Text-based Programming in Elementary School: A Comparative Study of Programming Abilities in Children with and without Block-based Experience

Marcos J. Gomez
Universidad Nacional de Cordoba
Cordoba, Argentina
mgomez4@famaf.unc.edu.ar

Marco Moresi
Universidad Nacional de Cordoba
Cordoba, Argentina
mrc.moresi@gmail.com

Luciana Benotti
Universidad Nacional de Cordoba / CONICET
Cordoba, Argentina
benotti@famaf.unc.edu.ar

ABSTRACT

This paper describes an elementary school intervention to teach a text-based programming language to 10-11 year old students. We compare students with no previous programming experience with students with 3 semesters of experience with a block-based programming language. We analyze students' performance and learning based on detailed logs in an online programming platform and on multiple choice tests. Although both groups have a similar percentage of syntactical errors, the experienced group showed a better performance on exam scores and a lower number of test case errors. These findings suggest that, 10-11 year old students benefit from block-based experience when learning a new text-based programming language.

CCS CONCEPTS

• **Social and professional topics** → **K-12 education**; • **Applied computing** → *Computer-assisted instruction*; • **Software and its engineering** → *Imperative languages.*

KEYWORDS

Learning analytics; elementary education; text-based languages; block-based languages

ACM Reference Format:
Marcos J. Gomez, Marco Moresi, and Luciana Benotti. 2019. Text-based Programming in Elementary School: A Comparative Study of Programming Abilities in Children with and without Block-based Experience. In *Innovation and Technology in Computer Science Education (ITiCSE '19), July 15–17, 2019, Aberdeen, Scotland UK*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3304221.3319734

1 INTRODUCTION

In the last years, governments, universities, companies and organizations around the world have joined forces to bring the teaching of Computer Science (CS) at elementary, middle and high schools [11, 14, 22, 29]. In this context, there is still an ongoing debate on whether elementary school children are developmentally ready to learn programming or even to use computers (see for example [6]).

While research on teaching programming in higher educational levels is vast, and there is increasing consensus on curricular designs (e.g., [15]), studies over elementary schools are not as common [8]. Research based on teaching CS in elementary school can be classified (according to [2]) in three categories: unplugged activities [3], block-based visual programming [28] and robotics [5, 21]. Most of the documented experiences use block-based platforms such as Scratch [24, 28], Snap! [16], Alice [9], Code.org [17]. Block-based languages are supposed to allow students to focus on concepts, leaving aside the syntactic complexity present when learning to program using a text-based language. But, how well can elementary school students transition from block-based languages to text-based languages? In other words, our research question Q1: *do children with block-based experience have programming abilities that make it easier to learn a text-based language*?

In order to address this question we pose two non directional hypotheses. Ha is related to syntactic errors, Hb is related to semantic errors, defined as follows. A student makes a syntactic error when her/his program does not comply with the grammar of the programming language. A student makes a semantic error when she/he holds a misconception about the program behavior. In Section 3 we formally define these two kinds of errors.

Ha (null): There is no significant difference in the amount of syntactic errors that students with and without previous block-based programming experience make.

Hb (null): There is no significant difference in the amount of semantic errors that students with and without previous block-based programming experience make.

With the purpose of testing these hypotheses and contributing to a CS curriculum selection and scope, we designed an observational study for elementary school children. We carried out programming lessons in a real school setting focusing on loops, conditionals, sequences, and parameters; and their application to programming. The main contributions of this paper are three. First, we introduce an online coding tool that logs an student learning process while programming, registering each submission made by the student along with formative feedback automatically generated by the tool. Second, we analyze comparatively how 10-11 year old children with and without previous programming experience create programs in a text-based programming language. Finally we evaluate the

ITiCSE '19, July 15–17, 2019, Aberdeen, Scotland UK

ACM ISBN 978-1-4503-6301-3/19/07…$15.00
https://doi.org/10.1145/3304221.3319734

students on fundamental programming concepts in programming challenges involving code interpretation.

The paper is organized as follows. Section 2 reviews elementary school experiences and situates this paper with respect to previous work. Section 3 presents the online coding tool, called Mumuki, as well as the text-based programming language used for our observational study. Section 4 describes the student groups involved as well as the methods for data collection. Section 5 discusses the results of our study, their implications and potential threats to validity.

2 PREVIOUS WORK

There are several tools which allow elementary school children to create, run and debug simple computer programs using specific platforms that are both challenging and attainable for most children [7, 13, 18]. Previous work [12, 18] reports that even children who are preliterate, have diverse background and different developmental stages, engage with tangible platforms, such as wooden or virtual blocks. Some implications of learning programming at such an early age have been analyzed. According to Clements [7], children who use computer assisted programs have the opportunity to analyze a situation and reflect on the properties of objects they have to manipulate. Clements concluded, that it links intuitive children knowledge (about movements and drawings) into more explicit formal ideas using programming language and "facilitates positive beliefs about the creation (of technological products)". Other studies [12] have suggested that computer programming develops higher order thinking skills such as metacognitive competencies. Students who used Logo and tangible programming could propose different methods to solve a problem and had the ability to debug a program. This means, that children are monitoring their thinking looking for the program mistakes. This is a strong indicator of metacognition.

While exploring how to teach programming to little children some researchers have found that the difficulties in children programming laid in their immature motor skills and on syntax problems [13, 25]. Thus, there has been a vast development on specific programming platforms to address the developmental traits of small children (such as Toon Talk, Scratch Jr, CHERPS, etc). In this context, block-based programming has been an interesting line of work to teach programming to little children. Morgado et al [25] propose lesson plans for preschool and elementary school children based on their extensive experiences in the classroom. They documented how elementary school children engage while learning competences and concepts such as syntax, parameter passing, compound procedures, parallelism and concurrency, communication channels, input and client and servers using Toon Talk platform. As the result of their exploratory study, they suggested different strategies to teach each concept, such as using metaphors to teach variables. Although most interventions to teach programming to preschool and elementary school children achieve high student engagement and participation, we still need to understand more how the use of these platforms promotes developing programming abilities and learning particular concepts.

There are few previous studies that explore what programming concepts children can understand at different developmental stages to establish school curricula content and scope. Martinez et al [21] found that 4 to 10 year olds could learn and apply basic programming concepts of sequencing, conditionals, and loops by engaging in a mix of unplugged activities and block-based programming. Magnenat et al [20] taught robot programming to different elementary school children using a block-based language to program a robot action for different events. Comparing the groups performance, they found that most children understood and solved simple tasks such as moving a robot upon a touch of a button or identifying robot's instructions. Both studies teach and evaluate using the same block-based language. Hence, it is not clear whether the learning is language specific or conceptual.

Dagiene et al [10] compared students from Finland, Sweden and Lithuania of ages 7 to 12 performing computational thinking tasks exercises. Using multiple choice questions, they evaluated concepts such as graph analysis, search algorithms, data structures, pattern matching, etc. They found no strong difference across age groups, but rather among student backgrounds.

The previous work more related to the research question Q1 that we posed in Section 1 are the following two. First, Armoni et al [2] found that students who learned Scratch in middle school more quickly grasped concepts in text-based languages when they reached high school, although they did not perform significantly better in the course assessments. Then, Weintrop and Wilensky [31] compared separate groups of high-school students learning basic programming constructs. One group learned a block-based language while the other learned a text-based language. Their results suggest that novice programmers' understanding of basic programming constructs (such as conditionals and loops) is better in block-based languages. Differently from us, both studies focus on high-school students.

Summing up, there is considerable enthusiasm around the world towards introducing young children to programming. However, there is a need to systematically evaluate the impact of such initiatives on the learning outcome, independently from the programming language. In this paper, we explore elementary school students' conceptual gain during their transition from block-based programming into text-based programming.

3 THE ONLINE PROGRAMMING TOOL

In this section we introduce Gobstones and Mumuki, which we use for our study. Gobstones [19] is a text-based programming language and Mumuki is an online programming environment. A screenshot of a sample programming exercise using the Gobstones language in Mumuki [4] is shown in Figure 1.

Gobstones [19] is a programming language developed for CS1 university students who do not have previous programming knowledge. Gobstones is an imperative programming language whose syntax is based on a simplified C syntax, and its output is restricted to moving color balls on a board. Martinez-Lopez et al. in [19], describe the three elements that are part of Gobstones philosophy: a graphical output, a simplified syntax and a focus on fundamental programming constructs. As can be seen in Figure 1, the graphical output is simple and concrete. It is a rectangular board of a programmable size, made of cells which can hold colored balls, such as the blue ball in the figure. The balls can be manipulated by a head that works on a single cell at a time, and is depicted as a yellow

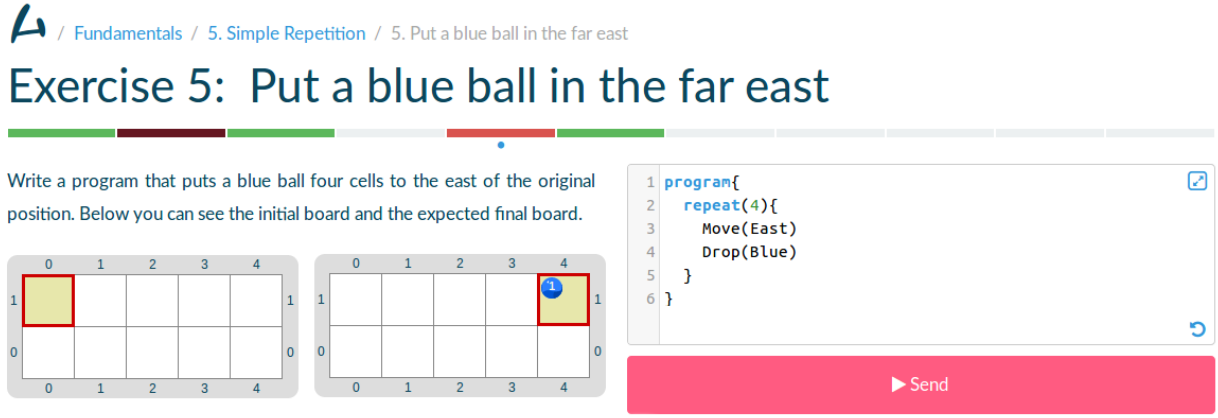

Figure 1: Mumuki screenshot with an exercise description on the left and a program written by a student on the right.

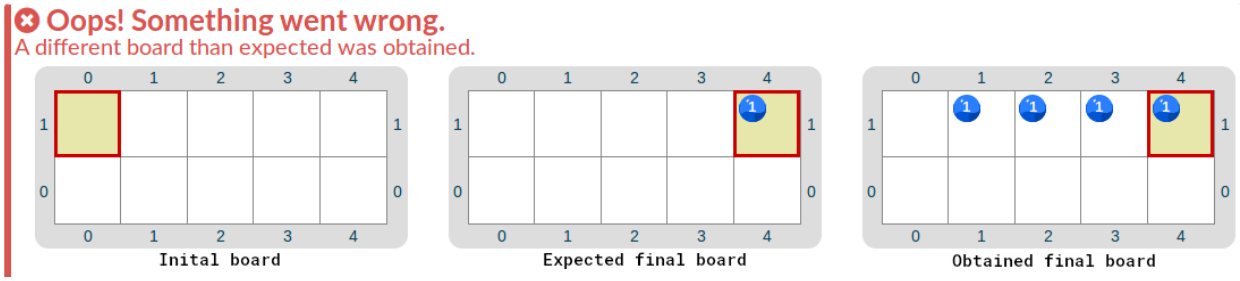

Figure 2: Feedback generated by Mumuki for the program in Figure 1, the program is executable but incorrect.

cell with a red border. Each board has only one yellow cell, which means that the head is over this cell. The head can execute the following primitives defined in [19]: (1) `Drop`, which puts a ball of a given color (for example Red, Green, Blue, or Black) on the current cell. (2) `Grab`, which takes a ball of a given color from the current cell, if it exists, and makes the board explode otherwise. (3) `Move`, which moves the head in a given direction (North, East, South, or West), if there is a cell in this direction, and makes the board explode otherwise.

Mumuki [4], as can be seen in Figure 1, includes a progress bar indicating which exercises have been solved (in green), which have been attempted but are either not executable because of a syntax error (in dark red) or incorrect because of a semantic error (in light red), and which have not been attempted yet (in grey). The students can solve the exercises in any order they want, but Mumuki suggests an order with the progress bar. The current exercise is marked with a blue dot. The figure shows the title and the description of the programming problem and makes explicit the board where the program starts running and the expected final board. Students have to write a Gobstones program that transforms the initial board into the expected final board. Given this goal, we define two possible kinds of errors. A student makes a *syntactic error* when her/his program does not comply with the grammar of Gobstones and is marked in *dark red* by Mumuki. A student makes a *semantic error* when the program reaches a different final board than that expected. That is, she/he holds a misconception about the program. Semantic errors are marked in *light red* by Mumuki. A light red program is syntactically correct but does not achieve the exercise goal. As follows we exemplify both kinds of errors.

The exercise in Figure 1 asks the student to write a program that drops a blue ball four cells to the east of the initial position. In the right hand panel, Mumuki includes an editor where the student can write the solution. In the figure, the solution is marked in light red. The feedback generated by Mumuki when the student presses send for this incorrect solution is shown in Figure 2. The feedback shows that the obtained final board is different than the expected one. Figure 3 shows a correct solution for the running example. The difference with the program in Figure 1 is that `Drop(Blue)` is outside the repeat. In Figure 4 the student made a syntactic error. The student forgot to write a closing parenthesis. In this case, the exercise is assessed as dark red because the code is not syntactically correct and therefore it cannot be executed.

Every solution submitted by a student is stored by Mumuki in its database. The timestamp, the code, and the tests results are also stored, so the full history can be reproduced. This information could be used by a teacher in order to improve the exercises, to explore common mistakes, etc. In this paper we use it in order to analyze the kinds of errors made by the students.

```
SOLUTION: program{
            repeat(4){
                Move(East)
            }
            Drop(Blue)
          }
```

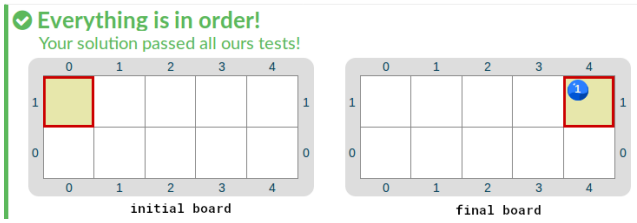

Figure 3: Correct solution for the exercise in Fig. 1.

```
SOLUTION: program{
            repeat(4){
                Move(East
            }
            Drop(Blue)
          }
```

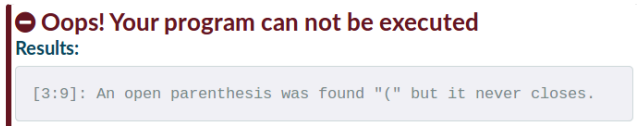

Figure 4: Non executable program for the exercise in Fig. 1.

4 STUDY DESIGN

In this section we describe our observational study designed to test the hypotheses posed in Section 1. In 4.1 we present the schools, the intervention design and the background of the students in programming. In 4.2 we describe the kinds of logs made per school using Mumuki and the multiple choice exam.

4.1 The setting for the study

We performed interventions in two different privately run primary schools receiving state public funds. There are no children below the poverty line and most children have middle class parents. The schools serve the same neighbourhood. In both of them we taught Gobstones using Mumuki. Both experiments were carried out at the schools during school hours with 10 and 11 year old students, attending 5th grade of elementary school. One of these schools is called Escuela Nueva Juan Mantovani (ENJM), and the other, Nuestra Señora de Valle (NSDV).

The complete experience lasted 5 hours in each school. While using Mumuki, students worked alone or in groups of two or three. The same teacher was in charge of the experience at both schools. Students were asked to complete 23 programming exercises organized in 2 Mumuki lessons. Through these exercises, students learned programming concepts such as sequence, conditionals, loops and parameters. After the experience, students took part of a multiple choice exam.

Students from ENJM had previous experience with fundamental programming concepts. Programming is part of the core curricula of the school. The students that participated in the experience had three semesters of previous experience learning to program. Students had 1 hour of programming per week. The 3 programming semesters were taught at ENJM following an experience based pedagogy and the curriculum was project-based. A complete description of the curricula and the didactic sequences used can be found in [1]. In short, the ENJM students were introduced to programming and other CS topics by developing their own video-games and animations with Scratch as well as unplugged activities.

Students from the other school, NSDV, did not have any previous experience in programming. Their teachers and the head of the school reported that programming was not part of the curricula. The students were asked to auto-assess their programming background previous to this experience. All of the students confirmed that they have not been exposed to programming in other settings (e.g., at home, or in some outreach program).

4.2 Programming logs and exam

In total, 129 students participated in the experiences. 63 from ENJM and 66 from NSDV. The ENJM students used Mumuki in 32 groups of 2 or 3 students and 5 groups of 1 student, while in NSDV, the students used Mumuki in 31 groups of 2 or 3 students. Each experience lasted in total 5 hours. Due to school restrictions, in the ENJM experience, 5 one hour meetings were held, while in NSDV, the experience lasted 2 two-hour-and-a-half meetings.

Groups of students that participated in the experiences produced a total of 2066 errors while attempting a median of 17 exercises per group in each school. The Table 1 presents the total of errors per school. The 31 groups of students from NSDV made 1225 errors, and the 37 groups from ENJM made 841 errors. Although there were more groups in ENJM, they made less errors than the NSDV students. The table also reports the number of exercises attempted by all the groups as well as the time on task defined as the sum of minutes between submissions for all students in a given day. The time on task for both groups was almost the same.

School	ENJM	NSDV
Number of student groups	37	31
Number of exercises attempted	607	611
Time on task (in minutes)	3046	3072
Number of syntactic errors (dark red)	305	385
Number of semantic errors (light red)	536	640
Total number of errors	841	1225

Table 1: The upper part reports basic metrics of the investment in learning comparatively for both schools. The lower part compares the types of program errors per school.

After the experience, students took part of a multiple choice exam. The exam was not mandatory. The exam was composed of 6 multiple choice questions. Each question had a program written in Gobstones and an initial board. The student had to apply the Gobstones program to the initial board, and select the correct final

board from 4 possible options. A sample question from the exam[1] is shown in Figure 5. The correct answer is Option 2.

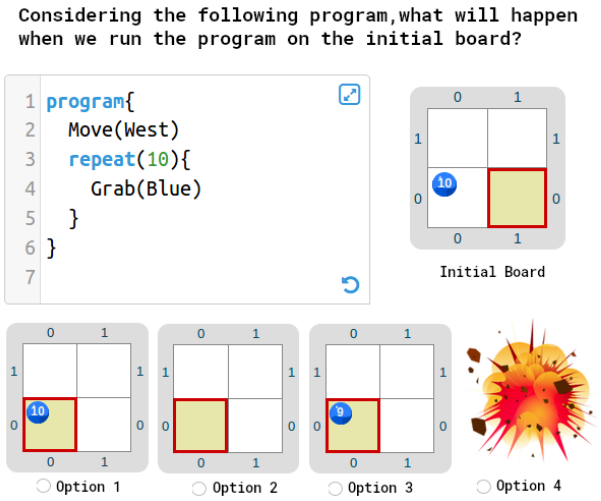

Figure 5: Sample exercise from the multiple choice exam.

All questions had the same weight on the final score. The scale of the exam score was from 0 to 10. Table 2 shows the number of students per school that participated in the experience divided by gender, and how many students took the multiple choice exam.

School	ENJM			NSDV		
Gender (F/M) and Total	**F**	**M**	**T**	**F**	**M**	**T**
Number of students participating	30	33	63	37	29	66
Number of students tested	26	31	57	18	11	29

Table 2: Participants distribution (not mandatory exam).

5 FINDINGS

In this section we describe our findings with respect to the hypotheses posed in Section 1. Using the online system logs, we first compare the amount of syntactic and semantic errors for both schools. The system logs were made by groups of 1, 2 or 3 students. Then we present the results of the multiple choice exam by school. The exam was taken individually. Finally we discuss the limitations of the study.

5.1 Programming logs findings

Table 3 shows the amount of syntactic (dark red) and semantic errors (light red) made by ENJM and NSDV schools during the experience. In order to compare schools performances while solving programming exercises in Mumuki we define two metrics. The *semantic ratio* per group is the amount of programs with semantic errors that the group submitted divided by the amount of exercises that the group attempted. The *syntactic ratio* is analogous but it considers the syntactic errors.

[1]The complete test is available here (in Spanish): url anymized

Table 3 reports the average per group (of 1, 2 or 3 students) for these two metrics for both schools, in the ratio column. It also reports the raw amount of programs in both types for both schools, in the # column.

	ENJM		NSDV		ratio t-test	
	#	ratio	#	ratio	p	t
Syntactic errors	305	0.70	385	0.62	0.588	0.543
Semantic errors	536	0.74	640	1.01*	0.024	2.31

Table 3: Average group ratios and raw amount of syntactic and semantic errors. Statistically significant differences are marked with * for unpaired t-test performed on the ratios.

The table also presents the results of applying unpaired t-tests to ENJM and NSDV average ratios for both metrics. The semantic ratio average of ENJM is 0.74 while in NSDV average is 1.01. In other words, after overcoming syntactic errors, ENJM groups make semantic errors in 7 out of 10 attempted exercises in average. In contrast, NSDV groups make semantic errors in 10 out of 10 exercises. The unpaired t-test shows that there is a difference in this metric that is statistically significant between groups from ENJM and NSDV (p=0.024). These results give evidence for rejecting the null hypothesis Hb (null) in favor of the alternative one: groups with previous programming experience make significantly less semantic errors than groups with no such experience.

The difference between schools for the syntactic ratio average is not statistically significant, based on the unpaired t-test, as can be seen in the table (p=0.588). Therefore, the Ha (null) hypothesis cannot be rejected. This is coherent with the fact that both groups do not have previous experience in text-based coding languages, so syntactic errors appear in similar proportion in both schools.

5.2 Multiple choice exam findings

Figure 6 shows a box plot of the exam marks (over a maximum mark of 10 points) for both schools. The median mark for NSDV is 6.66 and for the ENJM is 8.33 (the difference is 1.67 points). The exam was designed to include as alternative answers common programming misconceptions. For example, in Figure 5 the incorrect options (1,3 and 4) illustrate different misconceptions about the loop behavior. The difference between the median marks in both schools is statistically significant using unpaired t-tests (p=0.0014). This gives further evidence for rejecting the null hypothesis Hb (null) in favor of the alternative one.

5.3 Limitations of the study

In this section we discuss the potential threats to validity for the present study.

First, there is no formal assessment of the background of the NSDV students. Their previous background is self-reported by their teachers, directors and the students themselves as described in Section 4. All of them confirmed no previous exposure to programming.

Second, the temporal organization for both schools was not exactly the same. ENJM students were taught the material in 5 one-hour meetings while the NSDV students had 2 lessons of 2.5 hours. Meetings were held once a week for both schools. The teacher

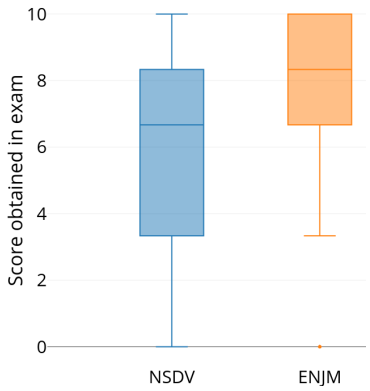

Figure 6: Box plot of the exam marks for both schools.

observed that, at the beginning of every meeting, students spent some time getting ready for the task. However, it does not seem to impact the time on task as reported in Table 1.

Finally, only 29 of the 66 students that participated in the NSDV experience decided to take part in the exam, as reported in Table 2. In the ENJM experience 57 out of 63 decided to take the exam. The exam was not mandatory at any school. While students may have different reasons not to take an exam, self-confidence in their performance is probably one of them.

6 DISCUSSION AND CONCLUSIONS

In this paper we describe an elementary school study carried out at two schools with 10-11 year old students. One of the them had 1 year and a half of experience learning programming with a block-based language. The other had no previous programming experience. In this context we explored the research question Q1: *do children with block-based experience have programming abilities that make it easier to learn a text-based language?*

We comparatively analyzed errors in a text-based programming language using logs from a programming tool and a multiple choice exam. We evaluated two non directional hypotheses Ha and Hb. Ha is related to the amount of syntactic errors and Hb to the semantic errors made in a text-based programming language new for both of them. We use logs from a programming environment called Mumuki which automatically corrects programming exercises in order to test both hypotheses.

When testing Ha we find that children make a similar amount of syntactic errors regardless of their background. This finding is coherent with previous work [26, 27] which argues that block-based languages do not address syntactic problems but merely delay them. Such previous work was performed at university level while ours is at primary school. A reason to start with block-based languages could be that not everything can be learned at the same time. Weintrop and Wilensky [31] compared separate groups of high-school students learning basic programming constructs. One group learned a block-based language while the other learned a text-based language. Their results suggest that novice programmers' understanding of basic programming constructs (such as conditionals and loops) is better in block-based languages.

We evaluate Hb using both the logs and the multiple choice exam. In the logs we find that, after overcoming syntactic errors, the children with experience make significantly less semantic errors than the children without block-based experience. In average, 7 out of 10 attempted exercises had an error for each experienced group while each group without experience made a semantic error in 10 out of 10 exercises. The Gobstones exercises, such as the one in Figure 1, allow for small changes to the code that have a strong impact on the correctness of the exercise as discussed in Section 3. It is possible than through trial-and-error the students can reach a correct solution while programming misconceptions remain. By considering the amount of semantic errors, we observe more trial-and-error in the school with no background in block-based languages.

We also evaluate the Hb hypotheses through a multiple choice exam. The exam was designed to provide common misconceptions [2, 30, 31] associated to fundamental programming constructs as alternative wrong answers. We find that the mark in the exam is significantly higher for children with block-based programming experience. Based on the logs and the exam results, we conclude that the group with previous block-based experience made less semantic errors than those without experience. Armoni et al [2] found that students who learned Scratch in middle school more quickly grasped concepts in text-based languages when they reached high school, although they did not perform significantly better in the course assessments. Differently from them, our study was done in elementary school. Moreover, we did find significant differences in the exam between our schools. This may be due to the fact that Armoni et al text-based experience was longer than ours. Our experience lasted 5 hours in total while theirs was carried out during a whole semester (total hours are not reported). As a result, in their case, the group of students with no previous block-based experience had more time to catch up.

Summing up, we have presented evidence that 10-11 year old students with block-based experience learn a new text-based language more easily than similar students with no block-based experience. In future work we will compare the schools across different kinds of programming misconceptions and analyze differences by gender.

With all the enthusiasm and effort towards introducing programming to a younger population globally [1, 2, 6–8, 21], this paper presents empirical evidence to contribute to the ongoing discussion. We explore students' actual learning gain in the eventual transition from popular teaching environments such as block-based programming into text-based programming. Further, our goal is to advance research involving elementary school, which is not as popular as research involving university or even high school. Meerbaum-Salant et al [23] pose the question: should we make things "easy" for students during their initial studies or should we teach them the "right way" from the beginning? Our empirical evidence supports the claim that children with block-based experience develop programming abilities that make it easier to learn a text-based language. When starting from elementary school, the "easy" way could be a step towards the "right way".

ACKNOWLEDGMENTS

This work was partially funded by grants PICT-2014-1833, PDTS-CIN-CONICET-2015-172.

REFERENCES

[1] Carlos Areces, Luciana Benotti, Joshep Cortez-Sanchez, Raul Fervari, E. Garcia, , Cecilia Martinez, Martin Onetti, Eduardo Rodriguez-Pesce, and Nicolas. Wolovick. 2018. *Ciencias de la Computacion para el aula: 2do ciclo de Primaria* (1 ed.). Fundacion Sadosky: Program.ar.

[2] Michal Armoni, Orni Meerbaum-Salant, and Mordechai Ben-Ari. 2015. From scratch to real programming. *ACM Transactions on Computing Education (TOCE)* 14, 4 (2015), 25.

[3] Tim Bell, Jason Alexander, Isaac Freeman, and Mick Grimley. 2009. Computer science unplugged: School students doing real computing without computers. *The New Zealand Journal of Applied Computing and Information Technology* 13, 1 (2009), 20–29.

[4] Luciana Benotti, Federico Aloi, Franco Bulgarelli, and Marcos J Gomez. 2018. The Effect of a Web-based Coding Tool with Automatic Feedback on Students' Performance and Perceptions. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18)*. ACM, New York, NY, USA, 2–7.

[5] Luciana Benotti, Marcos J Gómez, and Cecilia Martínez. 2017. UNC++ Duino: A kit for learning to program robots in Python and C++ starting from blocks. In *Robotics in Education*. Springer, 181–192.

[6] Marina U. Bers. 2010. The Tangible Robotics Program: Applied Computational Thinking for Young Children. *Early Childhood Research & Practice* 12, 2 (2010).

[7] Douglas H Clements and Julie Sarama. 2002. Teaching with computers in early childhood education: Strategies and professional development. *Journal of Early Childhood Teacher Education* 23, 3 (2002), 215–226.

[8] K-12 Computer Science Framework Steering Committee et al. 2016. K-12 computer science framework. (2016).

[9] Stephen Cooper, Wanda Dann, and Randy Pausch. 2003. Teaching Objects-first in Introductory Computer Science. *SIGCSE Bull.* 35, 1 (Jan. 2003), 191–195.

[10] Valentina Dagiene, Linda Mannila, Timo Poranen, Lennart Rolandsson, and Par Söderhjelm. 2014. Students' Performance on Programming-related Tasks in an Informatics Contest in Finland, Sweden and Lithuania. In *Proceedings of the 2014 Conference on Innovation; Technology in Computer Science Education*. ACM, 153–158.

[11] Barbara Ericson, Mark Guzdial, and Maureen Biggers. 2007. Improving secondary CS education: progress and problems. In *ACM SIGCSE Bulletin*, Vol. 39. 298–301.

[12] Louise P Flannery and Marina Umaschi Bers. 2013. Let's Dance the "Robot Hokey-Pokey!" Children's Programming Approaches and Achievement throughout Early Cognitive Development. *Journal of Research on Technology in Education* 46, 1 (2013), 81–101.

[13] Louise P Flannery, Brian Silverman, Elizabeth R Kazakoff, Marina Umaschi Bers, Paula Bontá, and Mitchel Resnick. 2013. Designing ScratchJr: support for early childhood learning through computer programming. In *Proceedings of the 12th International Conference on Interaction Design and Children*. ACM, 1–10.

[14] A. Fowler, J. Pirker, I. Pollock, B. de Paula, M. Echeveste, and M. Gómez. 2016. Understanding the Benefits of Game Jams: Exploring the Potential for Engaging Young Learners in STEM. In *Proceedings of the 2016 ITiCSE Working Group Reports*. ACM, New York, USA, 119–135.

[15] Joanna Goode, Gail Chapman, and Jane Margolis. 2012. Beyond curriculum: the exploring computer science program. *ACM Inroads* 3, 2 (2012), 47–53.

[16] Brian Harvey, Daniel Garcia, Josh Paley, and Luke Segars. 2014. Snap!(build your own blocks). In *Proceedings of the 45th ACM technical symposium on Computer science education*. ACM, 749–749.

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

[18] Elizabeth R Kazakoff, Amanda Sullivan, and Marina U Bers. 2013. The Effect of a Classroom-Based Intensive Robotics and Programming Workshop on Sequencing Ability in Early Childhood. *Early Childhood Education Journal* 41, 4 (2013), 245–255.

[19] Pablo E Martínez López, Daniel Ciolek, Gabriela Arévalo, and Denise Pari. 2017. The GOBSTONES method for teaching computer programming. In *2017 XLIII Latin American Computer Conference (CLEI)*. 1–9.

[20] Stéphane Magnenat, Jiwon Shin, Fanny Riedo, Roland Siegwart, and Morderchai Ben-Ari. 2014. Teaching a Core CS Concept Through Robotics. In *Proceedings of the 2014 Conference on Innovation and Technology in Computer Science Education*. ACM, NY, USA, 315–320.

[21] Cecilia Martinez, Marcos Gomez, and Luciana Benotti. 2015. A Comparison of Preschool and Elementary School Children Learning Computer Science Concepts Through a Multilanguage Robot Programming Platform. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*. 159–164.

[22] Cecilia Martinez, Marcos Gomez, Marco Moresi, and Luciana Benotti. 2016. Lessons Learned on Computer Science Teachers Professional Development. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*. 77–82.

[23] Orni Meerbaum-Salant, Michal Armoni, and Mordechai Ben-Ari. 2011. Habits of programming in scratch. In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*. ACM, 168–172.

[24] Orni Meerbaum-Salant, Michal Armoni, and Mordechai Ben-Ari. 2013. Learning computer science concepts with scratch. *Computer Science Education* 23, 3 (2013), 239–264.

[25] Leonel Morgado, Ken Kahn, and Maria GB Cruz. 2010. Preschool cookbook of computer programming topics. *Australasian Journal of Educational Technology* 26, 3 (2010), 309–326.

[26] Dale Parsons and Patricia Haden. 2007. Programming osmosis: Knowledge transfer from imperative to visual programming environments. In *Procedings of The Twentieth Annual NACCQ Conference*. 209–215.

[27] Kris Powers, Stacey Ecott, and Leanne M. Hirshfield. 2007. Through the Looking Glass: Teaching CS0 with Alice. *SIGCSE Bull.* 39, 1 (March 2007), 213–217.

[28] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay S Silver, Brian Silverman, et al. 2009. Scratch: programming for all. *Commun. ACM* 52, 11 (2009), 60–67.

[29] David Thompson and Tim Bell. 2013. Adoption of new CS high school standards by New Zealand teachers. In *Proc of the Workshop in Primary and Secondary Computing Education*. ACM, 87–90.

[30] Ashok Kumar Veerasamy, Daryl D'Souza, and Mikko-Jussi Laakso. 2016. Identifying Novice Student Programming Misconceptions and Errors From Summative Assessments. *Journal of Educational Technology Systems* 45, 1 (2016), 50–73.

[31] David Weintrop and Uri Wilensky. 2015. Using Commutative Assessments to Compare Conceptual Understanding in Blocks-based and Text-based Programs. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*. 101–110.