Towards better tools and methodologies to teach computational thinking to children

Laís V. Minchillo[1], **Augusto Vellozo**[2], **Edson Borin**[1], **Juliana F. Borin**[1]

[1]Instituto de Computação – Universidade Estadual de Campinas (UNICAMP)
Av. Albert Einstein, 1251 – Cidade Universitária, Campinas/SP - Brasil

[2]TecSinapse
Av Dr. Chucri Zaidan, 940 - 16º andar - São Paulo/SP - Brasil

lminchillo2@gmail.com, augusto.vellozo@tecsinapse.com.br,
edson@ic.unicamp.br, juliana@ic.unicamp.br

Abstract. *Computational Thinking is a useful skill to solve problems in all areas of knowledge. Efforts around the world aim to teach children this skill and in some countries it is already part of the curriculum. In this paper, we (i) describe our experiments teaching computational thinking concepts to children, (ii) describe the insights derived from this work, and (iii) propose a set of new hypothesis that should be tested in order to guide the development of a methodology to teach computational thinking to children.*

1. Introduction

Computational thinking (CT) is a tool to solve problems that applies to all areas of knowledge. The term was made popular by Wing (2006), who defined it as solving problems, designing systems and understanding human behaviour through concepts fundamental to computer science (CS).

Computational thinking must not be seen as a technical skill, but rather as a way to organize thoughts and solve problems, and it is natural to consider teaching it in school, either as a separate course, or as a new tool to existing disciplines [Wing 2008, Barr and Stephenson 2011, Barcelos and Silveira 2012, França and Amaral 2013, Lye and Koh 2014, Code.org 2017, CSTA 2017, Buitrago Flórez 2017, Eloy et al. 2017]. In many countries this skill is already part of the basic curriculum [UK Department for Education 2013, Smith 2016], and it is expected that new generations have a better understanding of technology and its different applications.

Teaching computational thinking to children is not a recent idea - Papert (1972) published a paper on the subject. According to him, children learn by doing and by thinking about what they do, and innovation in teaching must bring better things to do and better ways to think. At that time, the author also claimed that computing was by far the richest innovation area for teaching. Papert was one of the creators of Logo, a programming language designed to provide a fun environment for children to study and learn math and programming concepts [Logo Foundation 1991], as well as author of the book Mindstorms: Children, Computers and Powerful Ideas (1980), in which he defends the benefits of teaching computer literacy.

Thanks to Prodecad for hosting our experiments and to TecSinapse for financing this project

Several countries have included computational thinking in their school curricula, as well as programming and CS concepts and other related subjects (such as logical thinking, problem solving, abstraction, planning, among others). The United Kingdom's government has included CT and CS concepts in their national curriculum, stating that a high-quality computing education equips pupils to use computational thinking and creativity to understand and change the world [UK Department for Education 2013]. In the United States, ex president Obama launched an initiative to include Computer Science in the K-12[1] curriculum. In Brazil, the Brazilian Computing Society (SBC) is working to include Computer Science in the national curriculum [SBC 2017].

Over the past decade several authors have described what CT and CS concepts to teach and how to teach them, however, there is still no consensus on how to teach CT and how to include it in the schools. In this paper, we describe our experiments teaching computational thinking concepts to children using a mobile application paired with a physical robot and discuss the insights derived from this work. We also propose a new set of hypothesis that should be tested in order to guide the development of an effective methodology to teach computational thinking to children.

The rest of this paper is organized as follows: Section 2 presents the computational thinking concepts that are frequently discussed by key sources. Section 3 list the related work. Section 4 shows our experimental setup. Section 5 discusses the experimental results and, finally, Section 6 brings our conclusions and suggestions for future work.

2. Programming and CT concepts

Computational thinking is not the same as programming, but rather a skill programmers use in order to solve different problems. As a consequence, using programming as a way to teach CT is common in the literature [Lye and Koh 2014, Buitrago Flórez 2017, Eloy et al. 2017]. In this section we enumerate the main CT concepts cited by a few key sources: the Computer Science Teachers Association (CSTA), a membership organization that supports and promotes the teaching of computer science; Code.org, a non-profit organization dedicated to expand access to computer science in schools and increase participation by women and underrepresented minorities. They organize the annual Hour of Code campaign and provide a curriculum for K-12 computer science and they are supported by several companies including Amazon, Facebook, Google and Microsoft; SBC, Brazilian Computing Society, a non-profit organization whose goal is to encourage research and teaching in computing. We also chose three of the most cited papers in the CT topic - Barr and Stephenson (2011), Grover and Pea (2013) and Brennan, K. and Resnick, M. (2012) - as well as one paper well cited in Brazil - França and Amaral (2013) - one of the first works in the country to discuss teaching CT in schools.

These concepts are:

- Sequence: a series of individual steps.
- Algorithm: a sequence of instructions to solve a task.
- Loop: the execution of the same sequence multiple times.
- Event: an external action that triggers a command sequence.
- Conditional: making decisions based on predefined conditions.

[1] K-12 is the school curriculum for children aged up to 12 years old.

- Debugging and testing: executing an algorithm to find errors or to validate the proposed solution.
- Problem decomposition and modularization: divide a problem in smaller ones that can be solved more easily.
- Function: a sequence of instructions one can use with a given input to execute a task, possibly generating an output and modifying the original input to better suit it's purposes.
- Nested loop and conditional: a loop within a loop, or a conditional with another condition.
- Recursion: a function that calls itself.
- Parallelism: executing more than one instruction at a time, or execute more than a sequence of instructions at a time. Parallel tasks can be independent or not.

Table 1 shows the list of sources and concepts discussed by them.

Table 1. Computational thinking concepts by author

	CSTA 2017	Code.org 2017	SBC 2017	Barr and Stephenson 2011	Grover and Pea 2013	Brennan and Resnick 2012	França and Amaral 2013
Sequence		•	•	•	•	•	•
Algorithm	•	•	•	•	•		
Loop	•	•		•		•	•
Event	•	•			•	•	•
Conditional	•	•		•	•	•	•
Debugging	•	•	•	•	•	•	•
Test	•	•	•	•	•	•	•
Decomposition, functions, modularization	•	•	•	•	•	•	•
Nested for, nested if	•	•					
Parallelism	•			•	•		•
Recursion	•		•	•	•		•

Some sources also propose to group these concepts in modules, each one associated with a target age range. Tables 2 and 3 shows two of such module divisions.

Table 2. Computational thinking modules, Code.org

Level	Ages	Prerequisites	Concepts
1	4 to 6	Reading (basic)	Sequence, loop, event, problem solving
2	6 or more (Recommended 7 to 10)	Reading, math	Conditional, algorithm, debugging
3	6 or more (Recommended 9 or 10)	Completed module 2	Problem decomposition, functions, nested loop, nested conditional

Table 3. Computational thinking modules, CSTA K-12

Level	Ages	Concepts
1	8 to 11	Algorithm, problem solving, design and implementation, test, problem decomposition
2	11 to 14	Algorithm, design and implementation, parallelism, abstraction, problem decomposition, check different algorithms that solve the same problem
3	14 to 17	Functions, parameters, classes and pre-defined methods in problem solving, describing steps to developing software, explain how sequence and recursion are used to build algorithms, design and simulation of environments, abstraction, parallel programming

In Brazil, SBC (2017) has started to define how computing should be included (or modified) in the national curriculum. There are three main categories:

- Computational thinking: understand and use models and representation to describe information, processes and techniques to build algorithmic solutions; describe solutions though algorithms that can be executed in parts or in total by machines, as well as build computational models for complex systems; analyze problems and solutions to not only find automated solutions, but be able to evaluate their efficiency and correctness.
- Digital world: understand how information can be described and stored; understand how information is processed by computers and the relation between hardware and software; understand how digital devices communicate with each other, how the data is transmitted and how the integrity and safety of information is guaranteed.
- Digital culture: understand the impact of the digital revolution and advances in the digital world on humanity; utilize in an efficient and critical manner tools to help obtain, analyze, synthesize and communicate information of different formats and with different purposes; analyze ethical and moral questions created by the digital world.

Table 4 shows how SBC is grouping computational thinking concepts by school level.

Table 4. Computational thinking concepts by school level, SBC

Level	Concepts
Preschool Ages 3 to 5	Understand a problem and identify a sequence of steps to solve it. Represent these steps in an organized manner. Create steps to solve problems related to body movement and spatial trajectories.
Elementary school Ages 6 to 10	Abstraction to describe data such as lists and graphs. Identify the abstractions needed to build steps and to define algorithms that involve daily situations around the children. Use a visual language to represent algorithms. Understand problem decomposition.
Middle school Ages 11 to 14	Use visual and native languages to represent data and processes. Formalize the concepts of data structures. Use recursion to solve problems. Build new solutions by reusing solutions to problems of different context. Relate an algorithm in visual language to code in a programming language.
High school Ages 15 to 17	Work in groups designing solutions to problems integrated in other areas of the curriculum using computers, phones and other computing machines. Compare problems and reuse solutions. Analyze algorithm's cost and efficiency and justify if a solution is feasible and adequate. Argue about algorithm's correctness. Understanding the limits of computing to differentiate what can or can not be automated.

3. Related work

Several authors have described which CT and CS concepts to teach and how to teach them, and as we discussed in the previous sections, in some countries this has already been included in the national school curriculum offering every child the opportunity to learn and benefit from computational thinking. The fact that there is no consensus on how to teach CT and how to include it in the schools, especially in Brazil, has been part of our motivation for this work.

There has been several initiatives towards teaching CT in Brazil. Eloy et al. (2017) described an experience training teachers with the goal of promoting the practice of programming and development of CT in Brazilian public schools. In their pilot project they worked on four main areas: implementation in schools, curriculum design, teacher training and monitoring and evaluation. Their first guiding material to build the curriculum was the online platform Programaê[2]. Teachers were included in improving this curriculum through discussion sessions and questionnaires. They had over 500 students participating in the activities, and the sessions taking place in 2016 had an average of 80% student retention.

[2]http://programae.org.br/

Godinho et al. (2017) present a project to introduce CT and encourage children to become technology creators. The project was recognized by SBC in 2016 for bringing computing to children, teenagers and people who otherwise had little contact with the area. Their encounters included mini-courses, unplugged activities or tasks in Code.org's Hour of Code[3], Scratch[4], CodeMonkey[5], Monster Coding[6] or App Inventor[7]. Almost 300 students participated in their activities and through feedback questionnaires approximately 90% rated the experience as excellent or great.

Aono et al. (2017) use Scratch allied with an expository methodology to teach CT to elementary school students with ages 10 and 11. The children applied the concepts they learned into a project: building a "Flappy Bird" game. Every student that participated was able to build the game successfully, but all of them needed some help from the supervisors to implement the hardest parts, like the use of variables and counters.

Silva Junior and França (2017) discuss how existing tools are being used in the classroom in Brazil and their effectiveness. In total, 9 tools were analyzed for their interaction, platform, programming language and other characteristics. The tool found to be most adequate was Portugol Studio, since it is fully available in portuguese, it is appropriate for beginners and it features a user friendly interface. Besides that, it offers features to help teachers use it in their classes.

In this work, we aimed to teach CT to children using a mobile game paired with a physical robot, in an informal environment - unlike the regular classroom setting. Our goal was to have the children free to explore the application and learn from its use. The robot appears only as a motivating factor and as part of the play. The game itself saves logs of several actions, allowing us to analyze more than just the code the students developed, but rather see the interaction as a whole.

4. Experimental setup and methodology

We designed experiments with students from a local public school. In total there were twenty five children aged 9 to 11. The children were asked to solve problems by controlling a physical robot using a block-based visual programming language. The problems and the tools were designed to motivate the children to learn computational thinking skills. The tools and the methodology are described in the next sections.

4.1. Teaching tools

We designed and implemented an Android application and a physical robot to support our experiments. The application communicates with the robot via Bluetooth and provides a visual block-based programming interface through Google's Blockly library [Google for Education 2012]. Figure 1 shows a picture of the experiment's environment, including the robot and a tablet running the application.

In some of the activities, the user is required to solve a problem by programming the robot using a block-based visual programming language. The application provides a canvas in which the users can drag programming blocks and connect them to compose

[3] https://code.org/learn
[4] https://scratch.mit.edu/
[5] https://www.playcodemonkey.com/
[6] http://monstercoding.com/
[7] http://appinventor.mit.edu/

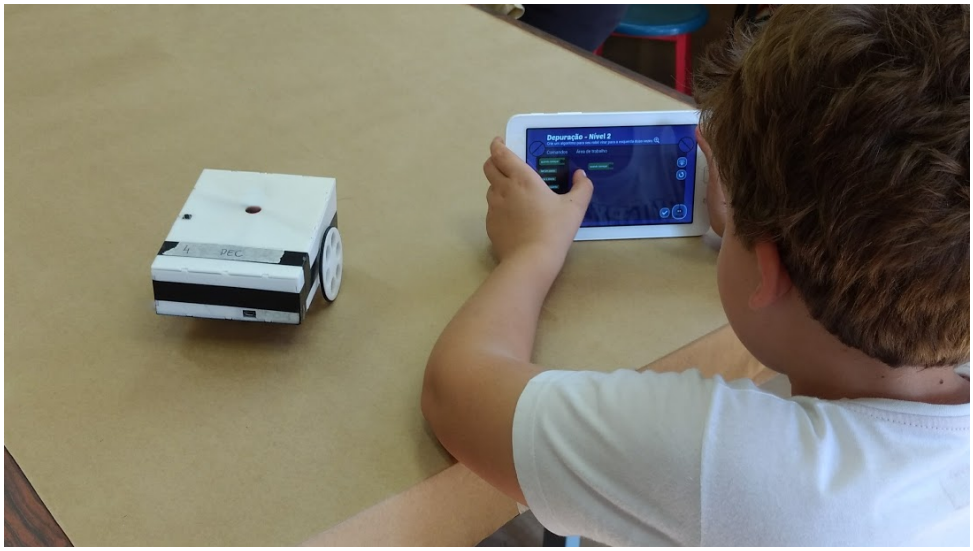
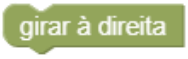


Figure 1. The experiment's environment, including both the robot and the application

their program. Blocks are shaped so that only connections that make sense are allowed. For example, the user may add a "Step forward" and a "Turn right" block and connect them to express a sequence of commands, however, a "Number" block may not be connected to a "Step forward" block. This feature minimizes issues associated with programming syntax, which are common in text based programming languages. Table 5 lists the programming blocks available in the application and Figure 2 shows a screenshot of the application, in which the user has a canvas on the right side and the blocks available for the given activity.

Table 5. Programming blocks

Step forward		Turn left	
Turn right		Number	
Repeat		Repeat a number of times	
Number comparison		Distance in front of the robot	
If		If else	

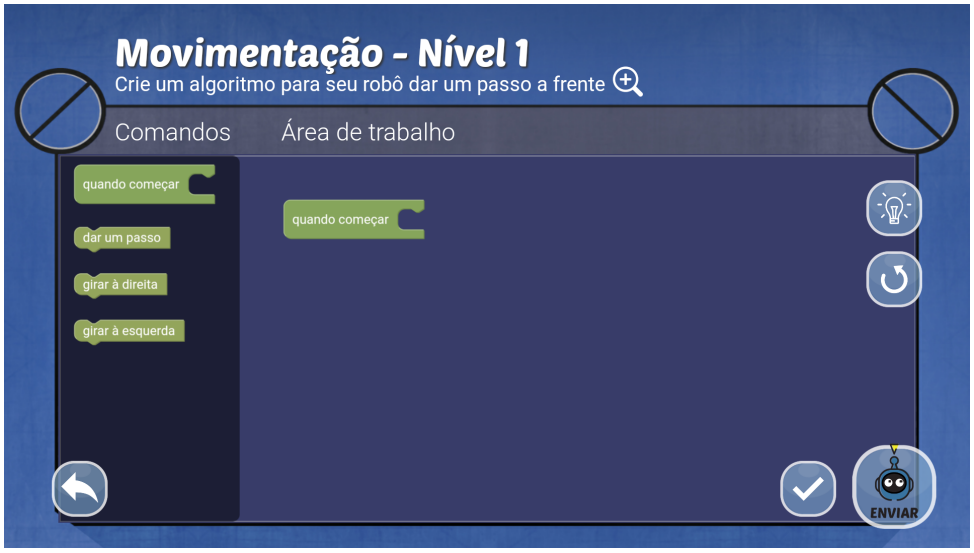


Figure 2. One of the challenges in the application

The Android application is a prototype of an educational game that has six levels, each one introducing new concepts and having multiple activities within it. Table 6 presents the levels and the CT concepts used in each one of them. The CT concepts were selected based on the age of the children and the amount of time we would have with them.

Table 6. Proposed levels and concepts

Levels	**Concepts**
Move the robot	Sequence
Have the robot make a path of a certain shape (e.g. square)	Sequence, algorithm
Find mistakes in given algorithms	Sequence, algorithm, testing, debugging
Have the robot repeat the same tasks multiple times	Sequence, algorithm, loop, problem decomposition
Have the robot decide on what action to take based on external conditions	Sequence, algorithm, conditional
Combine repetition and conditional scenarios	Sequence, algorithm, conditional, loop

There were three types of activities available: **tutorials** that explained what concepts the following activities would involve, and hints of how the user could solve the problems that would be presented next; **quizzes** to test the user knowledge, each one comprised by a question and three possible answers - of which only one was correct; **programming challenges** that involved either creating a new algorithm to solve a problem or to fix an existing algorithm.

In order to help our evaluation we had the application log some of the actions, as listed in Table 7, as well as the timestamps and the current activity identification number.

Table 7. Application logs

TUTORIAL-OPENED	User opened the tutorial
TUTORIAL-NEXT	Next tutorial page
TUTORIAL-PREV	Previous tutorial page
TUTORIAL-CLOSE	User closed the tutorial
QUIZ-OPEN	User opened the quiz
QUIZ-CORRECT	Correct quiz answer
QUIZ-INCORRECT	Incorrect quiz answer
QUIZ-CLOSE	User closed the quiz
ACTIVITY-OPEN	User opened the (programming) activity
ACTIVITY-HINT	User clicked hint
ACTIVITY-SEND	User sent the algorithm to the robot
ACTIVITY-CORRECT	User marked the current solution as correct
ACTIVITY-CLOSE	User closed the activity

4.2. Methodology

In total we had three sets of experiments - the first one with fourteen students, the second one with six students, and the third one with five students. In total there were ten girls and fifteen boys. One of our ideas was that children should work in pairs so that one could help the other. In the first experiment the children were divided in pairs by their teachers. This proved to be a poor strategy since some of the children were very uncomfortable with the person they were paired to. For this reason, in the next two experiments children were allowed to choose their pair and generally this proved to be a better approach.

For each experiment we had around seven encounters, one for introduction and the others for the actual activities. In the last session of each experiment we also asked the participants to answer a short feedback form - twenty two students filled this form out.

In the introduction session we provided a brief explanation of the research and interested students received a Consent Form to be signed by their parents or legal tutors. Also, as part of the introduction, each group was asked to name its robot.

After the children split in pairs and we handed over the robots and tablets, they were free to explore the application. The application itself is not capable of evaluating the user's solutions to programming challenges - there are several solutions to each of the proposed challenges, so simply having one correct algorithm and testing that it matches user's input would not suffice. To evaluate user's algorithms we would have needed a much more sophisticated setup so the application could have the robot's step-by-step information to only then determine if the desired solution was reached. We decided to let the students evaluate their own solutions by watching the robot to see if it behaved the way it was expected to. By doing so we could also observe whether the students had the ability to decide if their solution was correct.

We aimed at creating a playful and spontaneous environment, having as few evaluations and interventions as possible throughout the experiment, hoping it would encourage the children to *want to play*, rather than make them feel they *had to*. We could observe from the very beginning that competition was a much stronger motivating factor than collaboration. For that reason, in some occasions we proposed having the robots compete in a race with obstacles or an arena where the last robot standing would be the winner. The rules of the arena were simple: the robots had to keep moving and if it hit a wall or another robot, it was out for the round. Our goal was to combine the concepts of conditional (only move forward if there is no obstacle detected by the ultrasonic distance sensor, otherwise turn left or right) and repetition (never stop moving). In these competition environments it was clear that the students tried several solutions, reaching for the best one possible.

5. Results

At the introduction session we learned that most children were familiar with the use of smartphones, tablets and computers. They seemed very excited to work with the robots - particularly the boys. Asking each group to name its robot worked even better than anticipated because they developed a personal connection with it throughout the experiment.

During the experiments we aimed to intervene as little as possible while also being available to clear any doubts or to help if the children got stuck in any task. We observed that a very loose environment compromises their ability to focus on the proposed activities; however, it had a very positive effect on their will. This was confirmed by the teachers, who stated that the children were very excited to be participating in the experiments, especially considering they took place in their free period, when they could choose the activity they liked best.

One thing that stood out in their behavior was that they clearly preferred competition to collaboration. The pairs that were supposed to be working together divided the activities in a round robin fashion, and instead of helping each other they rushed their colleague so they could get to play with the robot faster. However, when an environment of team competition occurred - like the robot races - they would collaborate with their teammate to reach a better solution and try to win. In the regular activities, the pairs were not so motivated to keep trying different solutions when the first one didn't work, and would often lose attention in the current task and go see what other children were doing. Because of this behavior we expect that the children would reach their best when working individually or in a competition setting.

Another thing that stood out was that boys and girls showed a very distinct behavior: boys wanted to grab both the robot and the tablet straight away and go play with it - although not necessarily play within the scope of the proposed activities. Rather than that, most boys wanted to complete the tasks as fast as they could to either reach other groups that were in more advanced stages or to be the first to get to those stages. Girls on the other hand showed a much higher interest in reading the tutorials and completing the activities successfully. The children, in general, were very interested in discovering what the parts of the robot could do - like the "eyes" (an ultrasonic sensor) - and how to make it move or turn on the LEDs.

Because the application was unable to evaluate the user's solution, it was expected that the children themselves would figure out whether they had successfully completed the

given task or not. In many cases, they marked an activity as done even when they didn't program the robot as expected. There are some factors to consider:

- First, they could have thought that their solution was correct even if it wasn't, so it was an honest mistake.
- Being very familiar with other games and applications that can evaluate the solutions, they thought the application would only let them mark an activity as done if their solution was correct, so that's also an honest mistake.
- The third (and possibly worse) case is when the children mark the activity as done, when they knew it wasn't, so they could pass on to the next levels - to either reach their colleagues or to be the first to get to the next levels.
- There is also another extreme case: when the children's solution is correct but they aren't sure it is, so they keep trying to alter the code in order to see a better result in the robot.

Perhaps all of these issues can be solved by having a teacher or tutor check the child's solution before they can move on, and this certainly looks like a good solution to handle the aforementioned problems. However, the impact this aproach could have on the children's engagement should be evaluated.

We analyzed the application's logs in order to look for some correlations in the statistics. First, we graded the programming tasks using the following scale: 10 - completed the activity, 5 - partially completed the activity, and 0 - didn't complete the activity. Then, we compared the average grade in programming activities against the average number of wrong quiz answers. The result is presented in Figure 3, which contains a red line showing a linear fit of the data. Notice that the red line suggests that there is a negative correlation between these two metrics, i.e., the better the students scored in programming activities the less they select wrong quiz answers.

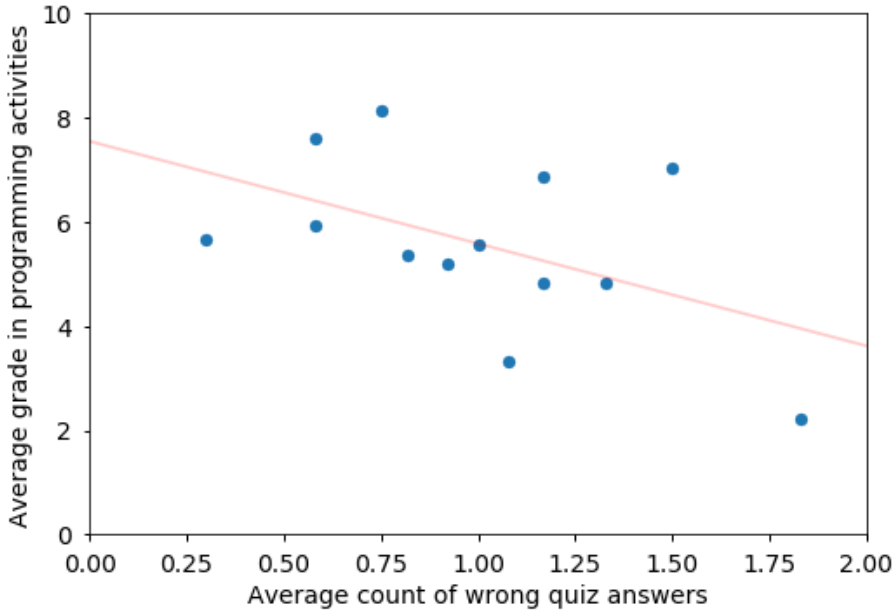


Figure 3. Grade in programming tasks versus count of wrong answers in quizzes

We also analyzed the percentage of the tutorials the children read. Each tutorial had a certain number of pages, and we checked the number of pages the children read.

Some children only opened the first page and already quit - meaning they never read the following pages.

Prior to the experiments, we anticipated that the children who read the tutorials would have a better performance in the quizzes and programming activities. As Figure 4 indicates, there seems to be a positive correlation between reading the tutorial pages and scoring higher on programming activities.

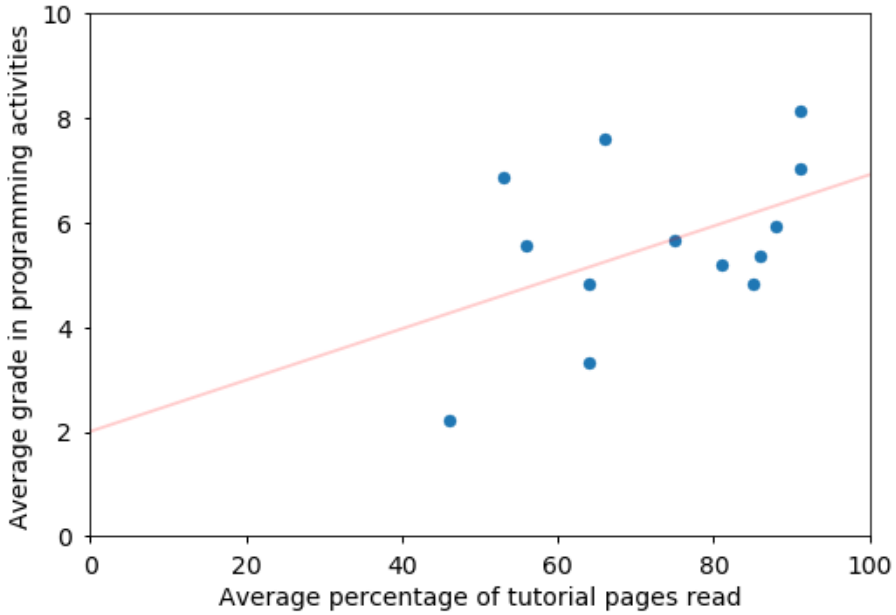


Figure 4. Grade in programming tasks versus percentage of tutorial pages read

Even though the linear regression indicates that there is a positive correlation between grades in programming activities and number of tutorial pages read, we noticed that different groups with very similar grades had read different amounts of tutorials pages. One of the possible reasons this happened is due to the different children's backgrounds. Some of them could already have more knowledge of similar games and activities, and therefore even not reading the tutorials - or reading less of them - had a better comprehension of the activities and how to solve them.

Figure 5 shows the average count of wrong quiz answers against the percentage of tutorial pages read. Again, as indicated by the linear regression, there seems to be a negative correlation between the percentage of tutorial pages read by the students and the amount of incorrect quiz answers.

In general, these results indicate that children who read more tutorials achieve higher grades in programming tasks and select fewer incorrect answers in quiz activities.

We also evaluated how well the children understood each concept by looking at the average grade that was obtained for the programming tasks involving that concept. Figure 6 shows a graph of average grades per concept. Activities involving the first concepts - movement, sequence and debugging - were completed with success by most groups, indicating that the students either learned the concepts or had previous knowledge of them. The activities that involved the loop concept were only completed partially, with a much lower grade than the previous ones. Finally, no child was able to achieve the

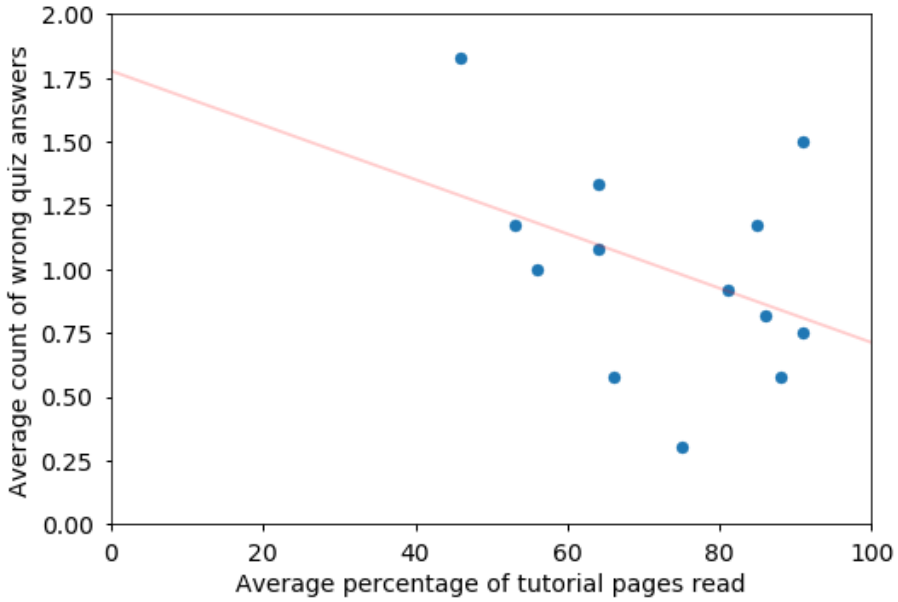


Figure 5. Count of wrong answers in quizzes versus percentage of tutorial pages read

expected results for the conditional concept. It is still unclear whether children at that age can't understand this concept well or if the way it was presented in our experiment was too complicated.

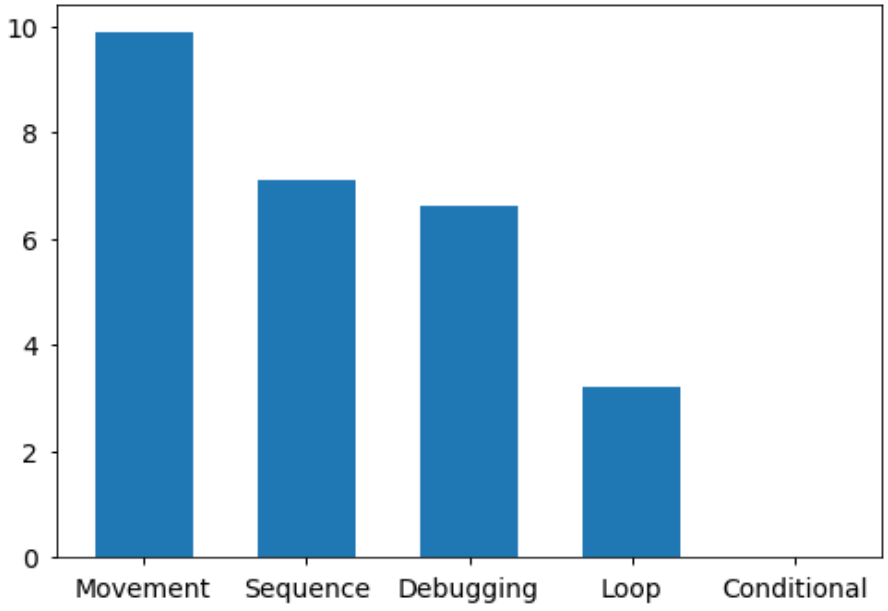


Figure 6. Average grade for all groups by concept

Table 8 shows the answers to the feedback form. Each question had three possible answers: yes, partially, and no. Overall, the feedback provided by the children was very positive - we only had 5.5% of negative answers across all questions, against 25.5% neutral and 69% positive. The only question that did not have a significant positive outcome was "Is the application easy to use?", indicating that we need to evaluate what's the best

Table 8. Questions and answers in the feedback form

Question	Positive answers	Neutral answers	Negative answers
Would you use the application and robot again?	16	5	1
Would you recommend the application and robot to a friend?	17	4	1
Is the application easy to use?	7	14	1
Is the application content fun and interesting?	19	2	1
Did you like working in pairs?	17	3	2
Average	**15.2 (69%)**	**5.6 (25.5%)**	**1.2 (5.5%)**

way to present the programming interface. The teachers also gave us very positive feedback, stating that the students were very excited to be participating and that they felt this was an excellent way to keep their attention.

The form also included space for comments and suggestions. Many of the children wrote that they would like to see other types of robots and other features: "*My suggestion is to create other types of robot and present it to other schools.*", "*I wish it [the robot] could talk and that it had a laser.*", "*My suggestions are that the robot should have arms and legs like other toys and be able to speak.*", "*A suggestion is to make the robot faster.*".

About working in pairs, the children were conflicted: some liked it ("*I liked working in pairs. It's fun that every level is different and has a tutorial.*") and other did not ("*I liked the idea of working in pairs but I didn't like my pair.*").

Finally, many of the feedback was related to having more activities and levels: "*I think it would be cool if you made more activities and different things to do.*", "*I wish there were more worlds.*", "*More levels. It would be fun and come back next year so more people can enjoy this project.*".

6. Conclusion

Over the past decade several authors discussed the importance of and methods to teach CT and CS concepts to children, nonetheless, there is still no consensus on the best teaching methodology. In this paper, we described our experiments teaching computational thinking concepts to children between 9 and 11 years old and share our insights in order to support the development of effective tools and methodologies to teach CT to children.

Our results indicate that:

- children were very excited to interact with the robot;
- by giving a name to the robot, children established a personal connection with it improving their engagement in the experiment;
- competition is a motivating factor and encourages teamwork;

- children did not have any difficulty with sequence concept, however, they had a hard time applying loop and conditional concepts.

Additionally, the interaction with the children helped us design the following **new hypothesis**:

- having a physical instrument to interact with is a motivating factor, whether it is a robot, a board, a set of command pieces, etc;
- adding tests, exams or other forms of formal evaluation can have a negative impact on the way children see the CT education project;
- leaving the children free to use the application and play with the robot make them more comfortable, but it does not necessarily mean a positive impact on their learning;
- having one robot per child or projects that require collaboration between the children to be completed improves learning experience and children's engagement in the activities;
- competition improves children's engagement in the activities;
- have teachers suggest activities that complement the regular disciplines may improve learning and engagement in the classroom.

Future experiments with a higher number of children and for a longer period are needed in order to test these hypothesis. We believe that such a study would greatly contribute to the development of methodologies to teach CT and CS to children.

References

Aono, A. H., Rody, H. V. S., Musa, D. L., Pereira, V. A., & Almeida, J. (2017) A Utilização do Scratch como Ferramenta no Ensino de Pensamento Computacional para Crianças. XXV Workshop sobre Educação em Computação, Anais do XXXVII CSBC (p. 2169).

Barcelos, T. S. & Silveira, I. F. (2012). Pensamento computacional e educação matemática: Relações para o ensino de computação na educação básica. In XX Workshop sobre Educação em Computação, Curitiba. Anais do XXXII CSBC (Vol. 2, p. 23).

Barr, V. & 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), 48-54.

Brennan, K. & 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).

Buitrago Flórez, F., Casallas, R., Hernández, M., Reyes, A., Restrepo, S., & Danies, G. (2017). Changing a Generation's Way of Thinking: Teaching Computational Thinking Through Programming. Review of Educational Research, 87(4), 834-860.

Code.org. (2017) Curriculum, https://code.org/educate/curriculum.

CSTA. (2017) CSTA K-12 Computer Science Standards, Revised 2017, https://sites.google.com/site/cstastandards/standards.

Eloy, A. A. D. S., Martins, A. R. Q., Pazinato, A. M., Lukjanenko, M. D. F. S. P., & Lopes, R. D. D. (2017, June). Programming Literacy: Computational Thinking in Brazilian Public Schools. In Proceedings of the 2017 Conference on Interaction Design and Children (pp. 439-444). ACM.

França, R. S. de, & Amaral, H. J. C. do. (2013) Proposta Metodológica de Ensino e Avaliação para o Desenvolvimento do Pensamento Computacional com o Uso do Scratch. In Anais do Workshop de Informática na Escola (Vol. 1, No. 1, p. 179).

Godinho, J., Torres, K., Batista, G., Andrade, E., & Gomide, J. (2017) Projeto Aprenda a Programar Jogando: Divulgando a Programação de Computadores para Crianças e Jovens. XXV Workshop sobre Educação em Computação, Anais do XXXVII CSBC (p. 2140).

Google for Education. (2012), Blockly, https://developers.google.com/blockly/.

Grover, S. and Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. Educational Researcher, 42(1), 38-43.

Logo Foundation. (1991) Logo, http://el.media.mit.edu/logo-foundation/.

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

Papert, Seymour. (1972) Teaching Children Thinking, Programmed Learning and Educational Technology, 9(5), 245-255.

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

SBC. (2017) Referenciais de Formação em Computação: Educação Básica, http://www.sbc.org.br/noticias/10-slideshow-noticias/1996-referenciais-de-formacao-em-computacao-educacao-basica.

Silva Junior, S. M. da, & França, S. V. A. (2017) Programação para todos: Análise Comparativa de Ferramentas Utilizadas no Ensino de Programação. XXV Workshop sobre Educação em Computação, Anais do XXXVII CSBC (p. 2199).

Smith, Megan. (2016) Computer Science For All, The White House, https://www.whitehouse.gov/blog/2016/01/30/computer-science-all.

UK Department for Education. (2013) National curriculum in England: computing programmes of study, https://www.gov.uk/government/publications/national- curriculum-in-england-computing-programmes-of-study.

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

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