An exploration of the role of visual programming tools in the development of young children's computational thinking

ROSE, Simon <http://orcid.org/0000-0002-8165-3016>, HABGOOD, Jacob <http://orcid.org/0000-0003-4531-0507> and JAY, Tim <http://orcid.org/0000-0003-4759-9543>

An Exploration of the Role of Visual Programming Tools in the Development of Young Children's Computational Thinking

Simon P. Rose, M. P. Jacob Habgood and Tim Jay
Sheffield Hallam University, Sheffield, UK
simon.rose@shu.ac.uk
j.habgood@shu.ac.uk
t.jay@shu.ac.uk

Abstract: Programming tools are being used in education to teach computer science to children as young as 5 years old. This research aims to explore young children's approaches to programming in two tools with contrasting programming interfaces, ScratchJr and Lightbot, and considers the impact of programming approaches on developing computational thinking. A study was conducted using two versions of a Lightbot-style game, either using a ScratchJr-like or Lightbot style programming interface. A test of non-verbal reasoning was used to perform a matched assignment of 40, 6 and 7-year-olds to the two conditions. Each child then played their version of the game for 30 minutes. The results showed that both groups had similar overall performance, but as expected, the children using the ScratchJr-like interface performed more program manipulation or 'tinkering'. The most interesting finding was that non-verbal reasoning was a predictor of program manipulation, but only for the ScratchJr-like condition. Children approached the ScratchJr-like program differently depending on prior ability. More research is required to establish how children use programming tools and how these approaches influence computational thinking.

Keywords: Visual programming, Education, Computational thinking, K-12, Lightbot, Scratch

1. Introduction

This paper focuses on two tools used to teach programming to young children, ScratchJr (Flannery *et al.*, 2013) and Lightbot (Lightbot Inc., 2016). Existing literature suggests that both these tools encourage computational thinking, yet there are clear theoretical contrasts in the type of programming interfaces that they use. We describe an exploratory study to investigate whether there was a difference in the way that young children use these tools and consider its relationship to developing computational thinking skills.

We live in a digital age where technology plays a key role in almost everything we do, making it increasingly important for us to understand how it works. Today's children will go on to live a life dominated by computing, both in the home and at work (Barr and Stephenson, 2011). Computing education is receiving increasing attention in classrooms worldwide, with the aim of developing digital, media and information literacies. The need for children to be effective users of computational tools has led to the re-examination of the concept of 'computational thinking'. Although the term was originally used by Papert (1980), Wing (2006) describes it as the problem-solving processes used by computer scientists. She stated that it should be taught as a basic skill across the school curriculum (2008). Since Wing reintroduced the concept of computational thinking; many researchers have attempted to clarify what it is and how we can teach it (e.g. Grover and Pea, 2013; Yadav, Hong and Stephenson, 2016).

Programming tools are seen as a means of developing computational thinking skills (e.g. Wilson and Moffat, 2010; Brennan and Resnick, 2012; Berland and Wilensky, 2015). This has led to the release of a variety of new tools, such as ScratchJr, Hopscotch and Kodable. Scratch remains the most widely-used of children's programming tools. It takes inspiration from constructionism and the LOGO programming language (Papert, 1980). Constructionism is a pedagogical theory based on constructivism (Piaget, 1970), which makes specific use of the construction of artefacts as a basis for building knowledge. Papert theorised that by thinking about programming, learners would learn about the process of thinking, and he believed these skills would transfer to other contexts (1980). Scratch provides a constructionist learning environment through block-based programming, where learners combine instruction blocks to form programs (Resnick *et al.*, 2009). Researchers have identified differing approaches when children program in Scratch (Meerbaum-Salant, Armoni and Ben-Ari, 2011).

Reference this paper as: Rose S P., Habgood J M. P. and Jay T, "An Exploration of the Role of Visual Programming Tools in the Development of Young Children's Computational Thinking" *The Electronic Journal of e-Learning Volume 15 Issue 4 2017*, (pp297-309) available online at www.ejel.org

Several countries have now introduced computer science into national curricula (Heintz, Mannila and Farnqvist, 2016), meaning that children as young as 5 years old are now learning basic programming skills. Whilst there is evidence to suggest that children can learn to program at this age (Bers, 2010; Fessakis, Gouli and Mavroudi, 2013), there is comparatively little empirical research on children's use of programming tools under the age of 7. This is particularly important due to the cognitive developments that children undergo around this age (Manches and Plowman, 2015). Some researchers are concerned that younger children struggle to understand fundamental computer science concepts like abstraction (e.g. Armoni, 2012).

2. Computational thinking

Seymour Papert first described computational thinking as part of his research into how children develop procedural thinking through computer programming (1980). Wing sparked a renewed interest in the topic (2006), suggesting that "to reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability" (p. 33). Furthermore, Wing suggested that teaching computational thinking enables children to learn to think in an abstract and algorithmic manner relevant to many disciplines, including mathematics and science. She went on to define computational thinking as "solving problems, designing systems, and understanding human behaviour, by drawing on the concepts fundamental to computer science" (Wing, 2006, p. 33), but ten years later there is still no unanimous agreement on a definition (Garcia-Peñalvo, 2016; Weintrop *et al.*, 2016).

There have been many efforts to clarify what is involved in computational thinking (e.g. Barr and Stephenson, 2011; Grover and Pea, 2013; Kalelioglu, Gulbahar and Kukul, 2016). There is a general agreement that it includes all the concepts that a computer scientist would typically use to solve computational problems (Riley and Hunt, 2014), but the list of concepts is up for debate. Table 1 shows the different concepts used in 7 existing definitions of computational thinking.

Table 1: The concepts included in existing definitions of computational thinking

Barr and Stephenson (2011)	Brennan and Resnick (2012)	Grover and Pea (2013)	Seiter and Foreman (2013)	Kalelioglu, Gulbahar and Kukul (2016)	Angeli et al. (2016)	Repenning, Basawapatna and Escherle (2016)
Abstraction	Abstracting and modularising	Abstraction and pattern generalisation	Abstraction	Abstraction	Abstraction	Abstraction
Algorithms and procedures	Sequences	Algorithmic notions of flow of control	Procedures and algorithms	Algorithms and procedures	Algorithms (including sequencing and flow of control)	
Data collection, analysis and representation	Data	Symbol systems and representations	Data Representation	Data collection, analysis and representation		
Problem decomposition		Structured problem decomposition	Decomposition	Decomposition	Decomposition	
Parallelisation	Parallelism	Iterative, recursive and parallel thinking	Parallelisation and synchronisation	Parallelisation		
Testing and verification	Testing and debugging	Debugging and systematic error detection		Testing and debugging		Analysis
Control structures	Conditionals and loops	Conditional logic		Mathematical reasoning		
Automation				Automation		Automation
				Generalisation	Generalisation	
Simulation				Modelling and simulations		
	Events					
		Efficiency and performance constraints				
		Systematic processing				
				Conceptualising		

For the purposes of this work, we have defined a working definition for computational thinking using the 7 most common concepts included in the definitions above:

- Abstraction and generalisation (removing the detail from a problem and formulating solutions in generic terms)
- Algorithms and procedures (using sequences of steps and rules to solve a problem)
- Data collection, analysis and representation (using and analysing data to help solve a problem)
- Decomposition (breaking a problem down into parts)
- Parallelism (having more than one thing happening at once)
- Debugging, testing and analysis (identifying, removing and fixing errors)
- Control structures (using conditional statements and loops)

This process helped to identify individual concepts and provided a deeper understanding of computational thinking. This is the definition of computational thinking used in the rest of the work and will be used to evaluate two programming tools for their potential to develop computational thinking skills.

3. Programming tools for young children

In the previous section, we defined a working set of computational thinking concepts. This section will analyse two programming tools designed for young children and evaluate their potential to develop computational thinking with respect to this set of concepts.

3.1 ScratchJr

Scratch is a block-based programming tool designed for children aged 8-16. It aims to "support self-directed learning through tinkering and collaboration" (Maloney, Resnick and Rusk, 2010, p. 2) and requires the application of computational thinking concepts (Resnick *et al.*, 2009).

ScratchJr is a version of Scratch redesigned for younger children aged 5-7 (figure 1). It maintains the creative programming elements of Scratch, which allow children to easily create short stories and games. Characters can be added to a scene, and are given behaviours by combining instruction blocks. The interface is entirely symbolic and contains only a third of the original Scratch instruction set because young children can struggle with several levels of decomposition (Flannery *et al.*, 2013). ScratchJr also executes instructions from left to right (the way that the English language is read) instead of the top to bottom approach used in Scratch. It has large buttons for touchscreen use, which apparently compounds difficulties that young children often have with mouse movement. The Cartesian coordinate system used in Scratch has been replaced by a natural coordinate system, and there is a grid that can be overlaid on top of the scene to help children calculate distance. Numerical parameter values have a maximum limit of 25, and children can execute individual instructions simply by pressing on them to help them explore what each instruction does. ScratchJr was developed using several age-appropriate design principles (Flannery *et al.*, 2013). It makes it easy to get started but provides room to use more complex concepts (low floor and high ceiling), it allows many pathways and styles of exploration (wide walls), ideas can be incrementally developed through experimentation (tinkerability), the interface is friendly and playful (conviviality) and it can be used with a wide range of learning outcomes (classroom support).

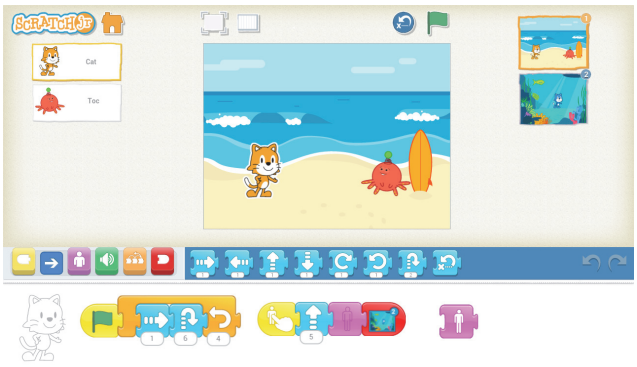

Figure 1: A scene from ScratchJr

3.2 Lightbot

Lightbot is an educational puzzle game. The player must arrange a fixed set of block-based instructions in a finite program space that tell a robot what to do (figure 2). The goal is to program the robot to turn all the blue blocks in a level into illuminated yellow blocks. This is done by navigating the robot to a blue block and executing the light command. Players can decompose a level into different sections, which can then be solved one after the other until they have a complete solution. Some of the later levels can only be completed through the correct use of procedures and conditionals. For procedures, the player is given other program spaces below the main program that can be called using special instructions. Conditionals are implemented using a paint tool that colours the robot so that only instructions of that colour are executed. Gouws, Bradshaw and Wentworth (2013) suggest that Lightbot is useful for practising computational thinking. It concentrates on using computational thinking as a problem-solving process, and players are rewarded for producing optimised solutions.

3.2.1 Non-verbal reasoning

Successful Lightbot players can use mental transformations to predict the movement of the robot, recognise patterns from other levels and implement these patterns using known sequences of instructions (Gouws, Bradshaw and Wentworth, 2013). This is comparable to non-verbal reasoning, which is the ability to analyse information and solve problems using visual information. Non-verbal reasoning contains both abstract (or diagrammatic) and spatial reasoning, which includes spatial transformations, recognising visual sequences, and identifying relationships between shapes and patterns. Non-verbal reasoning is not reliant upon or limited by language ability, and research suggests that it can indicate mathematical ability in children (Halberda, Mazzocco and Feigenson, 2008).

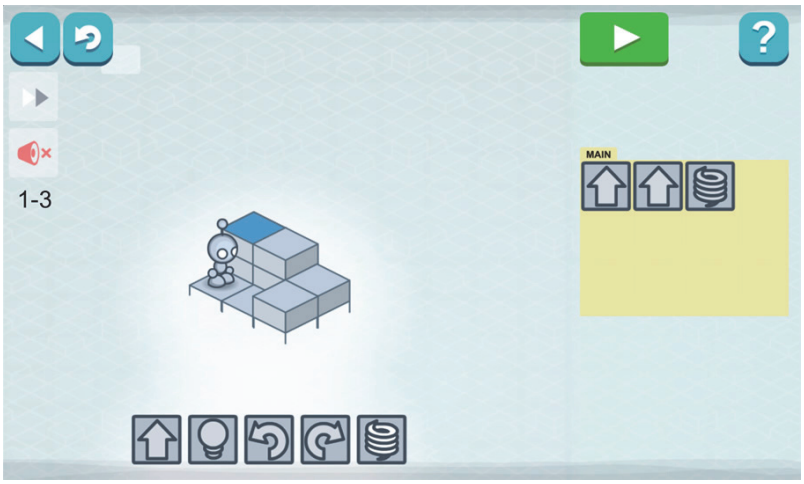

Figure 2: A simple level from Lightbot

3.3 Comparison of the tools

These tools were analysed for their support of computational thinking using the definition in the previous section (table 2). From this, it is reasonable to conclude that both tools encourage computational thinking. They both use almost all the common computational thinking concepts identified in section 2. The only difference is that Lightbot doesn't support parallelism.

Table 2: The computational thinking concepts used in ScratchJr and Lightbot

	ScratchJr	Lightbot
Abstraction and generalisation	• Understanding of the grid and character movement • Identifying common behaviours using instructions and instruction blocks	• Understanding of the grid and robot movement • Identifying common solutions to levels
Algorithms and procedures	• Sequencing instructions to create algorithms • Using procedures to repeat common instructions	• Sequencing instructions to create algorithms • Using procedures to repeat common instructions
Data collection, analysis and representation	• Counting movement needed using the grid	• Counting movement needed using the level grid
Decomposition	• Applying behaviours to different characters • Having multiple instruction blocks in one character • Applying behaviours in steps	• Breaking down and solving levels in parts
Parallelism	• Blocks of instructions are executed in parallel	
Debugging, testing and analysis	• The instruction currently being executed is highlighted • Programs can be re-run to check for errors • Instructions can be pressed individually to test what they do	• The instruction currently being executed is highlighted • Programs can be re-run to check for errors
Control structures	• Using blocks such as repeat and wait to control execution • Looping instructions using procedures	• Using conditionals in later levels • Looping instructions using procedures

Despite their similarities, there is a specific operational difference between the visual programming paradigms employed in ScratchJr and Lightbot. In ScratchJr, a limitless number of blocks can be added to the program space, these blocks are not executed unless they are linked to a trigger block or individually pressed to execute them. Whereas in Lightbot, the play button sequentially executes all the instructions included in the main program. Lightbot also limits how many instructions can be in the program depending on the current level. It is this operational difference which led us to explore how young children used these tools and whether they encouraged a fundamentally different programming approach.

4. Programming approaches

Turkle and Papert described two approaches to problem-solving. The first was an analytical top-down approach where solutions to problems are planned. The second was a bottom-up or "bricolage" approach, where solutions are attempted "by arranging and rearranging, by negotiating and renegotiating with a set of well-known materials" (1991, p. 136). In constructivist learning theory, a child builds knowledge through experience. The information they receive through interactions challenges their world view (Piaget, 1970). Constructionism applies this theory to the construction of artefacts (Papert, 1980). It is a pedagogical theory which suggests that learners should be given the opportunity to experiment and explore ideas by tinkering with an artefact. Learners are guided "by the work as it proceeds rather than staying with a pre-established plan" (Papert and Harel, 1991, p. 6), leading to self-directed learning. Scratch is based on these principles (Resnick *et al.*, 2009).

Research has shown that children aged 10-15 can learn computer science using Scratch (Meerbaum-Salant, Armoni and Ben-Ari, 2013; Sáez López, González and Cano, 2016). Despite this, there are some suggestions that Scratch may encourage unusual programming approaches (Meerbaum-Salant, Armoni and Ben-Ari, 2011). A top-down approach is traditionally taught in programming, where software is decomposed into coherent units that can be better maintained. Meerbaum-Salant, Armoni and Ben-Ari observed that 14 and 15-year-olds took the top-down approach to the extreme. They decomposed programs into many small blocks of instructions (sometimes hundreds) that lacked logical coherency. This can make programs particularly difficult to debug in Scratch and ScratchJr due to the way they both execute all instruction blocks in parallel. Children in the study by Meerbaum-Salant, Armoni and Ben-Ari (2011) became frustrated and lost motivation because

they did not understand what was happening in their programs. They also observed that Scratch programs were often developed using a bottom-up approach. In bottom-up programming, components are designed in isolation then linked together to form a complete solution. This can be an appropriate method of software design, but children once again took it to the extreme. When faced with a problem, they would attempt to solve it by "dragging all the blocks that seemed to be appropriate for solving the task, and then combining them into a script" (2011, p. 169). This tinkering behaviour is encouraged in Scratch and ScratchJr by the fact that instructions can be left in the script area without affecting the execution of the program.

We have identified two approaches to programming; top-down and bottom-up. Along with indications that both are used by children in Scratch. In contrast, Lightbot provides a programming interface which doesn't allow much tinkering. Instructions can be freely added to and deleted from the main program, but when an instruction is visible, it is always part of the program and executed in strict sequence. It was this central difference that provided the basis for this study, exploring the affect that the two programming paradigms had on children's approaches to programming. Lightbot contains only a subset of the commands available in ScratchJr, so it was decided that the programming tasks used for the study should be based on navigating robots (as in Lightbot), as such tasks could be easily undertaken in both programming environments.

5. Method

5.1 Aims and hypotheses

This was an explorative study examining young children's approaches to programming using the two different programming paradigms. Although some hypotheses were formed, the study was primarily undertaken to identify questions that could become the focus of future research.

Three hypotheses were initially formed based on the existing literature:

a) A ScratchJr-like programming interface would lead to more "tinkering" than a Lightbot interface.
b) A ScratchJr-like programming interface would lead to improved outcomes on problem-solving tasks.
c) Higher-ability players would benefit more from a ScratchJr-like programming interface.

5.2 Participants

The participants were from a large primary school in a low-income area in northern England. Most pupils at the school are of White British heritage. The school has a well above average proportion of disadvantaged pupils and pupils that require support for special educational need. The study participants included 20 boys and 20 girls between the age of 6 years, 3 months and 7 years, 3 months (*M* = 6 years, 9 months).

5.3 Materials and procedure

A non-verbal reasoning test was created for this study to produce matching pairs, based on the assumption that non-verbal reasoning is required in Lightbot (section 3.2.1). Standardised school worksheets (Primary Leap Limited, 2011) were used as a model for the questions. There were three types of questions; matching shapes (as seen in figure 3), selecting the odd one out from a series of shapes, and selecting the next shape or missing shape in a pattern. The possible answers to some questions were rotated, requiring the participant to perform mental transformations, similar to the rotation process required in Lightbot.

The test took place in the school IT suite in groups of 15. It was 40 questions long, and the participants had 5 minutes to answer as many as they could. Participants were told there was no rush to answer the questions, and that their answers should be carefully thought through. The time-limit and number of questions aimed to produce a greater range of test scores, reducing the possibility of ceiling effects.

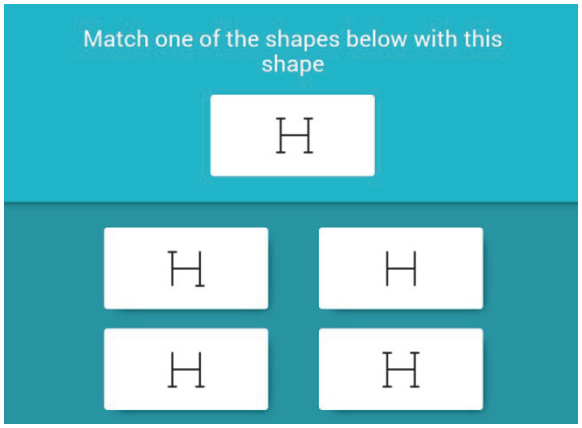

Figure 3: An example question from the non-verbal reasoning test

Two versions of a Lightbot style game for 6 and 7-year-olds were created for this study. One that used the Lightbot programming interface, and one that used a ScratchJr-like interface (see figure 4). The versions were identical apart from that in the ScratchJr-like version, instructions can be added to the program that will not execute unless linked to the trigger block (see table 3). The game has 15 levels; it begins with simple levels that require only forward and light instructions. The later levels then introduce more complex movements and levels with several lights. The difficulty progression was designed so that it could challenge more able children in the target age group.

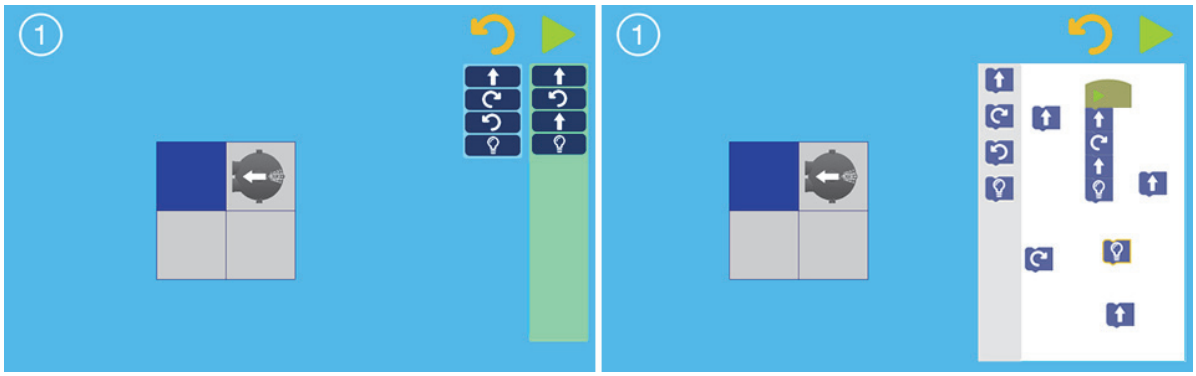

Figure 4: The Lightbot (left) and ScratchJr-like version (right) of the game

Table 3: The similarities and differences between the two versions of the game

	Lightbot	ScratchJr-like
Similarities	• A fixed set of instructions (forward, 90° rotation clockwise, 90° rotation anti-clockwise, light). • Instructions can each be used more than once. • Instructions can be added, rearranged and removed from the program space.	
Differences	• All instructions in the program will be performed by the robot.	• Only the instructions linked to the trigger block will be performed by the robot.

Two groups of 20 participants were created using the non-verbal reasoning scores as a matching variable (based on the assumption that non-verbal reasoning ability was required to be successful in the game). Each child then played one version of the game for thirty minutes in a small reading room joined to the children's classroom. Two laptops were set up facing away from each other so that one child from each group could play the game without being aware that they were using a different version to their classmate. Testing the conditions together meant that any extraneous variables (e.g. time of day) would affect both groups equally. All participants were given a uniform introduction to the game via a tutorial video.

5.4 Measures

A range of measures were used to explore how the participants used each version of the game:

1. The non-verbal reasoning scores for each participant.
2. Program manipulation; additions, moves and deletions of instructions per attempt.
3. The number of attempts needed by a participant to complete a level.

4. The highest level reached by each participant.
5. The time taken by a participant per attempt.
6. The time taken by a participant to complete a level.

As explained in section 5.3, the scores of a non-verbal reasoning test were used as a matching variable to create two even ability groups. It is, therefore, expected that these scores should predict how well a participant performed in the game and this would be demonstrated by a correlation between the participant test scores and the highest level they reached.

Program manipulation was measured by the number of additions, moves and deletions of instructions from the program space between each attempt. An attempt was defined as each time a participant ran their program by pressing the play button. It was hoped that this measure, and the amount of time between each attempt, would provide some indication of how participants are interacting with the game and could be used to infer something about their programming approach (particularly in terms of the amount of 'tinkering' taking place).

Overall performance was measured using the highest level reached by each participant. This can be combined with program manipulation to explore the effect of the two conditions on overall performance. Data was also collected on other performance measures, such as how many attempts it took for participants to complete a level and how much time it took them to do so. These measures can be used to show if the two games were as similar as expected.

6. Results

6.1 Comparing the difficulty of the two versions

The programming interface should have been the only difference between the two versions of the game. Independent T-tests were performed on the overall performance measures between groups (see table 4). On average, Lightbot players reached a slightly higher level in 30 minutes than the ScratchJr-like players, but this difference was not significant, $t(38) = .54$, $p = .59$. ScratchJr-like players spent slightly more time (in seconds) on each level than the Lightbot players, again the difference was not significant, $t(38) = -1.12$, $p = .27$. Finally, ScratchJr-like players took fewer attempts on average to complete levels than Lightbot players. This difference was not significant, $t(38) = .98$, $p = .33$.

Table 4: T-test detail for the overall performance measures between groups

Measure	Lightbot (N = 20)			ScratchJr-like (N = 20)		
	M	SD	SE	M	SD	SE
Highest level reached	10.25	4.25	.95	9.5	4.48	1
Average time taken to complete a level (seconds)	187	86.1	19.25	224.35	121.33	27.13
Average attempts needed to complete a level	9.4	6.86	1.53	7.54	5.02	1.12

6.2 Non-verbal reasoning as a predictor of game performance

The non-verbal reasoning test was designed to produce a range of scores, containing 40 questions and using a 5-minute time-limit. The median of the collected scores was 23.5 with a minimum of 15 and a maximum of 35. The middle 50% of scores were between 20.25 and 30.75. The groups were created based on the assumption that the scores would indicate how well participants would perform in the game. This was supported by a strong correlation between the scores and the highest level reached by each participant, $r(40) = .73$, $p < 0.001$. The game was designed to challenge the more able participants, but not be too difficult for the lower-ability participants. The median highest level that participants reached was 8.5, with a minimum of 4 and a maximum of 15 (the last level). The middle 50% fell between 6 and 15, with 13 participants reaching or completing the last level.

6.3 Comparing program interaction between groups

Participant's program manipulation was compared to see if there was any difference in how they were interacting with the game. A one-way ANCOVA was used to compare the average instruction additions, moves and deletions per attempt from the program. The non-verbal reasoning scores were used as a covariate because participant ability may have influenced the amount of program manipulation they performed. There was a significant difference between the conditions, $F(1,37) = 192.19$, $p < .001$. Participants in the ScratchJr-

like condition manipulated the program more (M = 9.06, SD = 3.27, SE = .73) than the participants in the Lightbot condition (M = 4.68, SD = 1.9, SE = .42). A similar one-way ANCOVA was used to test for differences in the time (in seconds) taken to formulate an attempt. This also showed a significant difference between conditions, $F(1,37) = 9.58$, $p = .004$. The participants in the ScratchJr-like condition took longer on average (M = 34.15, SD = 11.38) to construct their programs than the Lightbot group (M = 24.29, SD = 8.87).

6.4 Using non-verbal reasoning to predict program interaction

We then tested if non-verbal reasoning scores would indicate how much a participant was manipulating their program. A single linear regression was used to predict the average program manipulation per attempt based on the test score. The results of the regression show that overall the scores significantly predicted program manipulation, $\beta = .37$, $t(37) = 3.32$, $p = .002$. Participants with higher non-verbal reasoning scores performed more manipulation. The tests scores also explained a significant proportion of variance in program manipulation, R^2 of .55, $F(2,37) = 22.46$, $p < .001$. An interaction effect was added to test if there was a difference between groups. This showed that in the ScratchJr-like version, there was a bigger effect of the condition when participants had higher scores of non-verbal reasoning, $\beta = .35$, $t(36) = -2.35$, $p = .025$. This result was supported by a correlation between non-verbal reasoning and program manipulation in the ScratchJr-like condition, $r(20) = .65$, $p = .002$, but not in the Lightbot condition, $r(20) = .22$, $p = .342$ (figure 5).

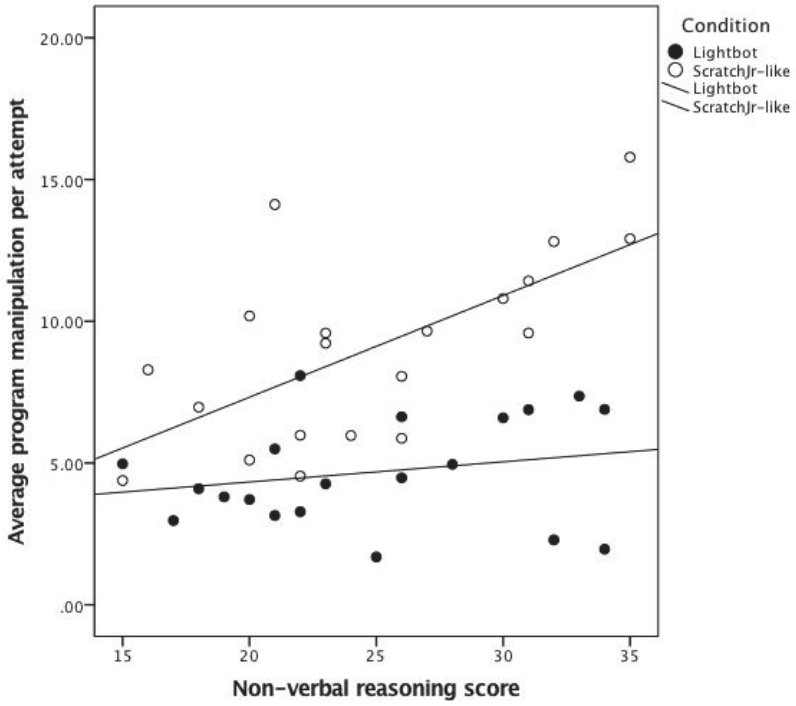

Figure 5: The correlations between non-verbal reasoning and average program manipulation per attempt for each condition

6.5 Using the highest level reached to predict program interaction

A similar analysis was then conducted using the highest level reached by each participant as an indicator of in-game success, instead of their non-verbal reasoning scores. Participants were divided according to whether they had completed level 8 or not, as this represented a median split. Using the non-verbal reasoning scores as a covariate, a two-way ANCOVA was conducted that examined the effect of the interface-type, and whether a participant completed level 8, on program manipulation. The results showed a significant interaction between the interface-type and program manipulation, $F(1,35) = 45$, $p < .001$, and a significant interaction between the interface-type and whether the participant completed level 8, $F(1,35) = 10.16$, $p = .003$. Completing level 8 alone was not a predictor of program manipulation, $F(1,35) = 1.77$, $p = .192$. The average program manipulation per attempt for both groups is shown in table 5. This is further supported by the graph in figure 6, which shows the average amount of program manipulation per attempt for each level.

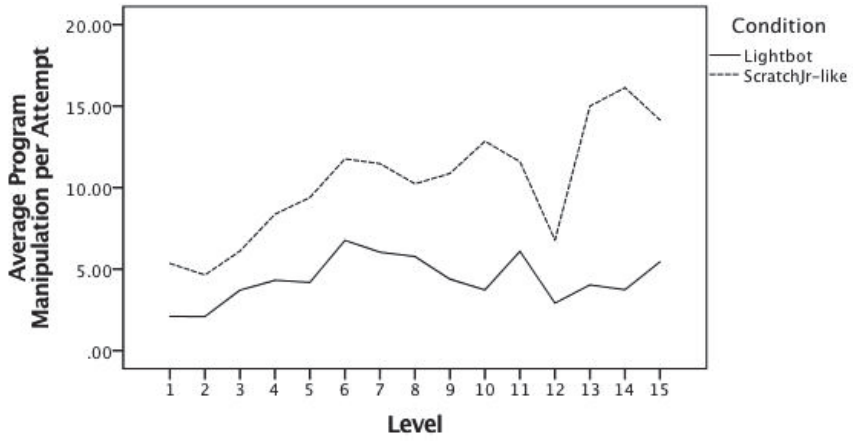

Figure 6: The average amount of program manipulation used per attempt in each level

Table 5: The average program manipulation per attempt dependant on whether a participant completed level 8.

Level Completed	Lightbot (*N* = 20)				ScratchJr-like (*N* = 20)			
	N	M	SD	SE	N	M	SD	SE
< 8	9	4.47	1.6	.53	11	7.06	2.01	.6
>= 8	11	4.84	2.17	.65	9	11.51	2.86	.95

7. Discussion

The results showed that overall performance was similar in both versions of the game. Participants reached a similar level, spent a similar amount of time on each level and took a similar number of attempts to complete each level. This is not surprising given that the programming interface was the only difference between the two versions. We can also say that the game provided a suitable level of challenge, as all participants completed at least the third level of the game and around a third of participants reached or completed the last level.

The non-verbal reasoning test produced a good range of scores (between 15 and 35), suggesting that the difficulty of the questions was appropriate for the age group. The correlation between these scores and the highest level reached shows that non-verbal reasoning was a strong indicator of success in the game, and justifies the use of these scores as a matching variable to create two even ability groups. Furthermore, if both Lightbot and ScratchJr encourage computational thinking as has been suggested (Flannery *et al.*, 2013; Gouws, Bradshaw and Wentworth, 2013) then this correlation could also support the idea that non-verbal reasoning and computational thinking are linked.

There was a pronounced difference in the amount of program manipulation between groups as measured by additions, moves and deletions of instructions. Participants in the ScratchJr-like condition performed 1.9 times more manipulation per attempt than the participants in the Lightbot condition. They also took 1.4 times longer on average to formulate each attempt. These findings are in line with the constructionist principles of Scratch's design and consistent with the idea that children using the ScratchJr-like interface are being guided "by the work as it proceeds rather than staying with a pre-established plan" (Papert and Harel, 1991, p. 6). The increased tinkering in the ScratchJr-like condition could also suggest a bottom-up, or bricolage, approach to programming.

The role of prior ability in the level of program manipulation was a particularly interesting finding of this study. Non-verbal reasoning test scores were used as an indicator of participant ability and our analysis showed that these were a strong predictor of program manipulation overall. The higher their score of non-verbal reasoning, the more program manipulation a participant performed, but interestingly the effect was only significant for the ScratchJr-like condition. This indicates a contrast between how lower and higher ability players approached tasks in the ScratchJr-like condition. High-ability players performed more tinkering than their low-ability counterparts, suggesting that they were more suited to the free-design approach of ScratchJr-like

instructions. On the other hand, Lightbot players all manipulated their programs roughly the same amount. This is an interesting finding given the similar overall performance of participants in both groups.

This finding was mirrored by using the highest level reached as an indicator of participant ability instead of non-verbal reasoning scores. Around half the participants progressed beyond level 8, so this point was used to group participants according to their success within the game. The analysis showed that there was a significant difference in manipulation within the ScratchJr-like group, with the more successful participants performing more manipulation on average throughout the game. Whereas the Lightbot participants altered their programs a similar amount per attempt no matter how successful they were within the game. This is further supported by the level-by-level data, which shows an increase in manipulation in the ScratchJr-like condition above level 8 (figure 6).

ScratchJr was designed with a low floor and (appropriately) high ceiling (Flannery *et al.*, 2013). Yet we have found indications that the ability (measured using non-verbal reasoning and game performance) of a child can affect how they interact with a ScratchJr-like programming interface. More-able children tinkered 1.6 times more with their ScratchJr-like programs than less-able children using the ScratchJr interface, while children using the Lightbot interface used the same amount of manipulation throughout the game. Of course, there are many possible explanations for this. Lightbot users may have used less manipulation because programs were faster to create, making a trial and error approach efficient, even as the levels got more difficult. The apparent consistency in strategy could also suggest that the Lightbot interface naturally helped players to decompose levels into smaller sections where instructions could be added incrementally to their programs. Nonetheless, the finding that higher-ability ScratchJr players performed more tinkering is very intriguing and it would be natural to consider whether their approach allowed them to develop a deeper understanding of the game's abstract concepts and design solutions that more accurately predicted what the robot needed to do. It may suggest that lower ability children need more support when using block-based programming tools like ScratchJr. Their lack of tinkering may be because of underdeveloped working memory and the cognitive load of the task (Sweller, 1988), leading us to question how low the floor should be in low floor and high ceiling design for young children. It also poses several possible questions about how programming tools are used in education; do less-able children get the support they require using these tools to meet learning outcomes? Is this influenced by the teacher's knowledge of the programming tool? And can cognitive load be reduced by teaching the skills required by these tools individually?

8. Limitations

This study was intended to be exploratory in nature, and clearly, the design has limitations which should be acknowledged. Post-hoc analyses are appropriate to exploratory work, and useful for generating new hypotheses, but limit the validity of the findings. Although a matched design was used, the matching variable would normally have been the same as the dependent variable of the study, rather than a separate measure. Future studies will use a pre-to post-test design based on a common measure of computational thinking to address this, but developing an instrument which reliably measures computational thinking is a non-trivial task (Jenson and Droumeva, 2016).

The software itself had some limitations which could have affected the outcome of the study. It was observed that many participants completed levels using non-optimal solutions. This included over compensating for turns and having to rotate back the other way or having instructions in a direction where the robot could not go (shown by the robot 'shaking its head' when the block was executed). Arguably the Lightbot version of the experimental software should have had finite program space per level to contrast with the bottom-up approach employed by ScratchJr. This would have required participants to produce optimal solutions consistent with the original Lightbot game. This could potentially have increased the difference between versions as ScratchJr programming doesn't have this restriction.

9. Conclusion

Using an exploratory approach, we aimed to establish how children used two different programming tools. We found indications that children aged six and seven interacted differently with the ScratchJr-like programming interface compared to the Lightbot interface. Children using the ScratchJr-like interface performed more program manipulation, which could indicate a more bottom-up programming approach. These findings are in line with the constructionist foundations of Scratch and ScratchJr. A more surprising finding was that more

able children (as measured by non-verbal reasoning skill or game performance), manipulated their programs more. This difference was only found in players using the ScratchJr-like interface, whereas ability had no effect on the program manipulation of Lightbot players.

This exploratory work offers some potential explanations for these findings, but more research is clearly required to establish how young children use programming tools and how they influence their development of problem-solving and computational thinking abilities.

This paper also examined the existing definitions (and expectations) of computational thinking, and we would share the concerns of other authors that these may be too broad (Weintrop *et al.*, 2016). We propose that future research in this area focuses on the individual concepts involved in computational thinking. Investigating whether programming tools can be used to develop concepts such as decomposition, abstraction and algorithmic thinking. This would seem particularly important given concerns that young children may struggle to understand these concepts (Armoni, 2012; Manches and Plowman, 2015) despite the pressure on schools to teach them.

References

Angeli, C., Voogt, J., Fluck, A., Webb, M., Cox, M., Malyn-Smith, J. and Zagami, J. (2016) 'A K-6 Computational Thinking Curriculum Framework: Implications for Teacher Knowledge', *Educational Technology & Society*, 19(3), pp. 47–57.

Armoni, M. (2012) 'Teaching CS in Kindergarten: How Early Can the Pipeline Begin?', *ACM Inroads*, 3(4), pp. 18–19. doi: 10.1145/2381083.2381091.

Barr, V. and Stephenson, C. (2011) 'Bringing Computational Thinking to K-12: What is Involved and What is the Role of the Computer Science Education Community?', *ACM Inroads*, 2(1), pp. 48–54. doi: 10.1145/1929887.1929905.

Berland, M. and Wilensky, U. (2015) 'Comparing Virtual and Physical Robotics Environments for Supporting Complex Systems and Computational Thinking', *Journal of Science Education and Technology*. Springer Netherlands, 24(5), pp. 628–647. doi: 10.1007/s10956-015-9552-x.

Bers, M. U. (2010) 'The TangibleK robotics program: Applied computational thinking for young children', *Early Childhood Research and Practice*, 12(2), pp. 1–20.

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

Fessakis, G., Gouli, E. and Mavroudi, E. (2013) 'Problem solving by 5-6 years old kindergarten children in a computer programming environment: A case study', *Computers and Education*. Elsevier Ltd, 63, pp. 87–97. doi: 10.1016/j.compedu.2012.11.016.

Flannery, L. P., Kazakoff, E. R., Bontá, P., Silverman, B., Bers, M. U., Resnick, M., Kazakoff, E. R., Bers, M. U., Bontá, P. and Resnick, M. (2013) 'Designing ScratchJr: Support for Early Childhood Learning Through Computer Programming', in *Proceedings of the 12th International Conference on Interaction Design and Children (IDC '13)*, pp. 1–10. doi: 10.1145/2485760.2485785.

Garcia-Peñalvo, F. J. (2016) 'What Computational Thinking Is', *Journal of Information Technology Research*, 9(3), pp. 5–8.

Gouws, L., Bradshaw, K. and Wentworth, P. P. (2013) 'Computational Thinking in Educational Activities An evaluation of the educational game Light-Bot', in *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*.

Grover, S. and Pea, R. (2013) 'Computational Thinking in K-12: A Review of the State of the Field', *Educational Researcher*, 42(1), pp. 38–43. doi: 10.3102/0013189X12463051.

Halberda, J., Mazzocco, M. M. M. and Feigenson, L. (2008) 'Individual differences in non-verbal number acuity correlate with maths achievement.', *Nature*, 455(October), pp. 665–668. doi: 10.1038/nature07246.

Heintz, F., Mannila, L. and Farnqvist, T. (2016) 'A Review of Models for Introducing Computational Thinking, Computer Science and Computing in K–12 Education', in *2016 IEEE Frontiers in Education Conference (FIE)*, pp. 1–9. doi: 10.1109/FIE.2016.7757410.

Jenson, J. and Droumeva, M. (2016) 'Exploring media literacy and computational thinking: A game maker curriculum study', *Electronic Journal of e-Learning*, 14(2), pp. 111–121.

Kalelioglu, F., Gulbahar, Y. and Kukul, V. (2016) 'A Framework for Computational Thinking Based on a Systematic Research Review', *Baltic Journal of Modern Computing*, 4(3), pp. 583–596.

Lightbot Inc. (2016) *Lightbot*. Available at: https://lightbot.com/.

Maloney, J., Resnick, M. and Rusk, N. (2010) 'The Scratch programming language and environment', *ACM Transactions on Computing Education*, 10(4), pp. 1–15. doi: 10.1145/1868358.1868363.http.

Manches, A. and Plowman, L. (2015) 'Computing education in children's early years: A call for debate', *British Journal of Educational Technology*, 48(1), pp. 191–201. doi: 10.1111/bjet.12355.

Meerbaum-Salant, O., Armoni, M. and Ben-Ari, M. (2011) 'Habits of Programming in Scratch', in *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*, pp. 168–172. doi: 10.1145/1999747.1999796.

Meerbaum-Salant, O., Armoni, M. and Ben-Ari, M. (2013) 'Learning computer science concepts with Scratch', *Computer Science Education*, 233, pp. 239–264. doi: 10.1080/08993408.2013.832022.

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

Papert, S. and Harel, I. (1991) *Situating constructionism*, *Constructionism*. Ablex Publishing Corporation.

Piaget, J. (1970) *Science of education and the psychology of the child*. New York: Orion Press.

Primary Leap Limited (2011) *Non-Verbal Reasoning Worksheets*. Available at: http://primaryleap.co.uk/primary-resources/Year+1/Reasoning/Non+-+Verbal/ (Accessed: 1 November 2015).

Repenning, A., Basawapatna, A. R. and Escherle, N. (2016) 'Computational thinking tools', in *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 1–5.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J. a Y., Silverman, B. and Kafai, Y. (2009) 'Scratch: Programming for All.', *Communications of the ACM*, 52(11), pp. 60–67. doi: 10.1145/1592761.1592779.

Riley, D. D. and Hunt, K. A. (2014) *Computational Thinking for the Modern Problem Solver*. CRC Press.

Sáez López, J. M., González, M. R. and Cano, E. V. (2016) 'Visual programming languages integrated across the curriculum in elementary school: A two year case study using "scratch" in five schools', *Computers & Education*, 97, pp. 129–141. doi: 10.1016/j.compedu.2016.03.003.

Seiter, L. and Foreman, B. (2013) 'Modeling the learning progressions of computational thinking of primary grade students', in *Proceedings of the ninth annual international ACM conference on International computing education research - ICER '13*, pp. 59–66. doi: 10.1145/2493394.2493403.

Sweller, J. (1988) 'Cognitive load during problem solving: Effects on learning', *Cognitive Science*, 12(2), pp. 257–285. doi: 10.1016/0364-0213(88)90023-7.

Turkle, S. and Papert, S. (1991) 'Epistemological Pluralism and the Revaluation of the Concrete', in *Constructionism*, pp. 161–191. doi: citeulike-article-id:513444.

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L. and Wilensky, U. (2016) 'Defining Computational Thinking for Mathematics and Science Classrooms', *Journal of Science Education and Technology*. Springer Netherlands, 25(1), pp. 127–147. doi: 10.1007/s10956-015-9581-5.

Wilson, A. and Moffat, D. C. (2010) 'Evaluating Scratch to introduce younger schoolchildren to programming', *Proceedings of the 22nd Annual Workshop of the Psychology of Programming Interest Group*, pp. 64–75.

Wing, J. M. (2006) 'Computational Thinking', *Communications of the Association for Computing Machinery (ACM)*, 49(3), pp. 33–35. doi: https://www.cs.cmu.edu/~15110-s13/Wing06-ct.pdf.

Wing, J. M. (2008) 'Computational thinking and thinking about computing', *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 366(1881), pp. 3717–3725. doi: 10.1109/IPDPS.2008.4536091.

Yadav, A., Hong, H. and Stephenson, C. (2016) 'Computational Thinking for All: Pedagogical Approaches to Embedding 21st Century Problem Solving in K-12 Classrooms', *TechTrends: for leaders in education & training*. TechTrends, 60(6), pp. 565–568. doi: 10.1007/s11528-016-0087-7.