

# Developing Computational Thinking in Early Ages: A Review of the code.org Platform

Rolando Barradas<sup>1,2,4</sup><sup>a</sup>, José Alberto Lencastre<sup>3</sup><sup>b</sup>, Salviano Soares<sup>1,4</sup><sup>c</sup> and António Valente<sup>1,2</sup><sup>d</sup>

<sup>1</sup>*School of Sciences and Technology, University of Trás-os-Montes and Alto Douro, Quinta de Prados, Vila Real, Portugal*

<sup>2</sup>*INESC TEC, Porto, Portugal*

<sup>3</sup>*CIEed - Research Centre on Education, Institute of Education, University of Minho, Campus de Gualtar, Braga, Portugal*

<sup>4</sup>*IEETA, UA Campus, Aveiro, Portugal*

**Keywords:** Computational Thinking, Early-age, code.org, Technology-enhanced\_learning.

**Abstract:** This article reports a pedagogical experience developed within the scope of a Ph.D. program in Electrical and Computer Engineering with application to Education. Starting with a contextualization on the evolution of computers and Computational Thinking, the article describes the platform used in this study - code.org -, highlighting the strengths that captivate the students. In the Case Study topic, we describe the study carried out, starting with a description of the students involved, followed by a description of the process and the analysis of the results, ending with the evaluation process performed by the students. The article ends concluding that code.org is a valid option to develop computational thinking at early-ages.

## 1 INTRODUCTION


Computational thinking is considered a key asset in the XXI century, as it allows to increase the analytical capacity of children in different areas of knowledge (Wing, 2006; Resnick, 2012) as well as to promote skills such as abstract, algorithmic, logical and scalable thinking (Resnick, 2012). These skills, associated with computer sciences, are transposed to other areas of knowledge and, as a consequence, to the daily lives of young people, making them more reflective and critical, thus better prepared for the world (Brennan & Resnick, 2012).


Within the scope of a Ph.D. program in Electrical and Computer Engineering with application to Education, to develop computational thinking at an early age (Coelho, Almeida, Almeida, Ledesma, Botelho & Abrantes, 2016) and using code.org, a well-known coding platform for children, we developed a pedagogical experience whose results are revealed in this article.


## 2 CONTEXTUALIZATION


Computers are everywhere, from our desktops to our pockets. The various physical forms in which we find them make it easy to use them daily in fields as diverse as industry, science, education, and entertainment. Today, most of us carry in our pockets, a smartphone with millions of times more computing power than NASA (National Aeronautics and Space Administration) had in 1969 when they placed a mission on the Moon (Puiu, 2019). From the construction of the first computer in 1946, the military-developed Electronic Numerical Integrator and Computer (ENIAC) to the present day, when the use of computers is a constant in our daily lives, the processing power of computers has continuously increased. Yet we rarely think about this evolution, since for most users today all this technology and processing power it's taken for granted as they've always lived surrounded by it.

The massification of computers has made them easily usable by anyone, regardless of age, educational background or professional occupation.

<sup>a</sup>  <https://orcid.org/0000-0001-9399-9981>

<sup>b</sup>  <https://orcid.org/0000-0002-7884-5957>

<sup>c</sup>  <https://orcid.org/0000-0001-5862-5706>

<sup>d</sup>  <https://orcid.org/0000-0002-5798-1298>

But at the same time has made it almost imperative for everyone to have, at least, basic computer skills, that include Internet and email, word processing, graphics and multimedia, and spreadsheets. Thanks to a wide range of intuitive applications and interfaces, computers can now be used without expert knowledge to solve complex problems or technical tasks as well as for the most varied situations of everyday life. However, as stated by Resnick (2012) and Resnick, Maloney, Monroy-Hernández, Rusk, and Astmon (2009), one of the biggest challenges that users face nowadays is to need to stop from being mere content consumers (programs, games, etc.) to become creators of such contents. To do so, one needs only to be aware of certain applications, sufficiently curious to search for information on the Internet or able to do so through a trial and error process.

Since the beginning of the millennium, ICT (Information and Communication Technologies) classes have become compulsory in schools. However, the study of ICT focuses on the transmission of knowledge about computer tools, software and hardware, and their use to solve everyday problems, such as writing a text, searching for information on the internet and learning how to communicate it efficiently. A natural evolution would be to make users understand how these tools work and how they are built.

Coding is a way of developing creative activities with children. Still, it also allows them to gain a broader view of computer uses, creatively solving real-world problems, by focusing primarily on design, planning, and implementation of a particular project.

In this context, it becomes necessary to mention an essential competence for the 21st century: computational thinking (Wing, 2007).

## 2.1 Computational Thinking

Computational thinking can be defined as the set of processes involved in formulating a problem and its solutions so that a computer (human or machine) can effectively solve it (Wing, 2017) and it's more connected to conceptualization than to coding itself (Wing, 2007).

The development of computational thinking promotes skills such as **(i) abstract thinking** - using different levels of abstraction to understand problems and solving them -, **(ii) algorithmic thinking** - the expression of solutions in different stages whose goal is to find the most effective way to solve a problem - **(iii) logical thinking** - formulating and excluding hypotheses - and **(iv) measurable thinking** -

breaking up a big problem into small parts or composing small parts to formulate a more complex solution (Resnick, 2012).

Brennan and Resnick's studies on computational thinking and the creation of interactive media products gave rise to a framework of reference for studying and evaluating the development of computational thinking that encompasses three dimensions: **(i) computational concepts**, **(ii) computational practices** and **(iii) computational perspectives**.

Regarding **(i), computational concepts**, Brennan and Resnick (2012) identified seven, namely:

**Sequences** – A set of steps or instructions that can be executed to complete a coding task, which is why it is important to define the correct order of execution of the commands since changing one of the commands could lead to completely different results;

**Loops** - These are mechanisms that allow you to execute the same sequence several times. When solving certain problems, it is possible to identify certain patterns of repetition;

**Events** - An event is a certain occurrence that causes a certain action to happen;

**Parallelism** - To solve certain problems several sequences may need to take place at the same time;

**Conditionals** - Conditionals allow a program to make decisions, through the use of decision structures;

**Operators** - They are used to express and solve mathematical and logical operations;

**Data** – Data structures are used to store, retrieve and update values stored in variables.

The **(ii) computational practices**, associated with the act of programming, are focused on the construction process, on thinking and learning, changing the focus from what is learned to how it is learned (Brennan & Resnick, 2012). Thus, four sets of practices were defined:

**Being Incremental and Iterative** - Action by which children develop, check whether a project works and continue to develop new approaches to the solution;

**Testing and Debugging** - It is through trial-error processes, analysis of previously made situations, that children check what does not work and correct errors;

**Reusing and Remixing** - You also learn when you build something using old projects or projects that others have already done;

**Abstracting and Modularizing** - The act of building something big by joining sets of smaller parts, since complex problems can be divided into smaller, simpler problems.

Regarding (iii) **computational perspectives**, Brennan and Resnick identified three major perspectives of children in their relation to computing, categorized as:

**Express** - computing is a means of creation and self-expression and allows students to start to see themselves as builders and not just consumers;

**Collaborate / Connect** - computing allows you to create with and for the others, making your creations an inspiration for your new projects or those of others, allowing the development of a critical spirit;

**Questioning** - Questioning technology with technology. Trying to understand how certain problems were solved can lead to questioning the functioning of other situations in the real world (Brennan, Chung & Hawson, 2011).

Therefore, it will be possible to evaluate the development of computational thinking in young people, analyzing the execution of projects/activities, designed taking into account the three dimensions defined.

Inevitably linked to computational thinking is the problem-solving method (Jonassen, 2011). Using coding as a way to develop computational thinking also allows you to stimulate students' creativity by solving real-world problems. According to Jonassen (2011), problem-solving skills development activities are the most relevant educational activities that students can perform because the knowledge built during the process is better understood and more easily retained.

Using the problem-solving method leads students to 'learn how to learn' (Papert, 1993) while looking for the solution of a problem instead of waiting for an answer given by the teacher, thus developing their domain of the procedures (Echeverría & Pozo, 1998). The use of this teaching method increases the motivation of students who become the main agent in the learning process.

### 3 THE code.org PLATFORM

Founded in 2013, code.org (Figure 1) is a nonprofit dedicated to expanding access to computer science in schools and increasing participation by women and underrepresented minorities.

The platform code.org maintains a very broad set of educative resources and tools that can be executed in almost all platforms, including smartphones and tablets which makes it very flexible and easy to use. Its vision is that every student in every school should

have the opportunity to learn computer science, just like biology, chemistry or algebra (Code.org, 2019).

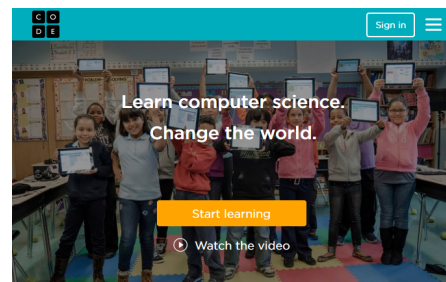


Figure 1: Code.org Platform Home Screen.

But what makes the platform so attractive to students? In our study we identified several details about the platform that we consider important to mention:

#### 3.1 The Easy Process of Accessing the Content

Access to the platform has been built to respond to almost any existing combination today. Students can start testing the platform without creating an account. More traditionally, students can use an account created on the platform. Additionally, they can also use a pre-existing account from other platforms, such as Google, Facebook, and Microsoft, as can be seen in Figure 2.

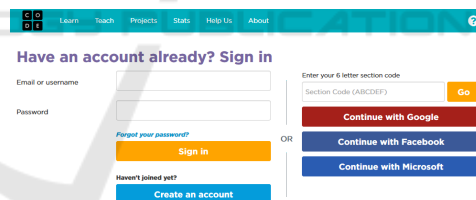


Figure 2: The various login possibilities on the code.org platform.

Any situation in which the student logs in with an account allows them to save their progress so that they can continue solving tasks, and also allows the teacher, through the creation of a class and assigning students, to monitor the learning process of each one individually.

Additionally, the fact that the platform is built as a web application that runs entirely in a browser environment means there is no need to pre-install or configure the devices where the students will work. The whole process of learning and development of computational thinking is done through a graphical environment in which students build their algorithms

by dragging and dropping instruction blocks to solve each of the challenges (Figure 3).



Figure 3: Coding by blocks.

### 3.2 Course Organization and Self-efficacy Control Tools

The Computer Science Fundamentals, Course 2, the object of this study, is organized in 19 Lessons, both online and offline, organized by logical order and with increasing difficulty of contents. Note that for this study, only online lessons finishing with Lesson 16 - Flappy Bird - were used, as students had previously taken offline coding classes (CS Education Research Group, 2015). Every lesson starts with an introductory video that explains the lesson's objectives and gives students some insight into the type of problems they will encounter. Starting with simple sequences and covering all the computational concepts identified by Brennan and Resnick (2012), the course ends with slightly more complex concepts such as nested loops and functions.

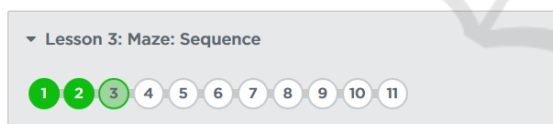


Figure 4: Viewing progress for self-efficacy control.

While performing a coding task, students are given the maximum number of blocks that should be used to find the optimal solution in each of the problems. If a student can solve a particular problem using the optimal solution, the problem number will be filled in green, on their progress overview. If they do solve it but not with the optimal solution, the problem number is filled with a light green color, as seen in Figure 4, so that the student can return to that program at a later time and find the optimal solution.

This course overview allows students to have a tool for monitoring their effectiveness, allowing them to compare with their peers and serving as motivation for task solving.

### 3.3 Known Characters

All challenges present in code.org were created using animations, sounds, and characters according to the age group of the target audience.



Figure 5: Example of a challenge involving familiar student characters.

The fact that the platform uses, in many of its challenges, familiar themes and characters such as Angry Birds and Plants vs. Zombies, pictured in Figure 5, makes the challenges even more attractive to students.

### 3.4 Instant Feedback

Kapp, Blair, and Mesch (2012) talking about gamified instruction scenarios, stated that the instant feedback was one of the most important elements in a Gamification system.

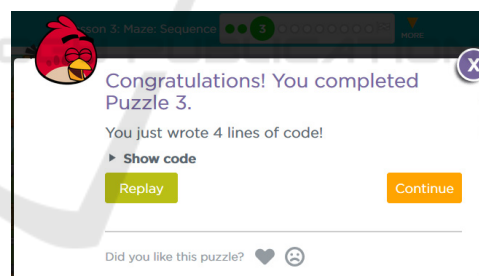


Figure 6: Instant Feedback Associated with Positive Reinforcement.

The code.org platform has interpreted these indications very well by designing a gamified system that maximizes feedback, notifying the student at all times about what they have already completed and what mistakes they have made (as seen earlier in the self-efficacy control tools). In addition to this task completion message, the system provides immediate corrective feedback. When students are successful in their task they are immediately rewarded with messages and sounds that indicate joy and success, as can be seen in the screen shown in Figure 6.

On the other hand, when something does not go so well, the student is informed immediately with a message of encouragement, as seen in Figure 7.

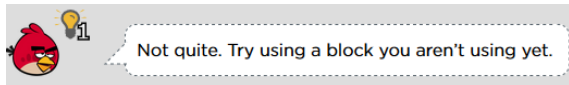


Figure 7: Instant feedback associated with non-success.

This instant feedback allows students to immediately view their progress and compare it with others.

### 3.5 Reward System

In addition to the already mentioned elements of motivation, and following the same principles of Gamification and reward system set out by Kapp, Blair, and Mesch (2012), code.org implemented a feature that allows students to print a course certificate, similar to the one shown in Figure 8, when they complete all Lessons.



Figure 8: Example of a certificate issued by the platform.

## 4 CASE STUDY

The study group consisted of 130 students from five 4th grade classes (9 and 10 years old) over two school years, divided as follows: 2017/2018, 28 students from Class 1 and 28 students from Class 2; 2018/2019, 25 students from Class 3, 26 students from Class 4 and 23 students from Class 5, out of a total of 130 students. Of those, 62 were females and 68 were males.

In the academic year 2017/2018, the classes were taught in a pedagogical pair regime, with the ICT teacher always assisted by the headteacher of the classes. However, it was not possible to maintain this organization of the classes in the 2018/2019 academic year, so they were taught only by the ICT teacher.

Initially, the basic concepts of the code.org platform were taught to all students. - i.e., how the

blocks fit together, workspace locations, interface details such as the action stage and execution buttons, as well as the use of platform access credentials. Subsequently, computational thinking was developed through hands-on laboratory problem-solving exercises (Jonassen, 2004) involving sequences, loops, parallelism, events, conditionals, operators, and data, which would allow students to create their first Flappy game.

Data on the execution of these tasks were collected through the platform's automatic records for statistical processing. Since the goal set for the course was for students to create their Flappy Bird, for this statistical treatment we only considered the online exercises from the first 16 lessons of code.org 's Computer Science Fundamentals, Course 2. Also used as instruments for data collection were the descriptive syntheses of the students elaborated by the teachers of the classes, an online questionnaire for the students and the notes of the ICT teacher on how the classes worked.

### 4.1 Results

The results of the 115 different problems were evaluated for each of the 130 students involved in the study, in a total of 14950 problems. The overall results are summarized in Table 1.

Table 1: Overall Test Results.

	No. of different problems/total analyzed	Completion Rate	Completion Rate with non-optimal solution
Global Results	115/14950	85.8%	3.7%

In this first analysis, we found that the percentage of problems solved is 85.8%, and only 3.7% of them did not get the optimal solution to the problem, which is a very positive result.

Analyzing the results according to the three dimensions of Brennan and Resnick's (2012) framework, **(i) computational concepts**, **(ii) computational practices** and **(iii) computational perspectives**, we obtained the following results:

#### (i) Computational Concepts

It is important to note that the number of problems indicated in Table 2 refers to the number of exercises in which each of the concepts was approached and that the same problem could address more than one concept. It is also possible to observe a disproportion between the number of problems that work some

Table 2: Test results, grouped by computational concepts.

Concept	No. of different problems/total analyzed	Completion Rate	Completion Rate with non-optimal solution
Sequences	46/5980	89.0%	2, 0 %
Loops	67/8710	86.5%	5.5%
Events	10/1300	69.7%	0%
Parallelism	10/1300	69.7%	0%
Conditionals	15/1950	72.8%	2.8%
Operators	25/3250	71.6%	1.7%
Data	10/1300	69.7%	0%

concepts. However, this fact was already expected since this was an initiation course so more complex concepts like Events, Parallelism and Data were only addressed in the final part of the course. The understanding of the Sequences was the skill that students had less difficulty to acquire, with a completion rate of 89.0%. Of these, only 2.0% of the results were not an optimal solution. On the opposite side, the Events, Parallelism and Data skills were those in which students had the most difficulties. Despite this, the completion rate is quite positive, at 69.7%, of which 100% reached the optimal solution.

### (ii) Computational Practices

All computational practices mentioned by Brennan and Resnick (2012) were addressed while solving the proposed problems, although, due to the type of problems present in the course, not all computational practices were given equal emphasis. Also, in this case, it is important to note that the number of problems indicated in Table 3 refers to the number of exercises in which each practice was addressed and that the same problem could address more than one practice.

Table 3: Test results, grouped by computational practices.

Practice	No. of different problems/total analyzed	Completion Rate	Completion Rate with non-optimal solution
Being incremental and iterative	82 /10660	89.7%	4.0%
Testing and debugging	23 /2990	78.9%	3.9%
Reusing and remixing	67 /8710	86.5%	5.5%
Abstracting and modularizing	10 /1300	69.7%	0%

Analyzing the results grouped by computational practices, it is possible to observe that the practices of *Being incremental and iterative* were the most addressed throughout the course with 82 different problems. It was also in these practices that students showed less difficulty with a completion rate of 89.7%. The students also demonstrated to be comfortable with the practices that involved *Reusing and Remixing*. In these problems, they managed to obtain a completion rate of 86.5%, with only 5.5% of these not reaching the optimal solution.

The practices that involved *Abstracting and modularizing* were those in which students obtained fewer good results. However, it was not possible to assess whether this is due to actual abstraction difficulties or whether, on the other hand, it is simply because the tasks dealing with this practice were the most complex exercises, therefore, the last to be solved, and, given the slow pace of some students, there was no time to solve them.

### (iii) Computational Perspectives

The three computational perspectives - *Express*, *Collaborate*, and *Question* - were cross-sectional throughout the process, although they were not objectively measured. Although the students developed the problem-solving tasks following the guidelines, in some tasks the freedom to create something new and/or personalize the existing one - *Express* - was implicit through the inclusion of personal elements in the provided scenarios. Although the work was mostly done individually, typically as soon as a lesson ended, the students endeavored to help their most delayed colleagues by doing peer work - *Collaborate*. Also, curiosity about the processes and the different problem-solving methods led them to *Question* the technology and to want to make new developments in the existing challenges - or games, as they called it.

## 4.2 Flappy Bird Challenge

We defined as a final objective for the course a Lesson in which, students had to build a Flappy game.

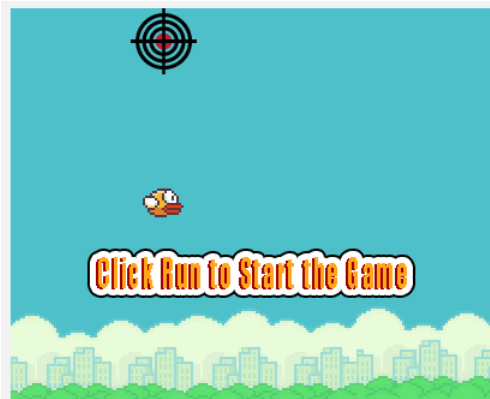


Figure 9: Flappy Bird Lesson.

Overall, the results were very positive, with a 63.9% completion rate of the challenge. Starting with the scenario represented in Figure 9, students created their personalized Flappy Bird, as shown in Figure 10.

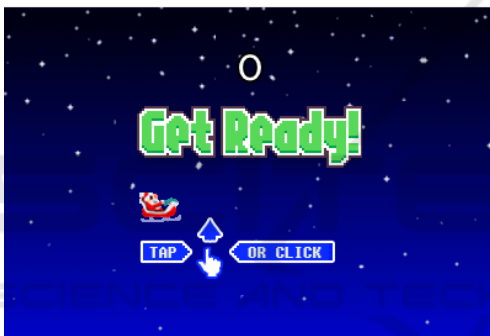


Figure 10: Student-Customized Flappy Bird Examples.

Students were encouraged to share their games with family and friends, via email, by sending a link that could be executed on all platforms where code.org works. In statistical terms, all students who managed to complete the challenge did so with optimal solutions. However, regarding the data analysis, this fact was not considered of much relevance, since, in the case of a work of creation, personalization, and sharing, almost all the solutions presented by the students could be considered optimal.

### 4.3 Daily Classes

As already mentioned in the description of the study, in the academic year of 2017/2018 the coding classes were held in a pedagogical pair regime, in which the ICT teacher was always assisted by the main teacher of each class. This was because the students were very young and it was the first time that they worked with computers and platforms such as code.org.

However, in the academic year of 2018/2019, it was not possible to maintain this organization. Although the overall results of the study were excellent, given this pedagogical change from one academic year to the next, we found it relevant to observe the partial results per class / academic year and then try to draw some conclusions.

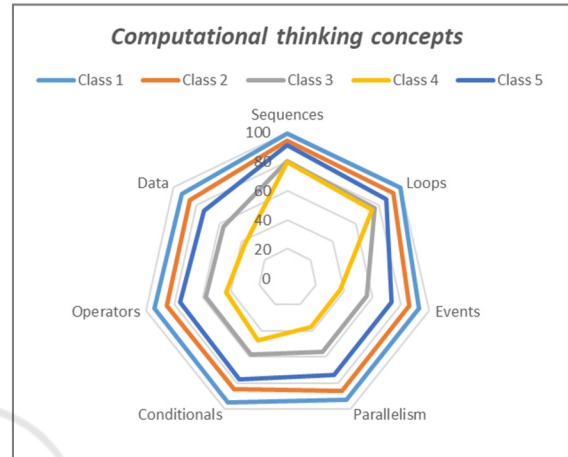


Figure 11: Comparative chart of tasks performed by computing concept.

Looking at Figures 11 and 12 - for a better reading of the graph, viewing from left to center, the lines sequentially represent the classes Class 1, Class 2, Class 5, Class 3 and Class 4 - it can be seen that, compared to the Classes 1 and 2, the lines referring to Classes 3, 4 and 5 are almost always closer to the center of the chart, moving away from the optimum results.

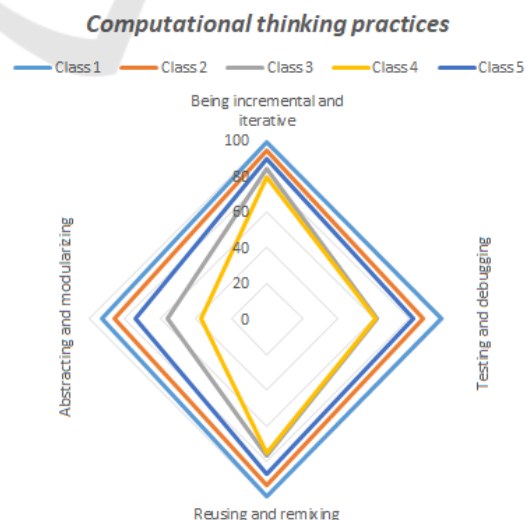


Figure 12: Comparative chart of tasks performed by computational practices.

Objectively, and comparing some results of task completion rates, in the academic year of 2017/2018, Classes 1 and 2, only 14.28% of the students did not complete the last challenge of the programmed course. On the other hand, in the academic year of 2018/2019, this number rose to 55.4%. Although the students were not the same, pedagogically speaking, this was the only variable changed throughout the study, so it should not be underestimated the importance of working in a pedagogical pair regime in this type of subject given the number of students in the classroom and their age.

#### 4.4 Evaluation by Students

To evaluate the work developed from the students' voices, we adapted a questionnaire based on the previous studies by Kalelioğlu (2015), a questionnaire made available to students in an anonymous online form, with some free text questions, allowing students to express their feelings about the developed work. The questionnaire contained the following questions:

**Question 1.** Did you have difficulties with the process of solving the problems? If you answered Yes or Some, say what were the difficulties.

**Question 2.** Was it possible to solve the various problems in several different ways? Did you try to do it?

**Question 3.** Do you think the code.org platform is easy to use?

**Question 4.** Did you have any problems with the platform?

**Question 5.** What was your favorite Lesson of code.org course 2?

**Question 6.** And what Lesson did you like least in code.org course 2?

**Question 7.** What do you think were the benefits (what did you learn) of coding using code.org?

**Question 8.** Do you like coding?

**Question 9.** What would you like to learn more about coding?

**Question 10.** Would you like to improve your knowledge?

After the end of classes for this study, students were invited to answer the online questionnaire. 76 valid responses were collected and 2 were canceled due to repetition of the answer and inconsistent data in the responses, such as "I had difficulties" but "I had no problems".

Analyzing the obtained answers, it was possible to retain the following conclusions:

When asked about **(1) difficulties in the problem-solving process**, 23 students reported that they had no problem in the problem-solving process. Only 5 students considered that they had problems. 48 students reported that they had some problems with this process.

However, when asked to refer the actual difficulties, we observed that only 3 of the 76 students reported real problems with the process and the platform - "I had difficulties in executing it"; "Had to reset to drag the blocks"; "At the beginning, I had some difficulties dragging the blocks". All the remaining responses referred to problems that are in no way related to the resolution process, but rather to some issues of previous knowledge, namely, notions of laterality ["turning to the left", "I had difficulties in those problems that asked to turn right or turn left"], mathematical questions ["knowing the angles", "I had difficulty finding the angles and number of pixels needed", "was calculating the pixels and degrees."], or even problems interpreting the problems ["Sometimes it was more difficult to pass the level because I didn't quite understand how it was supposed to be done", "I had some difficulties in understanding some exercises"]. Interpretation issues could be related to the previously mentioned fact concerning some of the incomplete or partially wrong translations to the local language.

Regarding the fact that **(2) it is possible to solve the proposed problems in several different ways**, 86.8% of the students reported that they tried to do it, but only 84.2% of those reported that they managed to do it in several different ways. This fact was also observed in the analysis of the results provided by the platform, where 6.9% of the problems were solved with non-optimal solutions, compared to the total number of exercises solved. This fact shows that despite the students realized that it would be possible to solve the problems in another way, as they appeared signaled in a different color, not all students were able to do so, to obtain the optimal solution.

Regarding the **(3) ease of use of the code.org platform**, 78.9% of the students who answered the survey found the platform to be easy to use ["Code.org is very easy to use and I had no problem with the platform. It is very easy, it is practical because it is suitable for all electronic devices and to enter it you just have to enter the password and enter, and you have a solution if you forget your password", "Yes I think it is easy to use code.org platform"]. Interestingly, some of the students who considered that the platform is not easy to use, consider that they had no problems in its use because when asked if they



had had **(4) problems with the platform**, 81.6% of the students reported that they did not have problems.

When referring to their **(5) favorite Lesson of the course**, the students elected first place, with 16 votes, Lesson 3, MAZE: SEQUENCE, the first they worked on, tied with Lesson 16, FLAPPY, the last considered for this study. Lessons 7: ARTIST LOOPS, 8: BEE LOOPS and 13: BEE: CONDITIONALS, were also the ones that most pleased the students, with 8, 10 and 7 votes respectively. Although many students chose as favorite the Lessons that focused on simpler concepts, 63.2% of the students who answered the survey chose as a favorite Lesson one of the ones that already involved more complex computing concepts such as *Loops, Events, Parallelism, Conditionals, Operators, and Data*. This result is in line with the general results of the study, and it demonstrates that students appreciated the complexity brought to the problems by the application of computational concepts with a higher level of difficulty. They enjoyed using their recently developed computational thinking.

On the other hand, when they referred to the **(6) Lesson that they liked least**, students who had preferred lessons with more elaborated concepts tended to like less the first lessons of the course with only basic concepts. The opposite was also noted, with students who preferred Lesson 3 saying they did not like the more complex lessons.

But it was when they were asked about their **(7) learning through the code.org platform** that students gave the most diverse answers. After analyzing the content and after a first floating reading (BARDIN, 1979), the main categories that emerged were included in Table 4.

Analyzing the collected data, it was possible to verify that, in general, students learned “that one should never give up and that we must have patience”, in a clear reference to the development of autonomy and resilience inherent in solving problems. Also, the fact that they learned “to think better”, “to solve problems in another way” and alone, “to do things in different ways and with creativity” shows the development of problem-solving skills.

In terms of acquired knowledge, students generally refer to the fact that “learning to work better on the computer and understanding the computer codes”, although they also use the term “Programming” a significant number of times. It is also important to mention the fact that some students consider that they had learned basic notions of laterality and Mathematics while solving exercises that involved

Table 4: Excerpt from the Free Text Responses.

Category	Evidence (examples)	Frequency
Programming	“I learned to program more or less well.”; I learned “how to program games.”; “The benefits were learning to program.”; “I learned to program very well.”; “I learned how to create some things”	15
Autonomy	“I learned to be more patient and not ask for help right away.”; “I learned to solve problems on my own”; “I with code.org learned that we can never give up.”; “I learned to work alone.”; “I learned that one should never give up and that we have to be patient”	13
Learn to work with computers	“Learn how to work with computers”; “I learned to work better on the computer”	12
Learn Math	“I was also able to look at a certain angle and identify it”; “I learned a lot of math.”; “I learned to use angles”	10
Play	“I learned to play code.org games.”; “play and at the same time learn.”; “I learned to play games on the site.”; I learned “to work, to play and to have fun.”	8
Think faster and better to solve problems	“At code.org I was able to: (...) think faster to execute problems”; “I learned to solve problems”; “By coding at code.org: I learned to think better (...).”; “I learned to use the brain more easily.”; “I was amazed to do things in different ways and with creativity”	9
Notions of Laterality	“I learned right and left”	4

mathematical concepts such as angles and pixel counts since it was important not only to learn to program but also to learn others contents while programming. Also important is that the students are learning while having fun since, as they say, it was ‘play and learn at the same time’.

To the question **(8) Do you like to program?**, we obtained 66 positive responses (88.2%) confirming that students were enjoying programming, at least in the visual form they have known so far. We also obtained 8 responses from students (10.5%) who said they like to program, but only certain types of

problems. As for negative responses, we obtained only one.

As for **(9) improving/deepening knowledge**, students tended to answer that they would like to learn more about two different topics: building and coding robots and coding games. Content analysis was also performed to this free-text answers (Bardin, 1979) and the main categories found were noted and replicated in Table 5.

Table 5: Excerpt of categories for the content analysis of the answers on deepening knowledge.

Category	Evidence (examples)	Frequency
Coding Robots	“In terms of coding I liked to control robots and many more things about coding and would like to improve my knowledge.”; “I would like to know if the robots are programmed the same way we program the code.org characters and yes I would like to improve my knowledge.”; “I would like to assemble robots, invent a game to work in tablets, use computers, etc.”	25
Coding Games	“I would like to learn everything that I have to learn and I would like to improve my knowledge.”; “I would like to make other games”; “Program games like this.”; “Make games. Yes, I would like to know more.”; “I would like to invent my games”; “I would like to know how to program my own game”	13

Regarding **(10) learning more/improving knowledge**, we obtained 97.4% of affirmative answers, with phrases similar to “yes I would like to know more”, “I would love to learn more” or “Yes, I would like to”. As for negative responses, we obtained only one - from the same student who said he does not like to program - and another student stated that he would like to learn “more or less”.

## 5 CONCLUSIONS

We call Computational Thinking to the ability to formulate a problem and find a solution, whether executed by a computer or not (Cuny, Snyder & Wing, 2010). To assess the development of computational thinking, Brennan and Resnick (2012) created a framework of references that identifies concepts, practices and computational perspectives. This study used the lessons provided by the code.org platform to analyze the development of computational thinking at an early age. As all the tasks proposed to the students included the search for the solution of a problem, concepts, practices and computational perspectives it is possible to say that computational thinking was promoted. Students achieved very positive results, at the same time they trained problem-solving skills, building and retaining knowledge better (Jonassen, 2011).

The main conclusion from this pedagogical experience is that the **code.org platform is a valid option to develop computational thinking at an early age and a good way for students to start solving real-life problems by stimulating the capacity of abstraction using simulated and experienced practice**. We believe that this pedagogical experience will provide these children with essential skills for increasingly complex life in the 21st century, which includes creativity and innovation, critical thinking and problem solving, communication and collaboration (Partnership for 21st Century Skills, 2009).

Code.org is designed very carefully, and is suitable for children, making it a very useful tool for the introductory coding classes. The use of some of the gamification strategies, such as narratives, trophies and instant feedback, works as an involvement factor for students. Also, the fact of having clues for solving problems, the possibility of partially solving the exercises and later being able to return to complete them at 100%, makes code.org a flexible and appropriate tool for the age group under study. It was also possible to observe that students felt involved in recognizing some of the characters used in the challenges and that they were learning computer science concepts while having fun.

There are, however, other conclusions to be taken from the experience, namely concerning the importance of the organization of this type of class in a pedagogical pair regime or in smaller groups of students, since the proposed tasks require very close monitoring, which becomes extremely difficult to achieve by a single teacher.

Regarding its limitations, this study results are limited by the number of students involved and also by the size of the classes, which may have affected the statistical results. Additionally, the study is also limited by its duration, since, taking advantage of more time, it would have been possible to transmit the students a deeper knowledge about the fundamentals of coding and to evolve to more complex scenarios.

## 5.1 Final Reflection

Every student in every school should have the opportunity to learn how to program. The existence of these initiatives can help to bridge the technical gap of human resources in the IT area, developing children's skills and potential earlier, through carefully designed scenarios that allow them to understand the principles of operation of computers and their software. Rather than trying to teach a specific coding language, the primary purpose of early-age coding classes is to provide students with problem-solving skills. In this learning process, children learn many other things. They are not simply learning to program, they are coding to learn (Resnick, 2013).

Such experiences demonstrate the importance that the use of this type of platform has for children. The early introduction of coding activities in the curriculum is essential as it contributes to the development of children by turning them into producers and not simply consumers of content and technology.

## ACKNOWLEDGMENTS

This work was partially financed by the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia, through national funds, and co-funded by the FEDER, where applicable.

This work was partially funded by CIEd – Research Centre on Education, project UID/CED/01661/2019, Institute of Education, University of Minho, through national funds of FCT/MCTES-PT.

We would like to thank the Colégio Paulo VI (Gondomar, Portugal) the authorization to carry out this study on its premises, and students of the 4th grade of the academic years of 2017/2018 and 2018/2019 by their collaboration.

## REFERENCES

- Bardin, L. (1979). *Análise de conteúdo*. Lisboa: Edições 70.
- Barradas, R., & Lencastre, J. A. (2017). Gamification e Game-Based Learning: Estratégias eficazes para promover a competitividade positiva nos processos de ensino e de aprendizagem. In *Revista Investigar em Educação* (Issue Mundo digital e Educação, pp. 11–37). Sociedade Portuguesa de Ciências da Educação.
- Barradas, R.; Lencastre, J. A.; Soares, S.; Valente, A. (2019). Usability Evaluation of an Educational Robot for STEM Areas. 11th International Conference on Computer Supported Education. Heraklion: INSTICC and University of Crete.
- Bell, T., Witten, I. H., & Fellows, M. (2015). *CS Unplugged*. University of Canterbury, NZ. [http://csunplugged.org/wp-content/uploads/2015/03/CSUnplugged\\_OS\\_2015\\_v3.1.pdf](http://csunplugged.org/wp-content/uploads/2015/03/CSUnplugged_OS_2015_v3.1.pdf)
- Brennan, K., Chung, M., & Hawson, J. (2011). *Scratch Curriculum Guide Draft*. *Nature*, 341(6241), 73.
- Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. Annual American Educational Research Association Meeting, Vancouver, BC, Canada, 1–25. <https://doi.org/10.1.1.296.6602>
- Code.org (2019). About | code.org. Retrieved April 2019, from <https://code.org/about>
- Coelho, A.; Almeida, C.; Almeida, C.; Ledesma, F.; Botelho, L.; Abrantes, P. (2016). *Iniciação à Programação no 1.º Ciclo do Ensino Básico, Linhas Orientadoras para a Robótica*. Lisboa: DGE. Retrieved September 17, 2017, from [http://www.erte.dge.mec.pt/sites/default/files/linhas\\_orientadoras\\_para\\_a\\_robotica.pdf](http://www.erte.dge.mec.pt/sites/default/files/linhas_orientadoras_para_a_robotica.pdf)
- Costermans, J. (2001). *As actividades cognitivas - raciocínio, decisão e resolução de problemas*. Coimbra: Quarteto Editora.
- CS Education Research Group (2016). *Computer Science Unplugged*. Retrieved September, 2015, from <http://csunplugged.org>
- Cuny, J.; Snyder, L.; Wing, J.M. (2010). Demystifying computational thinking for non-computer scientists. Retrieved from <http://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf>
- Echeverría, M.; Pozo, J. (1998). Aprender a resolver problemas e resolver problemas para aprender. In J. Pozo, *A solução de problemas: aprender a resolver, resolver para aprender*. Porto Alegre: Artmed.
- Eguchi, A. (2014). Robotics as a Learning Tool for Educational Transformation. International Workshop Teaching Robotics, Teaching with Robotics & International Conference Robotics in Education, 27–34. <https://doi.org/10.4018/978-1-4666-8363-1.ch002>
- Figueiredo, M., Torres, J. (2015). *Iniciação à Programação no 1.º Ciclo do Ensino Básico Linhas Orientadoras*. Retrieved October 1, 2016, from [http://www.erte.dge.mec.pt/sites/default/files/Projetos/Programacao/IP1CEB/linhas\\_orientadoras.pdf](http://www.erte.dge.mec.pt/sites/default/files/Projetos/Programacao/IP1CEB/linhas_orientadoras.pdf)
- Jonassen, D. (2004). *Learning to solve problems - an instructional design guide*. São Francisco: Pfeiffer

- Jonassen, D. (2011). Learning to solve problems. A Handbook for Designing Problem-Solving Learning Environments. New York: Routledge
- Brennan, K., Chung, M., & Hawson, J. (2011). Scratch Curriculum Guide Draft. *Nature*, 341(6241), 73.
- National Research Council. (2010). Report of a Workshop on The Scope and Nature of Computational Thinking. Washington: The Nacional Academies Press.
- Nunes, D. (2011). Ciência da Computação na Educação Básica. *Jornal da Ciência*. Retrieved from <http://www.jornaldaciencia.org.br/Detaile.php?id=79207>
- Papert, S. (1993). *The Children Machine*. New York: BasicBooks
- Partnership For 21ST Century Skills. (2009). Framework for 21st Century Learning. Retrieved from [http://www.p21.org/storage/documents/docs/P21\\_framework\\_0816.pdf](http://www.p21.org/storage/documents/docs/P21_framework_0816.pdf)
- Pedro, A.; Matos, J. F.; Piedade, J.; Dorotea, N. (2017). Probótica: Linhas Orientadoras. Lisboa: DGE. Retrieved from [http://www.erte.dge.mec.pt/sites/default/files/probotica\\_-\\_linhas\\_orientadoras\\_2017.pdf](http://www.erte.dge.mec.pt/sites/default/files/probotica_-_linhas_orientadoras_2017.pdf)
- Pólya, G. (2003). *Como resolver problemas*. Lisboa: Gradiva
- Kalelioğlu, F. (2015). A new way of teaching programming skills to K-12 students: Code.org. *Computers in Human Behavior*, 52, 200-210
- Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J. a Y., Silverman, B., & Kafai, Y. (2009). Scratch: Programming for All. *Communications of the ACM*, 52, 60–67. <https://doi.org/10.1145/1592761.1592779>
- Resnick, M. (2012). Point of View: Reviving Papert's Dream. *Educational Technology*, 52(4), 42–46.
- Resnick, M. (2013). Learn to Code, Code to Learn. Retrieved February 7, 2019, from <https://www.edsurge.com/news/2013-05-08-learn-to-code-code-to-learn>
- School, C. A. (2012). A curriculum framework for Computer Science and Information Technology. Retrieved from <http://www.computingschool.org.uk/data/uploads/Curriculum%20Framework%20for%20CS%20and%20IT.pdf>
- Stahl, G.; Koschmann, T.; Suthers, D. (2006). Computer-supported collaborative learning: An historical perspective. In R. K. Sawyer, *Cambridge handbook of the learning sciences* (pp. 409-426). Cambridge, UK: Cambridge University Press.
- Wing, J. M. (2006). Computational Thinking. *Communications of the ACM*, 49, pp. 33-35.
- Wing, J. M. (2007). Computational Thinking. Retrieved on May 1, 2019, from [https://www.cs.cmu.edu/afs/cs/usr/wing/www/Computational\\_Thinking.pdf](https://www.cs.cmu.edu/afs/cs/usr/wing/www/Computational_Thinking.pdf)
- Wing, J. (2017). Computational Thinking's Influence on Research and Education for All. *Italian Journal of Educational Technology*, 25(2), 1–12. <https://doi.org/10.17471/2499-4324/922>
- Puiu, T., (2019). Your smartphone is millions of times more powerful than all of NASA's combined computing in 1969, Retrieved on February 21, 2019, from <https://www.zmescience.com/research/technology/smartphone-power-compared-to-apollo-432/>
- Kapp, K. M., Blair, L., & Mesch, R. (2012). *The Gamification of Learning and Instruction Fieldbook*. San Francisco, CA: Wiley