a state is detected through sensing or sensor observations (*sense*); this situation is then compared with the corresponding values of the computer program (*reason*), and the computer program is then used to execute subsequent actions (*act*) (Slangen, 2016). SRA-programming requires the use of more complex forms of iterations, conditionals and functions rather than restricted linear, sequential programming structures (Martinez et al., 2015). This requires logical reasoning of the ''if … then …'', ''wait … until …'', ''repeat … until …'' kind. The complexity is rooted in the ability to think in terms of scenarios, and to understand and apply the more complex concepts of programming (Popat & Starkey, 2019).

Earlier research shows that although the use of SRA contributes to the development of CT (Fanchamps et al., 2020; Slangen, 2016; Wong, 2014), pupils lack fluency in the more complicated language of conditional reasoning and nesting required for successful SRA-programming. In addition, previous research has also shown that pupils need (dedicated) assistance in order to understand the concept of SRA (Slangen, 2016). It has been demonstrated that the concept of SRA, encompassing a generalisable theory that contains several specific conceptual elements (e.g. parallel thinking, cause-effect relationships and conditional reasoning), is applicable within visually oriented programming environments where variations are compared only with a visual, on-screen output, or only with a physical tangible output (Fanchamps et al., 2019, 2020). It is questionable whether the distinction between a visual, on-screen output and a tangible output can influence a better understanding of the concept of SRA, with consequent effects on the development of CT. This study therefore aims to investigate whether the use of SRA-programming has an impact on the understanding of complex programming concepts, and whether the impact on CT when using visual programming environments depends on the use of an on-screen output or a physical, tangible output. Our research results are also compared to the performance of a control group that did not use either type of visual programming environment.

Theoretical framework

In this research, we are specifically interested in the effect on CT of applying SRA-programming in a visual programming environment, as demonstrated previously (Fanchamps et al., 2021), depending on the type of output (i.e. visual or tangible). We also want to know whether the type of output influences the understanding of the more complex programming concepts when SRA is used (Zapata-Cáceres et al., 2020).

From earlier research (Fanchamps et al., 2019), we know that primary school pupils show a higher level of CT when SRA is applied in robot programming. We also know that the application of SRA with an impact on CT depends on the design of the task and the environmental conditions in which robots are programmed (Slangen, 2016). In addition, we know that the application of SRA in a visual

programming environment with visual, on-screen output enables the development of CT (Fanchamps et al., 2021).

CT is a conceptualised way of thinking with the aim of solving problems by using fundamental concepts of computer science (Hsu et al., 2018; Wing, 2006). It refers to a logical approach towards solving problems through problem formulation, data organisation, analysis and representation (Denning & Tedre, 2019; Voskoglou & Buckley, 2012). CT is a process of thinking in which problems and their solutions can be reformulated to allow them to be presented in a form that can be effectively implemented by an information processing digital agent (Dummer, 2017; Leifheit et al., 2018; Vourletsis & Politis, 2021). Skills such as problem decomposition, algorithmic thinking, pattern recognition, parallelisation and abstraction are addressed (Catlin & Woollard, 2014; Chalmers, 2018; SLO, 2017). A solution-focused ability is strengthened, and this stimulates creative thinking about the use of digital tools to solve a problem (Lee et al., 2011; Tedre & Denning, 2016).

Programming in accordance with the SRA approach in which sensor-based programming is used and where sensory input determines consecutive actions, is characterised by connecting observations of (virtual or material) sensory input (*sense*) to a reasoning component which initiates actions based on these observations (*reason*) and a process of subsequent actions based on the given inferences (*act*) (Krugman, 2004; Slangen et al., 2011; Wong, 2014). When applying SRA-programming, complex programming concepts such as iterations, conditionals and functions are used (Basu et al., 2016; Werner et al., 2012). SRA-programming requires conditional, causal and iterative reasoning, abstract thinking and thinking in terms of parameters and variables (Estapa et al., 2018). The ability to functionally apply SRA in programming environments requires pupils to develop logical reasoning and systematic thinking (Fanchamps et al., 2019). Anticipating the requirements of the task design, and enabling and executing targeted interventions, demands the correct selection and implementation of sensors and actuators (Durak et al., 2019; Oswald et al., 1999). The application of SRA-programming requires pupils to adopt a different approach to solving a programming problem from applying a linear programming approach (Slangen, 2016; Wyeth et al., 2003).

From research by López et al. (2021), it appears that iterations, conditionals and functions, the prominent concepts underlying SRA-programming, are difficult for primary school pupils to comprehend. More specifically, these concepts are operationalised through programming elements such as nested loops, "if–then-else", "wait until", "while" or functions with parameters. Pupils tend to avoid applying these complex concepts due to the higher level of abstraction involved (Werner et al., 2012). Only when a programming environment or task design appeals to the added value of complex programming concepts will pupils be triggered to use them in solving a programming problem (Wahl & Thomas, 2002). When introducing complex programming concepts to primary school pupils, game-based and robot programming environments provide a promising opportunity to illustrate and reveal their functions and applications (Chevalier et al., 2021; Dlab et al., 2019; Martinez et al., 2015). Previous research indicates that applying SRA thinking, including the use of sensory input to anticipate unforeseen, changing events in the task design, forces pupils to abandon linear thinking and offers them the opportunity to

effectively understand and apply complex programming concepts in a goal-oriented way (Fanchamps et al., 2020). SRA thinking involves logical, causal and conditional reasoning and the ability to establish cause/effect relationships when using sensor input to anticipate changes in the task design.

The influence of the characteristics of the learning environment appears to be very important in programming applications (Durak et al., 2019; Gross & Powers, 2005), in order for pupils to not only comprehend the coding environment used but also the programming concepts themselves (Williams et al., 2015). The characteristics of the programming environment can influence pupils' performance, and provide opportunities for assessing and providing feedback (Ahmed et al., 2018; Allison et al., 2002). Furthermore, the design of the programming environment can characterise the ways in which pupils perceive, interact with and respond to the environment (Gomes & Mendes, 2008). It is important that pupils develop programming skills in an environment that supports them in learning the basic concepts of programming (Gomes & Mendes, 2007; López et al., 2021; Zaharija et al., 2013). The programming learning environment must therefore create the conditions for understanding and applying certain abstract programming concepts (Sáez-López et al., 2019; Werner et al., 2012). The design of the task and the learning environment can help pupils to understand the effect of the programming intervention and can clarify the functions of concepts (Popat & Starkey, 2019; Wahl & Thomas, 2002). In terms of the comprehension and application of the basic concepts of programming, visual programming environments prove to be perfectly suited due to their imaginative power and low level of abstraction, and are easily accessible to enable pupils learn the basic concepts of programming required (Kaučič & Asič, 2011; Tsai, 2019).

Direct manipulation environments (DMEs) are powerful tools for creating a learning environment that can make programming understandable. Robotic DMEs are concrete, physical artefacts (robots/constellations) that can be controlled by programming that makes use of actuators and sensors (Jonassen, 2006; Rekimoto, 2000). DMEs offer a potentially rich context for learning, understanding and practicing programming, for understanding the concepts of robotics, and for developing (general) problem-solving skills. The use of robotic DMEs provides the possibility to obtain 'instant' feedback from the technology on pupils' thinking and acting (Slangen et al., 2011). Examples of such environments include TechnoLogica, K'nex, Fischertechnic, Arduino, Makeblock and Lego Mindstorms, which allow pupils to build controllable structures that can be programmed to perform predefined tasks (Jonassen, 2000; Slangen et al., 2008, 2011; Slangen et al., 2009). The efficient and yield-oriented use of DMEs places demands on the programming environment and the task design with regard to enabling a problem-solving approach. DMEs tend to use complex programming concepts (e.g. nested loops, if–then-else, wait until, while, functions with parameters, etc.).

Robotic and virtual environments are considered powerful tools for learning complex programming concepts (Caci et al., 2013a, 2013b; López et al., 2021). A tangible output can be experienced and perceived through physical representations, unlike a visual output, which uses more mental representations (Chevalier et al., 2022; Marshall, 2007). Moreover, a visual programming environment with a tangible output also differs from a visual programming environment with an on-screen

output in terms of the connections between the physical and digital representations (O'Malley & Fraser, 2004). Furthermore, it can be argued that a three-dimensional, physical representation provides different forms of information, immersion and engagement from a two-dimensional, visual, on-screen representation (Price et al., 2003). Some studies indicate that learning to program is more effective and meaningful when the learner operates a tangible and meaningful object (Horn & Bers, 2019; Papert, 1980; Resnick et al., 1990); other research claims that the visual characteristics of the more virtual world are a better way to develop mental representations of the program based on the structure of the data flow (Navarro-Prieto & Cañas, 2001; Segura et al., 2020). In order to explain the learning outcomes resulting from different output modalities in programming, more research is needed into the influence of the interaction between a visual programming environment and a tangible or visual type of output (Skulmowski et al., 2016; Zhu, 2021).

Visual programming environments use on-screen code elements that help novice programmers to easily understand and construct the process of programming (Price & Barnes, 2015). The advantages of these visual programming environments are that no specific syntax needs to be mastered and that the level of abstraction is low, due to the high extent of visualisation and generalisation (Weintrop & Wilensky, 2015). Visual programming environments are also intended to support the understanding and application of control flow structures (Chao, 2016). Due to their attractiveness, transparency and clarity, visual programming environments can help to increase user engagement in solving programming tasks (Asad et al., 2016). In a visual programming environment, a computer program to solve a computational problem is constructed by manipulating visual programming elements in order to formulate and design a solution to the problem (Sáez-López et al., 2016). Through the on-screen execution of the constructed program, direct visual feedback can be obtained from which the user can anticipate and determine the subsequent interventions by means of problem-solving actions (Moreno et al., 2011; Tsai, 2019). Through simulation, visual programming environments can provide more complex and richer functionality than the physical boundaries of artefacts in the material world allow. Direct feedback obtained from visual output may be experienced as more powerful than feedback obtained via tangible output (Caci et al., 2013a, 2013b; Sefidgar et al., 2017). Visual programming environments also often include integrated, directional incentives which provide the user with instant information on whether the programming solution is the optimal one, or whether it could be constructed more efficiently. These incentives provide guidance, and the user may decide to use them when support is needed (Karalekas et al., 2020). Seen from these perspectives, visual programming environments offer excellent opportunities for solving challenging programming problems and acquiring CT (Papadakis et al., 2016; Rose et al., 2017).

In programming with a visual output, the application and execution of each programming operation is displayed purely on a screen (Sapounidis et al., 2015). The information obtained from the programmed operation can be characterised as two-dimensional percipient (Mladenović et al., 2020; Price et al., 2003). The elaboration of programming actions appeals to the more abstract imagine ability and reasoning capacity of the user (Price & Barnes, 2015), but it is not possible to fall back on the tangible and physically perceptible (Horn & Bers, 2019;

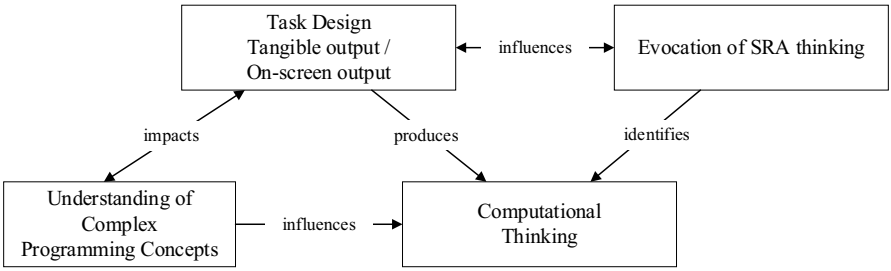

Fig. 1 Schematic representation of the conceptual model

O'Malley & Fraser, 2004; Sefidgar et al., 2017; Skulmowski et al., 2016). Nothing can be grasped in a hands-on way, and the execution of the programming action cannot be seen from more than one point of view (Sapounidis et al., 2015). When there is a physically perceptible, tangible output of the programming operation, concrete artefacts are controlled by the computer program (Chen et al., 2017; Jonassen, 2006). The information obtained from the execution of the programming intervention is perceptible in a three-dimensional way, from all points of view (Korkmaz, 2018). Imagination and reasoning abilities are stimulated at a low level of abstraction, and the execution is tangible at any moment (Bers, 2020; Ilieva, 2010; Wang et al., 2014). In addition, users can check their expectations of the execution in physical reality, at any point in time (Marshall, 2007).

Building on the above theoretical exploration, we hypothesise a relation between the SRA approach in a visual programming environment with different types of output and an influence on computational thinking. In addition, depending on the evocation of SRA thinking, we expect an influence on CT caused by a greater understanding of the concepts of programming. It is also expected that the more complex concepts of programming will initiate a more profound development of CT, and that the direct feedback obtained from visual output during visual programming will be more powerful than the feedback that can be derived from the execution of a visual program using a physical artefact. Our conceptual model, shown in Fig. 1, provides an overview of the relationships and interconnections between the independent and dependent variables, in which some connections are reciprocal.

Research question, sub-questions and hypotheses

Based on preliminary studies and research in the literature, our main research question is: What is the influence of the type of output in a visual SRA-programming environment on the development of CT and complex programming concepts among primary school pupils?

Supplementary to the main research question, our sub-questions are:

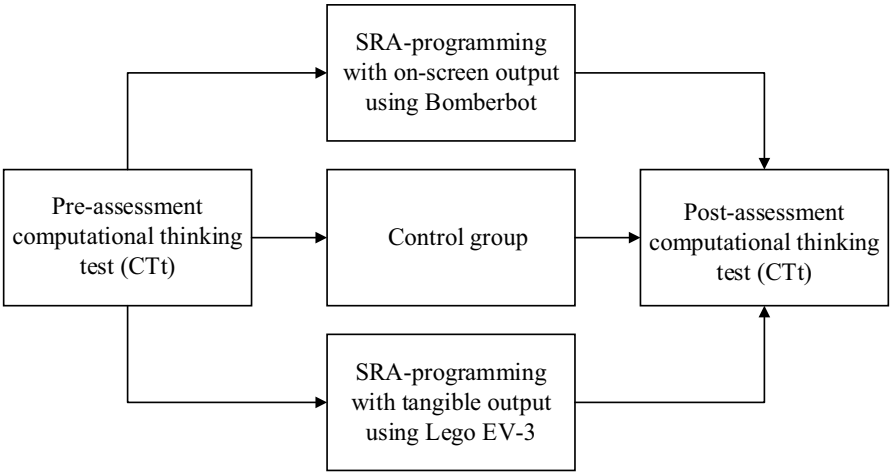

Fig. 2 Research design

1 To what extent can the influence on the development of CT be attributed to the evocation of SRA thinking and the type of output (tangible/on-screen) in SRA-programming?
2 What is the impact of visual SRA-programming environments (task design) on the understanding of complex programming concepts?
3 What is the influence of the understanding of complex programming concepts on CT?

These sub-questions result in the following hypotheses:

1. Pupils who apply SRA-programming in a visual programming environment show the development of CT.
2. Applying SRA-programming in a visual programming environment with visual, on-screen output leads to a higher level of development of CT compared to SRA-programming in a visual environment with tangible output.
3. The application of SRA-programming in a visual programming environment with visual, on-screen output has a greater impact on the understanding of complex programming concepts than SRA-programming in a visual environment with tangible output.

Method

This research should be seen as an exploratory approach to gain more insight into the effects of the difference in output on the development of CT. To this end, an exploratory study was conducted in which, by application of a pre-/post-test questionnaire survey, quantitative data were obtained (a) to determine the effect of

the intervention; (b) to assess the associated hypotheses and (c) to investigate the research questions.

As a pre- and post-assessment, a questionnaire on CT was applied. Quantitative data were collected from this exploration in order to answer the research question, sub-questions and hypotheses. In order to investigate the effect of the intervention, we used a pre-test/post-test design as illustrated in Fig. 2. For the dependent variable, this included the scores from a pre-/post-assessment of CT, and for the independent variables, we used two variations of visual SRA-programming interventions that differed in output (perception), namely Bomberbot, with on-screen output, Lego EV-3, with tangible output, and a control group that were not asked to program.

Participants

This research was conducted with pupils that ranged in age from 9 to 12, from grades 5 and 6[1] ($N=156$), from various primary schools selected at random from the South of the Netherlands, and from which two experimental groups (Lego $n=47$/Bomberbot $n=50$) and a control group ($n=59$) were randomly composed. None of the participating pupils were familiar with programming, apart from using basic computer programmes such as Word, PowerPoint and the Internet, and none had previously taken a CT test. The control group did not receive any programming offerings and followed the regular curriculum during the implementation of the study. All of the pupils from the primary schools involved were randomly assigned to one of the three conditions (Lego Ev-3, Bomberbot, control group). This ensured that the control group was a realistic reflection of the target group.

Materials

To answer our research questions, we used two visual programming environments with different types of output. This allowed pupils to learn complex programming concepts by applying SRA thinking. For each application, we aimed to deduce whether there was a difference in the effect on CT due to the different type of output. We used Bomberbot©, with a visual, on-screen output, and Lego EV-3 Mindstorms© robots, with a tangible output.

Bomberbot is a game-based, visual programming environment with an on-screen output in which pupils can learn the concepts of programming in a playful way. Bomberbot is a robot simulation that can be programmed to accomplish in virtual world tasks such as collecting stars, smashing obstacles and gems, opening treasure chests, and sliding unexpectedly across slippery surfaces. Using the 'drag and drop' method, programming commands need to be dragged from a command library into a worksheet in the correct sequence (El-Hamamsy et al., 2021). The command library provides the ability to work with fundamental programming concepts such as

[1] In this publication, we use the UK grade level system to represent the research population. Grades 5 and 6 in the UK correspond to groups 7 and 8 in the Netherlands.

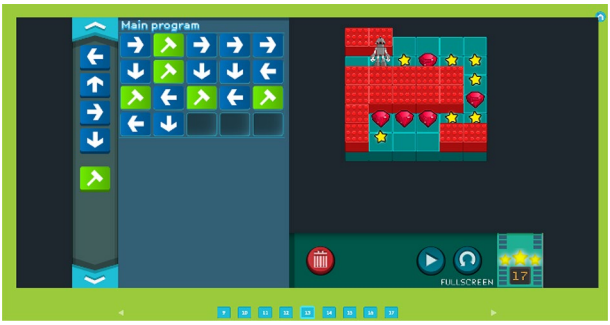

Fig. 3 Programming in Bomberbot©

iterations, conditionals and functions, which allows pupils to comprehend the principle of operation and effects of these concepts. Bomberbot is designed to provide the user with direct, self-correcting feedback and is characterised by a build-up from beginner to advanced level. The programming tasks to be performed are predefined, and consist of 20 missions to be carried out, with ascending complexity. The aim is to achieve all missions as efficiently as possible, allowing the user to earn various numbers of stars, ranging from three stars for the most efficient solution to one for each working solution. In Bomberbot, both the programming environment and the execution environment are represented simultaneously within the same on-screen image. Figure 3 shows the basics of programming in Bomberbot.

Lego EV-3 Mindstorms is a visual robotics programming environment in which tangible, perceptible robots are controlled by means of an interface. The application of Lego EV-3 is characterised by a direct effect in the physical space. By using 'drag and drop' programming blocks, which are arranged into a worksheet, the program controls a set of actuators and sensors. The controllable parameters and variables inside these blocks (for instance speed, direction, rotation, detection) can be influenced. Lego EV-3 teaches the applicability of basic concepts of programming such as iterations, conditionals and functions, and either predefined assignments or self-designed tasks can be used. In this research, we used 15 training missions and five final challenges in which a predefined robot, equipped with a push-button sensor and an ultrasonic sensor, had to be programmed to navigate through various labyrinth setups. The game element of the Lego challenge tasks is that SRA-programming ultimately leads to the most efficiently constructed program. This is determined through the application of SRA, based on the runtime of the robot when successfully completing each challenge task. In Lego EV-3 Mindstorms, the programming environment is separated from the physical task environment. Figure 4 shows a programming solution devised using Lego Mindstorms software, and Fig. 5 shows an example task in Lego EV-3.

To determine the level of CT between the pre- and post-measurements, we used the validated Computational Thinking test (CTt) (Román-González et al., 2017). This test makes it possible to generate information on the level of solving CT tasks (hypothesis 1), the understanding of the computational concepts involved

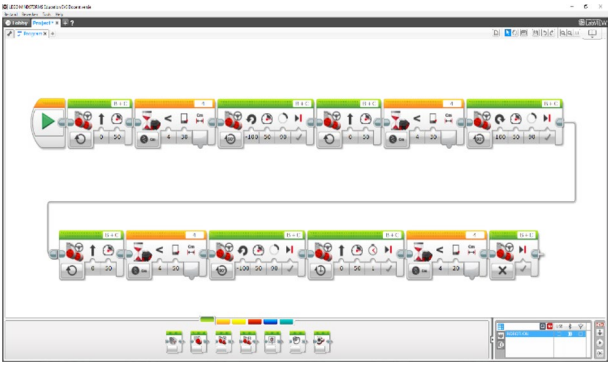

Fig. 4 SRA-programming in Lego EV-3 Mindstorms©

Fig. 5 Example of a labyrinth task in Lego EV-3©

(hypothesis 2), and the understanding of complex programming concepts (hypothesis 3). All pupils involved in this research completed this questionnaire individually. The questionnaire contained a total 28 items that relate to the various computational concepts involved, i.e. basic directions (28 items), loops (repeat times: 13 items, repeat until: 12 items), conditionals (if-simple: six items, if/else-complex: four items, while: four items) and functions (simple functions: four items, functions with parameters: zero items). The existence or nonexistence of nesting can be derived (19 items). The computational task required (completion: nine items, debugging: five items, sequencing: 14 items) to provide the right solution for each of the 28 questionnaire items can be deduced. To determine the reliability of the scale, we calculated Cronbach's alpha. It should be noted that a value for Cronbach's alpha of 0.70 is considered an acceptable reliability factor (Santos, 1999). The developers of the CTt indicate that for fifth and sixth graders ($N=176$), Cronbach's alpha should be $\alpha=0.721$ (Román-González et al., 2017). We measured a value of Cronbach's alpha of $\alpha=0.679$, thus almost complying with the required level of internal consistency for our scale with this particular sample. An explanation for this slightly lower

value for Cronbach's alpha can be found in the fact that the research sample size was smaller than that for which the designers of the CTt validated the test. In addition, the age category of the research population included in this study was towards the lower limit of the test, which may be the reason for the lower reliability. Taking this into account, the CTt performed almost as reported by the original authors, and the measurement results obtained were therefore be used as such.

Procedure

As a pre-test measurement (Fig. 2), all three groups completed the CTt. Following this, the group that were to program with Lego EV-3 Mindstorms received basic instruction and then completed 15 programming tasks, with five final challenge assignments in five 1-h sessions, by applying SRA-programming. The group that was to use Bomberbot completed, after a short introduction, 10 programming missions in five 1-h sessions using SRA-programming, each consisting of 15 programming tasks. The control group did not program with either of the programming environments. At the conclusion of the investigation, all three groups completed the CTt as a post-test measurement.

Results and data analysis

Our main research question, "What is the influence of the type of output in a visual SRA-programming environment on the development of CT and complex programming concepts among primary school pupils?", is answered by analysing the means for all the variables measured in this research. We aimed to explore (i) whether one of the two visual programming environments (Bomberbot and Lego EV-3) with different types of output, and if so which, led to significant differences with respect to the control group, and/or (ii) whether significant differences may occur in a comparison between the two programming environments. To this end, a variance analysis (Anova) and Levene's test were initially conducted for all variables to obtain preliminary indications and to assess whether equal variances should be assumed. Since specific hypotheses were formulated in this study, a subsequent contrast analysis was carried out to demonstrate possible significant effects and to confirm or reject these hypotheses. To make the magnitude of the effects visible, Cohen's d was calculated.

The pre- and post-measurement results from the CTt were entered into SPSS for quantitative data analysis, and the effects of the independent variables on the dependent variables were assessed. The differences in values were determined by comparing the means. In all of our statistical analyses, a significance level of 5% ($p = \leq 0.\ 05$) was assumed. The nature of the data met the conditions for the assumption of normality, indicating that the distribution of sample means (across independent samples) was normal. We tested whether our assumptions of the homogeneity of variances were violated ($p \leq 0.05$). Degrees of freedom were calculated, and a bootstrapping procedure was applied to re-estimate the standard error of the mean difference. The confidence interval was studied to assess the difference between the

means and to determine whether a value of zero was within the confidence interval. The Shapiro–Wilk test, used to demonstrate the normality of the variables, gave a value of $p = 0.126$, which was greater than the chosen alpha level of 0.05. The null hypothesis could not be rejected, as there was evidence that the tested data were normally distributed. Histograms also showed a normal distribution. It could therefore be assumed that all variables were normally distributed. The value for the extent of the effect size (Cohen's d) was calculated (it should be noted that $d = 0.2$ can be considered a small effect size, $d = 0.5$ represents a medium effect size, $d = 0.8$ indicates a large effect size, and any value above $d = 1.4$ is considered a very large effect) (Field, 2013).

Differences in the level of development of computational thinking

In order to provide a structured overview of the differences in the development of CT between the three groups, this aspect is divided into subcategories. The basis for this approach originates from the subdivision used by Román-González et al. (2017) in relation to the CTt. Table 1 shows the data for each of the subcategories for the three different groups.

An analysis of the means of the pre- and post-measurement results reveals that in comparison with the control group, the two groups that applied SRA-programming using Bomberbot and Lego EV-3 (a) solved more CT tasks successfully, (b) showed more control over loops, conditionals and functions, (c) showed more use of nesting and (d) applied sequencing, completion and debugging more often for the required task application. This can be deduced from the data, as both groups that applied SRA-programming with Bomberbot or Lego EV-3 showed higher average scores (M) in the post-assessment than the control group for all variables measured. The mean values should be considered as the average number of correctly answered questions per respondent, normalised to a value from zero to one. This increase can also be deduced from the percentage values that were calculated for each variable separately for each intervention group. This percentage calculation was applied because the three different intervention groups differed in terms of the number of respondents (Bomberbot: $n = 50$; Lego EV-3: $n = 47$; control group: $n = 59$). The percentage values per category were calculated on the basis of the number of items per category compared to the total number of items in the questionnaire, and this value was multiplied by the calculated mean (M). By illustrating the differences between the two intervention groups and the control group in the form of percentage values, the effect and impact of SRA-programming on CT could be objectively compared. For the combined categories, the percentages were calculated on the basis of the weighted averages. The category "loops: combined" is an aggregation of the subcategories "loops: repeat times" (13 items) and "loops: repeat until" (12 items). Since both subcategories have an overlap of three items, the category "loops: combined" is based on a total of 22 items.

The category "conditionals: combined" is an aggregation of the subcategories "conditionals: if-simple" (6 items), "conditionals: if/else" (4 items) and "conditionals: while" (4 items). Since there are two items that overlap between these three subcategories,

Table 1 Differences in computational thinking

Variable	Bomberbot				Lego EV-3				Control group			
	M	SD	Range	%	*M*	SD	Range	%	*M*	SD	Range	%
Pre-test: CT tasks correctly solved (28 items)	0.49	0.13	0.25–0.82	49.00	0.49	0.12	0.11–0.68	49.00	0.56	0.14	0.29–0.89	56.00
Post-test: CT tasks correctly solved (28 items)	0.61	0.14	0.36–0.96	61.00	0.60	0.13	0.36–0.86	60.00	0.55	0.15	0.29–0.89	55.00
Pre-test loops: repeat times (13 items)	0.41	0.16	0.08–0.85	19.04	0.45	0.16	0.00–0.69	20.89	0.50	0.17	0.15–0.85	23.21
Post-test loops: repeat times (13 items)	0.59	0.17	0.31–1.23	25.07	0.68	0.76	0.31–5.69	31.57	0.50	0.18	0.08–0.92	23.21
Pre-test loops: repeat until (12 items)	0.45	0.15	0.17–0.75	19.29	0.41	0.17	0.08–0.67	17.57	0.50	0.17	0.25–0.83	21.43
Post-test loops: repeat until (12 items)	0.54	0.17	0.17–1.00	23.14	0.51	0.17	0.17–0.83	21.86	0.50	0.21	0.08–0.92	21.43
Pre-test loops: combined (22 items)	0.43	0.13	0.20–0.80	33.79	0.43	0.12	0.08–0.64	33.79	0.50	0.14	0.20–0.76	39.29
Post-test loops: combined (22 items)	0.57	0.15	0.28–1.12	44.79	0.59	0.37	0.32–2.93	46.36	0.50	0.17	0.20–0.88	39.29
Pre-test conditionals: if-simple (6 items)	0.36	0.18	0.00–0.83	7.71	0.34	0.19	0.00–0.83	7.29	0.45	0.19	0.00–0.83	9.64
Post-test conditionals: if-simple (6 items)	0.52	0.22	0.00–1.00	11.14	0.42	0.20	0.00–1.00	9.00	0.40	0.20	0.00–0.83	8.57
Pre-test conditionals: if/else (4 items)	0.45	0.29	0.00–1.00	6.43	0.40	0.25	0.00–1.00	5.71	0.47	0.28	0.00–1.00	6.71
Post-test lconditionals: if/else (4 items)	0.58	0.32	0.00–1.00	8.29	0.53	0.25	0.00–1.00	7.57	0.46	0.31	0.00–1.00	6.57
Pre-test conditionals: while (4 items)	0.34	0.22	0.00–0.75	4.86	0.32	0.24	0.00–0.75	4.57	0.39	0.27	0.00–1.00	5.57
Post-test conditionals: while (4 items)	0.52	0.28	0.00–1.00	7.43	0.44	0.24	0.00–1.00	6.29	0.36	0.21	0.00–0.75	5.14
Pre-test conditionals: combined (12 items)	0.38	0.16	0.14–0.67	16.29	0.35	0.13	0.06–0.64	15.00	0.44	0.19	0.14–0.94	18.86
Post-test conditionals: combined (12 items)	0.54	0.21	0.06–1.00	23.14	0.46	0.16	0.14–0.83	19.71	0.40	0.18	0.08–0.78	17.14
Pre-test functions: simple (4 items)	0.37	0.25	0.00–1.00	5.29	0.47	0.30	0.00–1.00	6.71	0.50	0.31	0.00–1.00	7.14
Post-test functions: simple (4 items)	0.63	0.24	0.25–1.00	9.00	0.69	0.22	0.25–1.00	9.86	0.63	0.95	0.00–7.50	9.00
Pre-test use of nesting (19 items)	0.37	0.14	0.05–0.74	25.11	0.37	0.13	0.05–0.63	25.11	0.45	0.16	0.16–0.84	30.54
Post-test use of nesting (19 items)	0.54	0.17	0.16–0.95	36.64	0.50	0.14	0.21–0.79	33.93	0.44	0.17	0.21–0.84	29.86
Pre-test required task completion (9 items)	0.51	0.15	0.11–0.78	16.39	0.49	0.14	0.11–0.78	15.75	0.55	0.17	0.11–0.89	17.68
Post-test required task completion (9 items)	0.65	0.16	0.33–1.00	20.89	0.60	0.15	0.33–1.00	19.29	0.57	0.18	0.22–0.89	18.32
Pre-test debugging (5 items)	0.49	0.27	0.00–1.00	8.75	0.51	0.26	0.00–1.00	9.11	0.63	0.24	0.00–1.00	11.25
Post-test debugging (5 items)	0.60	0.27	0.00–1.00	10.71	0.63	0.25	0.00–1.00	11.25	0.57	0.27	0.00–1.00	10.18

Table 1 (continued)

Variable	Bomberbot				Lego EV-3				Control group			
	M	SD	Range	%	*M*	SD	Range	%	*M*	SD	Range	%
Pre-test sequencing (14 items)	0.46	0.16	0.14–0.86	23.00	0.48	0.15	0.14–0.86	24.00	0.53	0.15	0.29–0.93	26.50
Post-test sequencing (14 items)	0.61	0.16	0.36–0.93	30.50	0.58	0.15	0.21–0.93	29.00	0.51	0.17	0.21–0.93	25.50

M average, *SD* standard deviation, range = spread in measurement, % = percentage values related to the total number of items in the questionnaire multiplied by the calculated mean (M)

Table 2 Contrast analysis with a comparison of SRA-programming for all groups

Group	Bomberbot compared to control group			Lego EV-3 compared to control group			Bomberbot compared to Lego EV-3		
Variable	*t*	*p*	*d*	*t*	*p*	*d*	*t*	*p*	*d*
Total (28)	2.527	0.013*	0.409	1.810	0.072	0.293	−0.649	0.517–	−0.105
Loops: repeat times	1.161	0.247	0.188	2.150	0.033*	0.348	0.971	0.333	0.157
Loops: repeat until	1.319	0.189–	0.213	0.357	0.721	0.006	−0.904	0.367	−0.146
Loops: combined	1.554	0.122	0.251	2.078	0.039*	0.336	0.529	0.597	0.009
Conditionals: if-simple	3.085	0.002*	0.499	0.575	0.566	0.093	−2.336	0.019*	−0.383
Conditionals: if/else	2.059	0.041*	0.333	1.190	0.236	0.192	−0.803	0.423	−0.130
Conditionals: while	3.538	0.001*	0.572	1.814	0.072	0.293	−1.602	0.111	−0.259
Conditionals: combined	3.831	0.000*	0.619	1.660	0.099	0.268	−2.027	0.044*	−0.328
Functions	−0.054	0.957	0.000	0.456	0.649	0.007	0.490	0.625	0.079
Nesting	3.274	0.001*	0.529	1.980	0.049*	0.320	−1.193	0.235	−0.193
CT task: completion	2.469	0.015*	0.399	0.999	0.319	0.162	−1.374	0.171	−0.222
CT task: debugging	0.592	0.555	0.095	1.069	0.287	0.173	0.469	0.640	0.076
CT task: sequencing	2.920	0.004*	0.472	2.131	0.035*	0.345	−0.712	0.478	−0.115

Variable measurable value, total number of questions correct CT questionnaire, computational concept addressed loops, conditionals, functions, nesting; completion = completed by CT, debugging reformulating of problems, sequencing sequence, *t t* value contrast analysis, *p p* -value contrast analysis, d effect size based on Cohen's d

*Significant effect measured

the category "loops: combined" is assumed to contain a total of 12 items. Based on the measured values obtained, it can be stated that SRA-programming, with either Bomberbot or Lego EV-3, was the cause of this increase in comparison with the control group. This is despite the fact that for some variables, only a slight increase in the measured values in the post-test could be established. The control group did not score better in any category than either the Bomberbot and Lego EV-3 groups. In addition to the increase for the Bomberbot and Lego EV-3 programming interventions, it was striking that the control group showed a decline in the value measured for each variable.

Development in solving computational thinking issues at a higher level

In order to gain more insight into the impact of the interventions based on SRA-programming and whether this led to a higher level of CT, a contrast analysis with a three-group comparison was performed. Table 2 shows the data for the contrast analysis as applied to all variables, in a comparison for all groups.

Results of the contrast analysis and main significant effects

A contrast analysis of the total number of correctly solved CT tasks (28) shows that there was a significant difference, with a medium measurable effect, between the

group that programmed using Bomberbot and the control group t (153) = 2.527, p = 0.013, d = 0.409, and an almost significant difference with a small effect between the group that programmed using Lego EV-3 and the control group t (153) = 1.810, p = 0.072, d = 0.293. No significant difference could be measured between the groups that programmed using Bomberbot and Lego EV-3 (p = 0.517).

A contrast analysis of the differences in the application of "loops: repeat times" shows that no significant difference could be measured between the group that programmed using Bomberbot and the control group (p = 0.247), but there was a significant difference with a small effect between the group that programmed using Lego EV-3 and the control group t (153) = 2.150, p = 0.033, d = 0.348. No significant difference was found between the groups that programmed with Bomberbot and Lego EV-3 (p = 0.333).

A contrast analysis of the differences in the combined application of "loops: repeat times" and "loops: repeat until" shows that there was no significant difference between the group that programmed using Bomberbot and the control group (p = 0.122), but a significant difference with a small effect was measured between the group that programmed using Lego EV-3 and the control group t (153) = 2.078, p = 0.039, d = 0.336. No significant difference was measured between the groups that programmed using Bomberbot and Lego EV-3 (p = 0.597).

A contrast analysis of the differences in the application of "conditionals: if-simple" shows that there was a significant difference with a medium effect between the group that programmed using Bomberbot and the control group t (153) = 3.085, p = 0.002, d = 0.499, but no significant difference between the group that programmed using Lego EV-3 and the control group (p = 0.566). There was a significant difference, with a small, negative effect, between the groups that programmed using Bomberbot and Lego EV-3 t (153) = −2.336, p = 0.019, d = −0.383.

A contrast analysis of the differences in the application of "conditionals: if/else" shows that there was a significant difference with a medium effect between the group that programmed using Bomberbot and the control group t (153) = 2.059, p = 0.041, d = 0.333, but no significant difference between the group that programmed using Lego EV-3 and the control group (p = 0.236). No significant difference was measurable between the groups that programmed using Bomberbot and Lego EV-3 (p = 0.423).

A contrast analysis of the differences in the application of "conditionals: while" shows that a significant difference, with a medium to large effect, was measured between the group that programmed using Bomberbot and the control group t (153) = 3.538, p = 0.001, d = 0.572, and an almost significant difference with a small effect between the group that programmed using Lego EV-3 and the control group t (153) = 1.814, p = 0.072, d = 0.293. No significant difference was measurable between the groups that programmed using Bomberbot and Lego EV-3 (p = 0.111).

A contrast analysis of the differences in the combined application of "if-simple", "if/else" and "while" conditionals shows that there was a significant difference, with a medium to large effect, between the group that programmed using Bomberbot and the control group t (153) = 3.831, p = 0.000, d = 0.619, and an almost significant difference, with a small effect, between the group that programmed using Lego EV-3 and the control group t (153) = 1.660, p = 0.099, d = 0.268. A significant difference, with a small,

negative effect, was measurable between the groups that programmed using Bomberbot and Lego EV-3 $t\,(153) = -2.027$, $p = 0.044$, $d = -0.328$.

A contrast analysis concerning differences in the application of "nesting" shows that there is a significant difference with a medium effect between the group that programmed using Bomberbot and the control group $t\,(153) = 3.274$, $p = 0.001$, $d = 0.529$, and that a significant difference with a small effect was measurable between the group that programmed using Lego EV-3 and the control group $t\,(153) = 1.980$, $p = 0.049$, $d = 0.320$. No significant difference was measurable between the groups that programmed using Bomberbot and Lego EV-3 ($p = 0.235$).

A contrast analysis of the differences in the required task "completion" shows that a significant difference with a small effect was measurable between the group that programmed using Bomberbot and the control group $t\,(153) = 2.469$, $p = 0.015$, $d = 0.399$, but no significant difference between the group that programmed using Lego EV-3 and the control group ($p = 0.319$). No significant difference was measurable between the groups that programmed using Bomberbot and Lego EV-3 ($p = 0.171$).

A contrast analysis of the differences in the required task "sequencing" shows a significant difference with a medium effect was measurable between the group that programmed using Bomberbot and the control group $t\,(153) = 2.920$, $p = 0.004$, $d = 0.472$, and a significant difference with a small effect between the group that programmed using Lego EV-3 and the control group $t\,(153) = 2.131$, $p = 0.035$, $d = 0.345$. No significant difference was measurable between the groups that programmed using Bomberbot and Lego EV-3 ($p = 0.478$).

An analysis of the data obtained from this contrast analysis indicates that the influence of the different types of output by applying SRA-programming can have significant effects on the development of different characteristics of CT. Notable among these are the significant values that we have indicated with an * in Table 2. An interpretation of the reported results makes it clear that depending on the characteristics of the programming environment, a significant development of the (sub) characteristics of CT is measurable. A possible explanation for this may be that Bomberbot, with its visual, on-screen output, more effectively stimulates the use of conditional, cause-and-effect reasoning by means of the applied task design, and that Lego EV-3, with its tangible output, facilitates the understanding of iterations and nested loops and other types of linear thinking. The difference in the ability to perceive the impact of the programming intervention in Bomberbot more directly may also explain the significant results found. Bomberbot appears to be more focused on parallel programming interventions, whereas Lego EV-3 still leaves much room for finding a programming solution via linear programming. It is also noticeable that Bomberbot incorporates stimuli that encourage pupils to keep searching for the most optimal and efficient programming solution, which seems to have an effect in the development on sub-characteristics of CT.

Conclusions

From an examination of the data, it can be deduced that the control group initially showed a higher starting level (pre-assessment) for all variables present in this study, compared with the two other experimental conditions (Bomberbot and Lego EV-3).

However, the difference in the increases between the pre- and post-assessments of CT for the three different conditions indicates that the groups that received an SRA-programming intervention involving either Bomberbot or Lego EV-3 showed more growth (difference between pre- and post-assessment) and also achieved higher end-values in the post-measurement compared to the control group. In fact, the end-values for the control group did not increase, or hardly increased (or even decreased). This supports our claim that the application of SRA-programming is the cause of this identified increase. It can therefore be assumed that due to the application of SRA-programming in both visual programming environments, pupils (1) solved more CT tasks correctly; (2) solved more questions correctly which required the application of loops, conditionals, functions and nesting and (3) showed an increase in the application of CT (e.g. completion, debugging and sequencing) compared with the control group, thus indicating that SRA-programming contributes to a better understanding of complex programming concepts.

A further interpretation of the available data shows that there was a significant increase in the level of CT as a result of the application of SRA-programming using two different visual programming environments that differed in terms of output. With regard to the application of programming concepts and CT, both environments were comparable; the differences in the effects measured for both environments were caused as a result of the distinguishing features of the task design. In addition, the extent and type of visualisation and the visual program itself also seem to have a strong influence. The data analysed here indicate that both visual programming environments, with either a visual or tangible output, provided a substantial development in CT compared to the control group. Significant yields were measured for all variables considered in this research. The extent of the effect of applying SRA-programming was medium to large. The findings of this research indicate an impact on CT due to the use of SRA-programming.

The hypothesis that applying SRA-programming in a visual programming environment leads to a development of CT can be confirmed. This can be deduced from the total number of correctly solved CT tasks: both the Bomberbot and Lego EV-3 groups showed a substantial increase in the number of correctly solved CT tasks compared to the control group. For the group that programmed with Bomberbot, this development was significant, whereas for the group that programmed with Lego EV-3, this development was almost significant.

The hypothesis that the application of SRA-programming in a visual programming environment with visual, on-screen output leads to a higher level of development of CT compared to SRA-programming in a visual environment with tangible output can be confirmed based on the values for the required CT tasks of “completion” and “sequencing” considered in this research. For the CT task of “debugging”, no significance could be found, despite an increase in comparison with the control group.

The hypothesis that the application of SRA-programming in a visual programming environment with visual, on-screen output has more impact on the understanding of complex programming concepts than SRA-programming in a visual environment with tangible output can be confirmed based on our results for “conditionals” and “nesting”. The values for “loops: repeat times” and “loops: combined” indicate

that the application of Lego EV-3 causes a higher level of development than Bomberbot. The values for “loops: until” and “functions” indicate that despite an increase in comparison with the control group, no significance can be found for either visual programming environment, i.e. Bomberbot or Lego EV-3.

In general, it can be stated that visual SRA-programming environments with either a visual output or a tangible output are characterised by their own specific yield. In comparison with the control group, both groups showed a substantial development of CT through the application of SRA-programming. The application of SRA-programming within the visual programming environment of Bomberbot, with a visual output, gave a higher development of CT and more understanding of complex programming concepts than the use of SRA-programming with the visual programming environment of Lego EV-3, with a tangible output.

Discussion

The aim of this research was to find answers to the question of whether the type of output in a visual SRA-programming environment influences the development of CT and the understanding of complex programming concepts.

Our findings indicate that the use of visual SRA-programming environments with different types of output is a suitable way to establish the relationship between SRA-programming and the development of CT. The results obtained from our research show that visual SRA-programming with on-screen output predominantly leads to a higher development of CT and better understanding of complex programming concepts than visual SRA-programming with tangible output. We can also confirm that a visual programming environment with an on-screen output can create a higher level of understanding of complex programming concepts. This is in line with the assertions of Carlisle (2009), López et al. (2021), Werner et al. (2012) and Williams et al. (2015), who state that visual/on-screen oriented programming environments with an on-screen output provide an accessible, functional way to introduce these concepts into primary education. In this way, pupils acquire functional programming skills at a higher level, with a positive effect on CT.

Our results show that following the use of visual SRA-programming in both environments, pupils subsequently show development of computational concepts and, due to the influence of the SRA approach, can solve programming tasks of higher complexity. This is in line with claims made by Caci & D’Amico (2002) and Wong (2014), who state that anticipating differences between expected and observed events in the task design triggers a reconsideration of the underlying reasoning. The subsequent programming action ensures that pupils develop a higher level of cognitive ability, resulting in the ability to solve more complex programming tasks. The questions that then arise relate to why visual SRA-programming with an on-screen output gives rise to significant developments in terms of (1) the required CT tasks of “completion” and “sequencing”, and (2) complex programming concepts, "conditionals" and "nesting"; and (3) why a tangible output gives rise to significant developments in the complex programming concepts of "loops: repeat times" and "loops: combined"? We conjecture

that the underlying explanation is the more structured setup of the programming environment of Bomberbot, in comparison with the more open form implemented in Lego EV-3. A further explanation can be found in a study by Korkmaz (2018), who states that visual programming environments with a visual output make a more positive contribution to the development of logical-mathematical reasoning than those with a tangible output, but that a tangible output makes a more positive contribution to problem-solving skills.

Another aspect that may have influenced the results of the use of SRA-programming, apart from (1) the difference in output, (2) the similar difficulty of the task design and (3) the same drop-down method of programming, is that the two visual programming environments differ in the way feedback is provided to the user. In Bomberbot, feedback is provided as a constant guiding trigger, which provides pupils with input via visualisation as to whether the solution they have programmed is the most efficient one. This functions as an incentive, and more or less as a form of ongoing coaching. This view is endorsed by Papadakis et al. (2014), who state that in a visual on-screen programming environment, incentives can be used as a strong motivational tool for pupils to strive for the most efficient programming solution. However, Lego EV-3 is characterised by a more open, problem-solving approach to programming assignments. Pupils only receive feedback on the program that can be perceived by the concrete, tangible robot performing the programmed task as a reflection of the requirements of the environment and the task. This is in line with the perceptions of Asada et al. (1999), who claim that changes in a physical robotics environment evoke the programming actions to be taken, and can therefore be seen as directional feedback. The question that then arises is whether a visual incentive as a direct stimulus in a programming environment with a simulated reality works better than powerful, physical feedback as indirect stimulus in a tangible robotics environment. In a planned follow-up study, we expect to gain a better insight into the influence and relevance of the form of guidance that such programming environments provide to pupils.

Another noteworthy aspect is why pupils who are already familiar with the added functionality of the more efficient SRA-programming often do not apply it on a self-initiated base. One explanation is that pupils get stuck in the routine of a sequential trajectory, and/or that their first programming experiences have established the basis for choosing a linear approach without question. Our findings show that when using Bomberbot, pupils begin to understand initiated actions on which parallel programming routines rely. From a pedagogical perspective, it can be assumed that experience of programming with Bomberbot prior to experience of robotics programming could provide an opportunity to deter pupils from a linear, sequential programming approach.

For the control group, no measurable retention was caused by the application of the CTt. It may be a coincidence that this group showed a decline in CT, but it can be stated with certainty that no increase was detectable. It is also possible that pupils from the control group were less motivated to complete the CTt a second time, compared to the experimental groups. This may have been because pupils from the experimental groups were more likely to see a relationship between the assessment and the intervention. Further research is needed to explain this.

A further analysis and comparison between the use of Bomberbot and Lego reveals a number of striking findings. The in-depth examination of the contrast analysis (Table 2) shows that for the computational concept of "conditionals", Bomberbot showed significant improvement compared to Lego, and exactly the reverse was true for "loops". We conjecture that explanations for these differences can be derived from the specific characteristics of these programming environments. A primary characteristic of Bomberbot is that the visualisation of processes resulting from programming interventions, based on the on-screen display, is more imaginative. The ability to establish a relationship between a programming action and the execution of the programming process, and the perceptible effect of what each variable does, is immediately apparent in Bomberbot. Using Lego EV-3, this processing is indirectly visible via the tangible robot, and only the progress of the programming execution can be seen on the screen. From this, it can be inferred that the programming process is much more imitable in Bomberbot than in Lego.

We also believe that the design of each programming environment and the way in which information is provided to the user before, during and after the process may have a significant influence. The visualisations used and the instant feedback and support offered in Bomberbot can be conceived as direct guidance. The graphic design used for the tasks to be programmed and the requested application of conditionals in Bomberbot immediately appeal to parallel thinking as a characteristic of SRA-programming. This is in contrast to Lego Mindstorms software, where this indication of parallel thinking occurs indirectly, and the user must establish a relationship with parallel thinking independently. From this, we can conclude that the use of SRA-programming in Bomberbot is suggested more explicitly than in Lego EV-3 Mindstorms. The provision of incentives in Bomberbot also supports this statement.

This research contributes to the theory of CT and programming education. It also can be directional in terms of how programming can be operationalised within education. Based on our significant results, it can be concluded that the SRA approach is a promising concept for developing CT in a more profound and meaningful way. From the effects measured, it can be concluded that the application of SRA-programming, when programming in a visual environment, leads to an increased capability to solve programming tasks at a higher level of complexity. It does not matter what kind of output is used within a visual programming environment, as long as a SRA approach is applied. From this point of view, the user's preference for a particular type of output can be matched accordingly. Overall, it can be concluded that the design and specifications of the programming environment, rather than the difference in the type of output, partly determine the expected learning effect on specific characteristics of complex programming concepts, resulting in a higher level of CT.

Limitations and follow-up research

Several limitations and considerations can be identified regarding the results of this research. Due to these a certain lack of generalisability of the results obtained should be taken into account.

During the course of this research, it may be the case that non-experimental variables played a determining role in the final results. Reasons for this may include that pupils continued to work with programming environments at home or within their current primary school curriculum, or that pupils have developed over time as a result of their standard educational programme. In addition, our findings could also be explained by taking into account children's familiarity with computer games with visual output representing tangibles and/or their previous computer experience.

This research made use of two visual programming environments: Bomberbot, with an on-screen output, and Lego EV-3 Mindstorms, with a tangible output. In order to be able to generalise the results of our research, it should be replicated with other visual programming environments with different types of on-screen and tangible output. The issues of whether the nature of the programming task and the level of difficulty affect the outcome and learning effects should also be examined. There are also arguments that the use of visual programming environments with incentives provides a low threshold for giving guidance to users; the user is guided more explicitly through the tasks, and it is therefore questionable whether these incentives restrict freedom of choice.

It would also be interesting to further investigate (1) whether the use of SRA-programming in a visual programming environment in which a tangible output is first used and then a visual output (or vice versa) yields a greater understanding of complex programming concepts; (2) whether this can be attributed to the application of SRA and (3) to what extent this results in the subsequent measurable development of CT. We note that SRA-programming of robotics with tangible output involves a very different form of application of SRA compared to visual SRA-programming with on-screen output. Learning to apply SRA in one environment may have (dis) advantages in the other. In follow-up research, it would be worthwhile to further clarify the relationship between the use of SRA-programming, the sequence in which the type of programming environment is applied, and possible differences in the development of CT.

In this research, there was relatively little time available (five sessions of 1 h each) for pupils to learn SRA-programming. If more time had been available, this may have led to different results; for instance, pupils may have gained more understanding of the complex concepts of programming.

In terms of definition and assessing the development of CT among primary school pupils, it should be noted that CT is still a developing concept. The findings and results reported in our research relate to the interpretation of CT as elaborated by Román-González et al. (2017), and as operationalised in their validated instrument to identify and measure the development of CT. In further research, it would be valuable to consider other interpretations relating to CT.

In this research, the researcher also acted as the supporting supervisor. It is important to consider that initiating, supporting and supervising programming activities requires specific competences from the supporting teacher. Moreover, a teacher should be well equipped to adequately facilitate and guide such activities. In subsequent research, guidance will be provided by regular teachers who are competent with regard to the requirements of pedagogical content knowledge and effective coaching.

Acknowledgements The authors would like to thank Bomberbot Netherlands and Heutink Netherlands for making the programming environments available, and for their cooperation. We would also like to thank the randomly selected primary schools for their participation in this research.

Author contributions All authors contributed to the study conception and design. Material preparation, data collection and analysis were performed by NF. The first draft of the manuscript was written by NF and all authors commented on previous versions of the manuscript. All authors read and approved the final manuscript.

Funding Not applicable.

Data availability Can be requested from the first author.

Code availability Retrievable on request from Bomberbot© and Heutink with personal login code.

Declarations

Conflict of interest The authors declare that they have no conflicts of interest/no competing interests.

Ethical approval The Ethical research board (cETO) of the Open University of the Netherlands has assessed the proposed research and concluded that this research is in line with the rules and regulations and the ethical codes for research in Human Subjects (reference: U2019/01324/SVW).

Consent to participate Written informed consent was obtained from the parents of the individual participants.

Consent to publish The parents of the individual participants consented the data obtained to be published.

References

Ahmed, I., Lubold, N., & Walker, E. (2018). ROBIN: using a programmable robot to provide feedback and encouragement on programming tasks. In: Penstein Rosé C. et al. (Eds.), *Artificial Intelligence in Education. AIED 2018. Lecture notes in computer science* (Vol. 10948, pp. 9–13). Cham: Springer. https://doi.org/10.1007/978-3-319-93846-2_2

Allison, I., Orton, P., & Powell, H. (2002). *A virtual learning environment for introductory programming*. Paper presented at the 3rd conference of the learning and teaching support network: Subject centre of information and computer sciences, Loughborough, UK. Loughborough University (pp. 48–52).

Asad, K., Tibi, M., & Raiyn, J. (2016). Primary school pupils' attitudes toward learning programming through visual interactive environments. *World Journal of Education, 6*(5), 20–26. https://doi.org/10.5430/wje.v6n5p20

Asada, M., Kitano, H., Noda, I., & Veloso, M. (1999). RoboCup: Today and tomorrow—What we have learned. *Artificial Intelligence, 110*(2), 193–214.

Atman Uslu, N., & Usluel, Y. K. (2019). Predicting technology integration based on a conceptual framework for ICT use in education. *Technology, Pedagogy and Education, 28*(5), 517–531. https://doi.org/10.1080/1475939X.2019.1668293

Basu, S., Biswas, G., Sengupta, P., Dickes, A., Kinnebrew, J. S., & Clark, D. (2016). Identifying middle school students' challenges in computational thinking-based science learning. *Research and Practice in Technology Enhanced Learning, 11*(1), 13. https://doi.org/10.1186/s41039-016-0036-2

Bers, M. U. (2020). *Coding as a playground: Programming and computational thinking in the early childhood classroom*. Routledge.

Brennan, K., & Resnick, M. (2012). *Using artifact-based interviews to study the development of computational thinking in interactive media design.* Paper presented at annual American Educational Research Association meeting, Vancouver, BC, Canada (pp. 1–25).

Caci, B., & D'Amico, A. (2002). *Children's cognitive abilities in construction and programming robots.* Paper presented at the 11th IEEE International Workshop on Robot and Human Interactive Communication, Berlin, Germany (pp. 189–191). https://doi.org/10.1109/ROMAN.2002.1045620

Caci, B., Chiazzese, G., & D'Amico, A. (2013a). Robotic and virtual world programming labs to stimulate reasoning and visual-spatial abilities. *Procedia-Social and Behavioral Sciences, 93*, 1493–1497.

Caci, B., D'Amico, A., & Chiazzese, G. (2013b). Robotics and virtual worlds: An experiential learning lab. *Biologically inspired cognitive architectures 2012* (pp. 83–87). Springer.

Carlisle, M. C. (2009). Raptor: A visual programming environment for teaching object-oriented programming. *Journal of Computing Sciences in Colleges, 24*(4), 275–281.

Catlin, D., & Woollard, J. (2014). *Educational robots and computational thinking.* Paper presented at the 4th International Workshop Teaching Robotics, Teaching with Robotics & 5th International Conference Robotics in Education, Padova, Italy (pp. 144–151).

Chalmers, C. (2018). Robotics and computational thinking in primary school. *International Journal of Child-Computer Interaction, 17*, 93–100. https://doi.org/10.1016/j.ijcci.2018.06.005

Chao, P.-Y. (2016). Exploring students' computational practice, design and performance of problem-solving through a visual programming environment. *Computers & Education, 95*, 202–215. https://doi.org/10.1016/j.compedu.2016.01.010

Chen, G., Shen, J., Barth-Cohen, L., Jiang, S., Huang, X., & Eltoukhy, M. (2017). Assessing elementary students' computational thinking in everyday reasoning and robotics programming. *Computers & Education, 109*, 162–175. https://doi.org/10.1016/j.compedu.2017.03.001

Chevalier, M., El-Hamamsy, L., Giang, C., Bruno, B., & Mondada, F. (2021). Teachers' perspective on fostering computational thinking through educational robotics. *arXiv preprint* arXiv:2105.04980.

Chevalier, M., Giang, C., El-Hamamsy, L., Bonnet, E., Papaspyros, V., Pellet, J.-P., Audrin, C., Romero, M., Baumberger, B., & Mondada, F. (2022). The role of feedback and guidance as intervention methods to foster computational thinking in educational robotics learning activities for primary school. *Computers & Education*. https://doi.org/10.1007/978-3-030-82544-7_17

Corral, J. M. R., Ruiz-Rube, I., Balcells, A. C., Mota-Macías, J. M., Morgado-Estévez, A., & Dodero, J. M. (2019). A study on the suitability of visual languages for non-expert robot programmers. *IEEE Access, 7*, 17535–17550. https://doi.org/10.1109/ACCESS.2019.2895913

Denning, P. J., & Tedre, M. (2019). *Computational thinking*. MIT Press.

Dlab, M. H., Hoić-Božić, N., Anđelić, M., & Botički, I. (2019). *Digital games and tools for development of computational thinking in primary school.* Paper presented at the International Conference on Management, Economics & Social Science-ICMESS. Changsha, China (pp. 1–6).

Dummer, G. (2017). *Computational thinking*. Paper presented at the Panama Conferentie, Utrecht, Netherlands.

Durak, H. Y., Yilmaz, F. G. K., & Yilmaz, R. (2019). Computational thinking, programming self-efficacy, problem solving and experiences in the programming process conducted with robotic activities. *Contemporary Educational Technology, 10*(2), 173–197. https://doi.org/10.30935/cet.554493

Edwards, S. (2005). Identifying the factors that influence computer use in the early childhood classroom. *Australasian Journal of Educational Technology, 21*(2). https://doi.org/10.14742/ajet.1334

El-Hamamsy, L., Papaspyros, V., Kangur, T., Mathex, L., Giang, C., Skweres, M., Bruno, B., & Mondada, F. (2021). Exploring a handwriting programming language for educational robots. In M. Merdan, W. Lepuschitz, G. Koppensteiner, R. Balogh, & D. Obdržálek (Eds.), *Robotics in education. RiE 2021. Advances in intelligent systems and computing* (Vol. 1359). Cham: Springer. https://doi.org/10.1007/978-3-030-82544-7_25

Estapa, A., Hutchison, A., & Nadolny, L. (2018). Recommendations to support computational thinking in the elementary classroom. *Technology and Engineering Teacher, 77*(4), 25–29.

Fanchamps, N., Specht, M., Hennissen, P., & Slangen, L. (2020). *The effect of teacher interventions and SRA robot programming on the development of computational thinking.* Paper presented at the International Conference on Computational Thinking Education 2020, Hong Kong (p.p 69–72).

Fanchamps, N., Slangen, L., Hennissen, P., & Specht, M. (2019). The influence of SRA-programming on algorithmic thinking and self-efficacy using Lego robotics in two types of Instruction. *International Journal of Technology and Design Education*. https://doi.org/10.1007/s10798-019-09559-9

Fanchamps, N., Slangen, L., Specht, M., & Hennissen, P. (2021). The impact of SRA-programming on computational thinking in a visual oriented programming environment. *Education and Information Technologies, 26*(5), 6479–98.

Field, A. (2013). *Discovering statistics using IBM SPSS statistics*. Sage.

Gomes, A., & Mendes, A. J. (2007). *An environment to improve programming education.* Paper presented at the 2007 International Conference on Computer Systems and Technologies, Bulgaria (pp. 1–6). https://doi.org/10.1145/1330598.1330691

Gomes, A., & Mendes, A. (2008). *A study on student's characteristics and programming learning.* In J. Luca & E. Weippl (Eds.), *Proceedings of ED-MEDIA 2008--World Conference on Educational Multimedia, Hypermedia & Telecommunications* (pp. 2895–2904). Vienna, Austria: Association for the Advancement of Computing in Education (AACE).

Gross, P., & Powers, K. (2005). *Evaluating assessments of novice programming environments.* Paper presented at the First International Workshop on Computing Education Research, Seattle, Washington, USA (pp. 99–110). https://doi.org/10.1145/1089786.1089796

Hernandez, R. M. (2017). Impact of ICT on Education: Challenges and perspectives. *Journal of Educational Psychology, 5*(1), 337–347. https://doi.org/10.20511/pyr2017.v5n1.149

Horn, M. S., Solovey, E. T., Crouser, R. J., & Jacob, R. J. (2009). *Comparing the use of tangible and graphical programming languages for informal science education.* Paper presented at the SIGCHI Conference on Human Factors in Computing Systems. New York, USA (pp. 975–984). https://doi.org/10.1145/1518701.1518851

Horn, M., & Bers, M. (2019). Tangible computing. In S. A. Fincher & A. V. Robins (Eds.), *The Cambridge handbook of computing education research* (Vol. 1, pp. 663–678). Cambridge University Press.

Hsu, T.-C., Chang, S.-C., & Hung, Y.-T. (2018). How to learn and how to teach computational thinking: Suggestions based on a review of the literature. *Computers & Education, 126*, 296–310. https://doi.org/10.1016/j.compedu.2018.07.004

Iivari, N., Sharma, S., & Ventä-Olkkonen, L. (2020). Digital transformation of everyday life–How COVID-19 pandemic transformed the basic education of the young generation and why information management research should care? *International Journal of Information Management, 55*(102183), 1–6. https://doi.org/10.1016/j.ijinfomgt.2020.102183

Ilieva, V. (2010). *Robotics in the primary school. How to do it?* Paper presented at the Intl. Conf. on Simulation, Modeling And Programming For Autonomous Robots, Darmstad, Germany (pp. 596–605).

Israel-Fishelson, R., & Hershkovitz, A. (2022). Studying interrelations of computational thinking and creativity: A scoping review (2011–2020). *Computers & Education, 176*, 104353. https://doi.org/10.1016/j.compedu.2021.104353

Jonassen, D. H. (2000). *Computers as mindtools for schools: Engaging critical thinking*. Prentice Hall.

Jonassen, D. H. (2006). *Modeling with technology: Mindtools for conceptual change*. Pearson Merrill Prentice Hall.

Kafai, Y. B., & Burke, Q. (2013). Computer programming goes back to school. *Phi Delta Kappan, 95*(1), 61–65. https://doi.org/10.1177/003172171309500111

Karalekas, G., Vologiannidis, S., & Kalomiros, J. (2020). EUROPA: A case study for teaching sensors, data acquisition and robotics via a ROS-based educational robot. *Sensors, 20*(9), 2469.

Kaučič, B., & Asič, T. (2011). *Improving introductory programming with Scratch?* Paper presented at the 2011 34th International Convention MIPRO. Opatija, Croatia (p.p 1095–1100).

Kim, D.-H., & Kim, J.-H. (2003). A real-time limit-cycle navigation method for fast mobile robots and its application to robot soccer. *Robotics and Autonomous Systems, 42*(1), 17–30. https://doi.org/10.1016/S0921-8890(02)00311-1

Kong, S.-C., & Abelson, H. (2019). *Computational thinking education*. Springer Nature.

Korkmaz, Ö. (2018). The effect of scratch- and Lego Mindstorms Ev3-based programming activities on academic achievement, problem-solving skills and logical-mathematical thinking skills of students. *Malaysian Online Journal of Educational Sciences, 4*(3), 73–88.

Krugman, M. (2004). *Teaching behavior based robotics through advanced robocamps*. Paper presented at the 34th Annual Frontiers in Education, 2004. FIE 2004 (pp. F3D-1). doi: https://doi.org/10.1109/FIE.2004.1408624.

Kyriazopoulos, I., Koutromanos, G., Voudouri, A., & Galani, A. (2022). Educational robotics in primary education: A systematic literature review. *Research Anthology on Computational Thinking, Programming, and Robotics in the Classroom*. https://doi.org/10.4018/978-1-6684-2411-7.ch034

Lai, R. P. (2021). Beyond programming: A computer-based assessment of computational thinking competency. *ACM Transactions on Computing Education (TOCE), 22*(2), 1–27.

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J., & Werner, L. (2011). Computational thinking for youth in practice. *ACM Inroads, 2*(1), 32–37. https://doi.org/10.1145/1929887.1929902

Leifheit, L., Jabs, J., Ninaus, M., Moeller, K., & Ostermann, K. (2018). *Programming unplugged: An evaluation of game-based methods for teaching computational thinking in primary school.* Paper presented at the ECGBL 2018 12th European Conference on Game-Based Learning. Sophia Antipolis, France. (pp. 344–353).

Lith, P. V. (2006). *Masterclass robotica*. Elektuur.

López, J. M. S., Otero, R. B., & García-Cervigón, S. D. L. (2021). Introducing robotics and block programming in elementary education. *Revista Iberoamericana De Educación a Distancia, 24*(1), 95–113. https://doi.org/10.5944/ried.24.1.27649

Luxton-Reilly, A. (2016). *Learning to program is easy.* Paper presented at the 2016 ACM Conference on Innovation and Technology in Computer Science Education, Arequipa, Peru. Association for Computing Machinery. New York, USA (p.p 284–289). doi: https://doi.org/10.1145/2899415.2899432

Marshall, P. (2007). *Do tangible interfaces enhance learning?* Paper presented at the 1st International Conference on Tangible and Embedded Interaction, Baton Rouge, LA, USA (pp. 163–170). doi: https://doi.org/10.1145/1226969.1227004

Martinez, C., Gomez, M. J., & Benotti, L. (2015). *A comparison of preschool and elementary school children learning computer science concepts through a multilanguage robot programming platform.* Paper presented at the 2015 ACM Conference on Innovation and Technology in Computer Science Education. New York, USA (pp. 159–164). https://doi.org/10.1145/2729094.2742599

Mladenović, M., Žanko, Ž, & AglićČuvić, M. (2020). The impact of using program visualization techniques on learning basic programming concepts at the K–12 level. *Computer Applications in Engineering Education, 29*(1), 145–159. https://doi.org/10.1002/cae.22315

Moreno, R., Ozogul, G., & Reisslein, M. (2011). Teaching with concrete and abstract visual representations: Effects on students' problem solving, problem representations, and learning perceptions. *Journal of Educational Psychology, 103*(1), 32. https://doi.org/10.1037/a0021995

Navarro-Prieto, R., & Cañas, J. J. (2001). Are visual programming languages better? The role of imagery in program comprehension. *International Journal of Human-Computer Studies, 54*(6), 799–829. https://doi.org/10.1006/ijhc.2000.0465

Nouri, J., Zhang, L., Mannila, L., & Norén, E. (2020). Development of computational thinking, digital competence and 21st century skills when learning programming in K-9. *Education Inquiry, 11*(1), 1–17. https://doi.org/10.1080/20004508.2019.1627844

O'Malley, C., & Fraser, D. S. (2004). *Literature review in learning with tangible technologies*. In Learning with tangible technologies. A NESTA futurelab research report—Report 12. 2004. Bristol, UK.

Oswald, N., Becht, M., Buchheim, T., Hetzel, G., Kindermann, G., Lafrenz, R., et al. (1999). CoPS-Team Description. *RoboCup-99: Robot Soccer World Cup III*. In M. Veloso, E. Pagello, H. Kitano (Eds.), *RoboCup-99: Robot Soccer World Cup III. Lecture Notes in Computer Science* (Vol. 1856). *Lecture Notes in Artificial Intelligence*. Berlin, Heidelberg, Germany.

Papadakis, S., Kalogiannakis, M., Orfanakis, V., & Zaranis, N. (2014). *Novice programming environments. Scratch & app inventor: A first comparison.* Paper presented at the 2014 Workshop on Interaction Design in Educational Environments. New York, USA (pp. 1–7). https://doi.org/10.1145/2643604.2643613

Papadakis, S., Kalogiannakis, M., & Zaranis, N. (2016). Developing fundamental programming concepts and computational thinking with ScratchJr in preschool education: A case study. *International Journal of Mobile Learning and Organisation, 10*(3), 187–202.

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

Popat, S., & Starkey, L. (2019). Learning to code or coding to learn? A systematic review. *Computers & Education, 128*, 365–376. https://doi.org/10.1016/j.compedu.2018.10.005

Price, T., & Barnes, T. (2015). *Comparing textual and block interfaces in a novice programming environment*. Paper presented at the Eleventh Annual International Conference on International Computing Education Research. New York, USA (pp. 91–99). https://doi.org/10.1145/2787622.2787712

Price, S., Rogers, Y., Scaife, M., Stanton, D., & Neale, H. (2003). Using 'tangibles' to promote novel forms of playful learning. *Interacting with Computers, 15*(2), 169–185.

Ray, P. P. (2017). A survey on visual programming languages in internet of things. *Scientific Programming, 2017*, 1–6. https://doi.org/10.1155/2017/1231430

Rekimoto, J. (2000). *Multiple-computer user interfaces: Beyond the desktop direct manipulation environments.* Paper presented at the Conference on Human Factors in Computing Systems, The Hague, Netherlands. Association for Computing Machinery. New York, USA (pp. 6–7). https://doi.org/10.1145/633292.633297

Resnick, M., Ocko, S., & Papert, S. (1990). LEGO/logo–learning through and about design. *Epistemology and learning group.* MIT Media Laboratory Cambridge.

Rich, P., & Browning, S. F. (2022). Using Dr. Scratch as a formative feedback tool to assess computational thinking. *Research anthology on computational thinking, programming, and robotics in the classroom* (pp. 550–572). IGI Global. https://doi.org/10.4018/978-1-6684-2411-7.ch026

Román-González, M., Pérez-González, J.-C., & Jiménez-Fernández, C. (2017). Which cognitive abilities underlie computational thinking? Criterion validity of the computational thinking test. *Computers in Human Behavior, 72*, 678–691. https://doi.org/10.1016/j.chb.2016.08.047

Rose, S., Habgood, J., & Jay, T. (2017). An exploration of the role of visual programming tools in the development of young children's computational thinking. *Electronic Journal of e-Learning, 15*(4), 297–309.

Sáez-López, J.-M., Román-González, M., & 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 & Education, 97*, 129–141. https://doi.org/10.1016/j.compedu.2016.03.003

Sáez-López, J.-M., Sevillano-García, M.-L., & 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, 67*(6), 1405–1425. https://doi.org/10.1007/s11423-019-09648-5

Santos, J. R. A. (1999). Cronbach's alpha: A tool for assessing the reliability of scales. *Journal of Extension, 37*(2), 1–5.

Sapounidis, T., Demetriadis, S., & Stamelos, I. (2015). Evaluating children performance with graphical and tangible robot programming tools. *Personal and Ubiquitous Computing, 19*(1), 225–237. https://doi.org/10.1007/s00779-014-0774-3

Sefidgar, Y. S., Agarwal, P., & Cakmak, M. (2017). *Situated tangible robot programming.* Paper presented at the 2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI), Vienna, Austria (pp. 473–482).

Segura, R. J., del Pino, F. J., Ogáyar, C. J., & Rueda, A. J. (2020). VR-OCKS: A virtual reality game for learning the basic concepts of programming. *Computer Applications in Engineering Education, 28*(1), 31–41. https://doi.org/10.1002/cae.22172

Skulmowski, A., Pradel, S., Kühnert, T., Brunnett, G., & Rey, G. D. (2016). Embodied learning using a tangible user interface: The effects of haptic perception and selective pointing on a spatial learning task. *Computers & Education, 92*, 64–75. https://doi.org/10.1016/j.compedu.2015.10.011

Slangen, L. (2016). *Teaching robotics in primary school.* PhD thesis, Eindhoven University of Technology, Eindhoven. Retrieved from https://pure.tue.nl/ws/files/25754482/20160630_CO_Slangen.pdf. Accessed 26 Jan 2022

Slangen, L., Fanchamps, N., & Kommers, P. (2008). A case study about supporting the development of thinking by means of ICT and concretisation tools. *International Journal of Continuing Engineering Education and Life-Long Learning, 18*(3), 305–322. https://doi.org/10.1504/IJCEELL.2008.018834

Slangen, L., Keulen, H. V., & Gravemeijer, K. (2011). What pupils can learn from working with robotic direct manipulation environments. *International Journal of Technology and Design Education, 21*(4), 449–469. https://doi.org/10.1007/s10798-010-9130-8

Slangen, L., Keulen, H. V., & Jochems, W. (2009). De bijdrage van direct manipulation environments aan de ontwikkeling van technische geletterdheid in de basisschool. In *Onderzoek naar*

Wetenschap en Techniek in de Basisschool (pp. 115–131). Den Haag, Netherlands: Platform Béta Techniek.

SLO. (2017). Curriculum van de toekomst. Retrieved from http://curriculumvandetoekomst.slo.nl/21e-eeuwse-vaardigheden. Accessed 18 Nov 2021

Stamati, M. (2020). The importance of ICT in primary education: Interpretive schemes and practices in the island of Lesvos. *Multilingual Academic Journal of Education and Social Sciences, 8*(1), 168–182. https://doi.org/10.46886/MAJESS/v8-i1/7276

Tedre, M., & Denning, P. J. (2016). *The long quest for computational thinking*. Paper presented at the 16th Koli Calling International Conference on Computing Education Research, Koli, Finland (pp. 120–129). https://doi.org/10.1145/2999541.2999542

Tsai, C.-Y. (2019). Improving students' understanding of basic programming concepts through visual programming language: The role of self-efficacy. *Computers in Human Behavior, 95*, 224–232. https://doi.org/10.1016/j.chb.2018.11.038

Tsarava, K., Moeller, K., Román-González, M., Golle, J., Leifheit, L., Butz, M. V., & Ninaus, M. (2021). A cognitive definition of computational thinking in primary education. *Computers & Education, 179*, 104425. https://doi.org/10.1016/j.compedu.2021.104425

Voskoglou, M. G., & Buckley, S. (2012). Problem solving and computational thinking in a learning environment. *Egyptian Computer Science Journal, 36*(4), 18.

Vourletsis, I., & Politis, P. (2021). Exploring the effect of remixing stories and games on the development of students' computational thinking. *Computers and Education Open*. https://doi.org/10.1016/j.caeo.2021.100069

Wahl, F. M., & Thomas, U. (2002). Robot programming: From simple moves to complex robot tasks. *Institute for robotics and process control* (pp. 1–16). Technical University of Brawnschweig.

Wang, D., Wang, T., & Liu, Z. (2014). A tangible programming tool for children to cultivate computational thinking. *The Scientific World Journal, 2014*, 1–10. https://doi.org/10.1155/2014/428080

Weintrop, D. (2019). Block-based programming in computer science education. *Communications of the ACM, 62*(8), 22–25. https://doi.org/10.1145/3341221

Weintrop, D., & Wilensky, U. (2015). *To block or not to block, that is the question: Students' perceptions of blocks-based programming*. Paper presented at the 14th International Conference on Interaction Design and Children, Medford, MA, USA (pp. 199–208). https://doi.org/10.1145/2771839.2771860

Werner, L., Campe, S., & Denner, J. (2012). *Children learning computer science concepts via Alice game-programming*. Paper presented at the 43rd ACM Technical Symposium on Computer Science Education. New Your, USA (pp. 427–432). https://doi.org/10.1145/2157136.2157263

Williams, C., Alafghani, E., Daley, A., Gregory, K., & Rydzewski, M. (2015). *Teaching programming concepts to elementary students*. Paper presented at the 2015 IEEE Frontiers in Education Conference (FIE), El Paso, TX, USA (pp. 1–9). https://doi.org/10.1109/FIE.2015.7344134

Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33–35. https://doi.org/10.1145/1118178.1118215

Wong, L. L. (2014). Rethinking the sense-plan-act abstraction: A model attention and selection framework for task-relevant estimation. Paper presented at the Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence, Quebec, Canada (pp. 71–72).

Wyeth, P., Venz, M., & Wyeth, G. (2003). Scaffolding children's robot building and programming activities. In D. Polani, B. Browning, A. Bonarini, & K. Yoshida (Eds.), *RoboCup 2003: Robot Soccer World Cup VII. RoboCup 2003. Lecture notes in computer science.* (Vol. 3020). Springer. https://doi.org/10.1007/978-3-540-25940-4_27

Yadav, A., Hong, H., & Stephenson, C. (2016). Computational thinking for all: Pedagogical approaches to embedding 21st century problem solving in K-12 classrooms. *TechTrends, 60*, 565–568. https://doi.org/10.1007/s11528-016-0087-7

Yelland, N. (2005). The future is now: A review of the literature on the use of computers in early childhood education (1994–2004). *AACE Journal, 13*(3), 201–232.

Zaharija, G., Mladenović, S., & Boljat, I. (2013). Introducing basic programming concepts to elementary school children. *Procedia-Social and Behavioral Sciences, 106*, 1576–1584. https://doi.org/10.1016/j.sbspro.2013.12.178

Zapata-Cáceres, M., Martín-Barroso, E., & Román-González, M. (2020). *Computational thinking test for beginners: Design and content validation*. Paper presented at the 2020 IEEE Global Engineering Education Conference (EDUCON), Porto, Portugal (pp. 1905–1914). https://doi.org/10.1109/educon45650.2020.9125368

Zhang, L., & Nouri, J. (2019). A systematic review of learning computational thinking through scratch in K-9. *Computers & Education, 141*, 103607. https://doi.org/10.1016/j.compedu.2019.103607

Zhu, K. (2021). From virtual to physical problem solving in coding: A comparison on various multimodal coding tools for children using the framework of problem solving. *Research anthology on recent trends, tools, and implications of computer programming* (pp. 677–694). IGI Global. https://doi.org/10.4018/978-1-7998-3016-0.ch030

Nardie Fanchamps I work at the Open University The Netherlands as an Assistant Professor at the faculty for Educational Sciences within the Technology Enhanced Learning Innovations department. I am specialised in educational robotics, computational thinking, programming in education, augmented reality and VR. I am involved in several European / Global research projects where I am pleased to collaborate with colleagues from other Universities and partner organisations. I have a Master's degree in Mathematics and Educational Sciences. The subject of my Ph.D.-dissertation was "Sense-Reason-Act (SRA) Programming and the Impact on Computational Thinking". I have gained a lot of experience as a team leader, advisor, senior lecturer, programme leader, project leader, coach, coordinator, manager, tutor and implementer within our faculty and ZD branch with an involvement in many other challenging and exciting (research) projects. My expertise and interests also include STEM-learning, highly visualised learning environments and Interprofessional Collaboration (IPS).