RECyT
Year 25 / Nº 39 / 2023 / 82–90
DOI: https://doi.org/10.36995/j.recyt.2023.39.010

Computational thinking: an analysis through structured programming using Scratch

Pensamiento computacional: un análisis a través de la programación estructurada mediante Scratch

Antonieta Kuz[1, *]

1- Universidad Metropolitana para la Educación y el Trabajo (UMET). Sarmiento 2037, C1044 AAE, Buenos Aires. Argentina.
*E-mail: zukanto@hotmail.com

Received: 05/04/2022; Accepted: 19/08/2022

Abstract

In recent years, numerous initiatives have emerged to develop computational thinking. Computational thinking and programming are closely related they are both tools for working with algorithmic concepts. ICTs and, and specifically computer programs with a playful orientation for teaching programming are relevant since they take into consideration aspects related to the educational environment. Game-based learning is a complement that allows taking advantage of the playful component of games to train computational thinking and, therefore, various others skills. Scratch is one of the most used tools to teach programming, through visual and playful programming languages that seek to promote computational skills that involve problem solving, through active and constructive learning. In this study, theoretical foundations of structured programming are analyzed based on simple computing concepts such as handling sequences, control instructions such as loops and conditionals, and their adaptation using Scratch. For this article, a qualitative analysis is presented, supported by descriptive research. The partial findings, at this point, suggest the usefulness of applying video games to train computational thinking skills. In this way, opens the possibility of proposing extended uses of other interactive games such as Lightbot, PilasEngine, Pilas Bloques.

Key words: Computational Thinking; Computer Programming; Videogame; Virtual Learning; Information Technology; Programming Language.

Resumen

En los últimos años, han surgido numerosas iniciativas para desarrollar el pensamiento computacional. El pensamiento computacional y la programación están estrechamente vinculados dado que ambos son un medio para trabajar con conceptos algorítmicos. Las TICs y en particular los programas computacionales con orientación lúdica para la enseñanza de la programación son relevantes dado que tienen cuenta aspectos vinculados al entorno educativo. El aprendizaje basado en juegos es un complemento que permite aprovechar el componente lúdico de los juegos para formar el pensamiento computacional y, por lo tanto, diversas habilidades. Scratch es una de las herramientas más utilizadas para enseñar programación, mediante lenguajes de programación visuales y lúdicos que busca promover habilidades computacionales que involucran la resolución de problemas, a través del aprendizaje activo y constructivo. En este estudio se analizan los fundamentos teóricos de la programación estructurada en función de conceptos informáticos simples como el manejo de secuencias, instrucciones de control como bucles y condicionales, y su adecuación mediante Scratch. Para este artículo, se presenta un análisis cualitativo, sustentado en una investigación descriptiva. Los hallazgos parciales aquí sugieren la pertinencia del uso de videojuegos en el marco de la formación del pensamiento computacional. Se abre la posibilidad de pensar en usos extendidos a otros juegos interactivos como Lightbot, PilasEngine, Pilas Bloques.

Palabras clave: Pensamiento Computacional; Programación Informática; Videojuego; Aprendizaje Virtual; Tecnología de la Información; Lenguaje de Programación.

Introduction and theoretical framework

Currently learning process has changed with the incorporation of ICTs. In the information society, the technologies that facilitate the creation, distribution and manipulation of information besides different areas of knowledge currently work in a multidisciplinary way in the processing and analysis of information. A very important factor is the digitization of the economy and the transformation of work, which, on the one hand, promotes the emergence of new professions and the creation of new jobs and, on the other hand, leads to the modification of some professions or existing jobs by changing the methods, the way in which the tasks are carried out, the requirements of the job and, as a consequence, the needs necessary to carry it out. In recent years, computational thinking has been introduced into the curriculum of educational systems, due to a fundamental issue that involves the emergence of new basic skills. Computational thinking is part of the educational debate, as with other key skills, reading, writing and math skills and traditional key competencies (Soria Valencia & Rivero Panaqué, 2019). New technologies are an integral part of life and it is a certainty that they will play a leading role in your future as well. In a world increasingly governed by digital, programming is essential knowledge. Programming is one of the new challenges faced by teachers in the classrooms in the different sections of schooling, since programming does not simply mean coding in a programming language but also implies including computational thinking to solve problems, in the different subjects of the school trajectory. Understanding that programming is not a closed skill that only serves the purpose of creating code for a computer by a programmer or computer scientist, but rather serves to develop skills that allow the student to project new possibilities.

Within the spectrum of educational resources, a particular type is educational software whose didactic purpose is to teach programming and promote computational thinking in playful environments, through the generation of dynamic pedagogical environments. The teaching and learning processes supported by the use of Scratch promote and strengthen educational practices and enable interactivity in an exploratory and creative way through collaborative projects in different contexts. Computational thinking can be integrated into the curriculum through programming. Learning programming requires understanding abstract concepts that are not usually easy to assimilate due to the lack of concrete references that allow the student to be experienced. This is the reason why a wide variety of educational software has been developed with didactic purposes and whose main objective is to facilitate the learning of programming concepts, but which involve cognitive skills oriented to problem solving and Computational Thinking. (Morris et al 2017).

Many of these software are developed in predominantly visual environments with visual blocks, which is a widely used strategy to learn programming and develop computational thinking and the combination of graphics, animations, photos, music, etc. Knowing a detailed description of the Scratch tool is relevant, given that as a playful element it accompanies the diversity of initiatives related to the teaching and didactics of programming. For this reason, in this work we will make an analysis of structured programming through Scratch. The rest of the article is structured as follows: in section 2 we detail the theoretical framework and the aspects involved. In section 3 we detail the study case. In section 4 we detail the discussion and present the conclusions of this work.

Computational Thinking

Computer Science is an academic discipline with their own body of knowledge such as Computational Thinking. Wing (2006, 2010) defined Computational Thinking as follows:

The thought process involved in formulating a problem and its solutions in such a way that the solutions are represented in a way that they can be brought to an information processing agent.

Solve problems, design systems and understand human behavior, based on the fundamental concepts of computer science. Computational thinking includes a wide variety of mental tools that reflect the breadth of the field of computing ... [furthermore] it represents a universal attitude and skills that all individuals, not just computer scientists, should learn and use (2006, p. 33)

Phillips (2008) describes it as *"the integration of the power of human thought with the capabilities of computers"*.

Rojas López (2019) defines computational thinking as:

A type of analytical thinking, a set of cognitive and metacognitive strategies paired with processes, skills and computing methods (analysis, abstraction, decomposition, heuristic reasoning, planning, programming, model, pattern recognition, algorithm), its essence is to think about data and ideas, and to use and combine these resources to solve problems, design systems and understand human behavior, in such a way that a computer can carry out the solution effectively.

Computational Thinking is within the conceptual foundations of computing. The International Society for Technology in Education (ISTE) (Sykora, 2021) understands Computational Thinking as a problem solving process that includes (but is not limited to) the formulation of problems; data collection and analysis; representation of data through abstractions such as models and simulations; automated solutions through algorithmic

thinking; identification, analysis and implementation of possible solutions in order to achieve the most efficient and effective combination of resources and steps; and the generalization and transfer of this problem-solving process to a wide variety of contexts.

Currently, several authors indicate that the use of Computational Thinking is oriented to problem solving, and is strongly related to Computer Science, and specific skills.

Table 1: Conceptual frameworks of computational thinking. [Source: Adell et al. (2019)]

Concepts of Computational Thinking ISTE (2021)	Formulate computational solution problems Organize logically and analyze data Abstractions, including models and simulations Algorithmic thinking Efficiency evaluation and correction Generalization and transfer to other domains Supported by: provisions such as trust to deal with complexity, persistence in difficult problems, tolerance for ambiguity, open problems, communication and collaboration. Harness the power of technological methods to develop and test solutions, Collect data, Analyze data, Represent data, Decomposition, Abstraction Algorithms Automation, Testing, Parallelization, Simulation, Supported by: empowered learner, digital citizen, knowledge builder, designer, communicator, collaborator
New frameworks for studying and assessing the development of computational thinking (Brennan & Resnick, 2012)	Computational concepts: Sequences, Event Loops Conditional Parallelism, Operators, Data, Computational practices, Being incremental and iterative, Rehearse and debug, Reuse and remix, Abstract and modularize, Computational perspectives Express, Connect, Ask
Computing at School Concepts of Computational Thinking (Csizmadia et al., 2015)	Computing, Logical reasoning, Algorithmic thinking, Decomposition, Generalization, Patterns, Abstraction, Representation, Evaluation Supported by: reflection, coding, design, analysis and application techniques.
Concepts and processes with a high level of consensus among experts (Delphi1) (Rich & Langton, 2016)	Concepts Conditional logic, Efficiency, Hashing (summary functions), Iterators, Parallelization, Segmentation, Recursion, Loops, Variables, Functions, Matrices, Operators, Event management, Communication processes, Debugging, Group problem solving, Negotiation Data organization, Decomposition of problems.
Common Aspects in a Selected Literature Review (Corradini et al., 2017)	Mental processes: Algorithmic thinking Logical thinking, Problem decomposition, Abstraction, Pattern recognition, Generalization. Methods: Automation Data collection, analysis and representation Parallelization, Simulation Evaluation, Programming. Practical: Experiment, iterate, retouch Test and debug Reuse and mix Cross-cutting skills: Create, Communicate and collaborate, Reflect, learn, meta-reflection Tolerance of ambiguity

Learning programming requires understanding abstract concepts that are not usually easy to assimilate due to the lack of concrete references that allows the student to be experienced. Another relevant research on Computational Thinking is that carried out by Ortega Ruipérez & Asensio Brouard, (2021) in which they evaluate it from an approach oriented to the resolution of complex problems, and this is used as a problem-solving strategy. In their article, they validate the theoretical construct of an evaluation instrument to measure computational thinking resolution by solving complex problems. According to Zapata-Ros (2015), he considers that computational thinking deals with a new literacy, digital literacy, and as with other key skills: reading, writing and mathematical skills, it is required to teach it from the early stages of development. Recently, Zapata-Ros (2019) considers unplugged computational thinking a new term. Indicating unplugged computational thinking refers to a set of activities that are developed to promote skills in children that can be evoked later, to promote computational thinking. He also considers that it is important to promote learning programming progressively from a playful approach. Likewise, in the research of Balladares Burgos et al (2016) they propose a relationship between complex thinking and computational thinking through a reflection on education based on the conception of uncertainty of complex thinking and the connecting elements between a thought complex and computational thinking based on connectives and the challenges of a society 3.0.

Educational computer games

There is a wide range of educational software in recreational environments and, within them, we find those that have the purpose of teaching programming in educational environments. These programs are designed with the purpose of ensuring that students have simple programming platforms without becoming professional environments.

According to the report United Nations International Children's Emergency Fund (UNICEF) (2018), it indicates that play is an essential learning strategy since they affirm that "*play is one of the most important ways in which young children obtain essential knowledge and skills*". Both development and learning are holistic in nature, carrying them out in a school environment is a complex task; however, through play they can incentivize all areas of development, including motor, cognitive, social and emotional.

The digital game through educational and didactic software has been implemented as a resource in different fields of knowledge, giving rise to a large number of contributions of great variety. The implementation of digital games for the teaching of programming allows us to understand and see the game as a crucial factor for improving the quality of education, due to the results and the change in attitude observed in the student in the face of the frustration that it frequently generates learning the logic of programming. From the pedagogy of programming teaching, the game contributes to significantly improve its processes, to achieve the objectives set for an activity or program, and in the social construction of knowledge determined by aspects such as cognitive, affective and

1- Delphi, is a technique that seeks to reach a level of expert consensus

communicative. Digital games can have a positive impact on learning specifically in the areas of science, technology, engineering and mathematics. Moreover, games contribute to the development of critical thinking skills such as decision making and problem solving in motivating environments that let you try and experiment.

The pedagogical bases and theoretical principles of computational educational games is constructionism. Considering the idea that learning is a process where students are building it through knowledge and from the daily experiences of their environment continuously (Soleimani, 2019). Constructivism is also related to the maker philosophy, which consists of students building their own knowledge and being able to solve problems independently and autonomously, so that they can experiment, create, make mistakes and rectify, learning from the process and their mistakes (Gibbons & Snake-Beings, 2018)

Scratch

New technologies are an integral part of life and it is a certainty that they are occupying a leading place. In a world increasingly governed by digital, programming is essential knowledge. There are tools for learning programming and that contribute to the development of skills. Programming is one of the new challenges faced by teachers in the classrooms in the different sections of schooling, since programming does not simply mean coding in a programming language but also implies including Computational Thinking to solve problems (Lye & Koh, 2014), in the different subjects of the school trajectory. Many of these software are developed in predominantly visual environments with blocks, which is a widely used strategy to learn programming and develop Computational Thinking and the combination of graphics, animations, photos, music, etc.

Using a programming language, instructions can be performed and programmed to obtain results from a computer. Most programming languages are purely textual; that is, they use text sequences that include words, numbers, and punctuation marks, similar to written natural languages. Syntax is the visible part of a programming language and defines the set of rules that must be followed when writing the source code of programs to be considered correct. The syntax of a programming language is defined as the set of rules that must be followed when writing the source code of programs to be considered correct for that programming language. Learning to use the syntax presents certain difficulties until correctly handling a programming language, since there are languages that present a complex syntax. Given the difficulties in programming languages and taking up the constructionist ideas (Müller, 2020) embodied in the Logo language, the Lifelong Kindergarten research group of the MIT (Massachusetts Institute of Technology) media laboratory developed the platform for Scratch programming. This educational tool has great potential since one of its functions is that through proper pedagogical use, students develop creative skills, logical, deductive reasoning.

Scratch is a block-based programming language (Hu, et al. 2021), with a grammar and syntax such as the C language, Java or Python among others. Blocks have a visual grammar and their combination rules have the same role as syntax in text-based languages. Developers over time have made improvements to the tool and a large community has been generated to encourage collaboration among members as a source of ideas and improvements. Besides Scratch versions were evolving since 2007 to 2020 with the Scratch version 3.11.1.

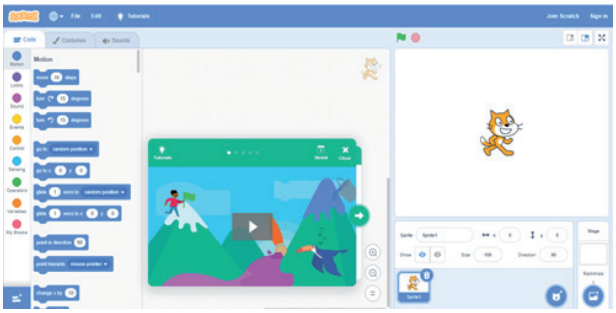


Figure 1: Screenshot of scratch.

The blocks represent instructions (see Figure 1) with imperative mode texts in several languages and include control structures, event handling, use of variables, messages between objects, movement and sound. It also has a library of scenes and objects to which material can be added from the other platforms.

Methodology

Through this case study, we seek to establish a relationship between Scratch programming and Computational Thinking through syntactic analysis from the structured programming paradigm. This research is based on a qualitative analysis, and is supported by a descriptive and documentary research, seeking to appreciate and reflect descriptively on the case and its possible contributions to the knowledge of the phenomenon of teaching programming and computational thinking in a playful environment such as Scratch.

Results

Based on this case study, we see that the blocks have a visual grammar and their combination rules have the same role as the syntax in text-based languages. The blocks

represent instructions with imperative mode texts in several languages and include control structures, event handling, use of variables, messages between objects, movement and sound.

It is relevant to notice that it is not enough just to consider the blocks, it is also important to take into consideration the library of scenes and objects to which material imported from other platforms can be added. In order to allow the learner to give his or her own impression and originality, it is possible to edit images, take photos, record and edit video sounds.

In this way, they can build original interactive projects for the student, such as video games, stories, songs, choreography, and simulations, among others. In addition, through a visual programming language like Scratch, students build their own code through scenes, characters, music and other components. As a consequence, this arouses great interest in the students, leading the teacher to delve even deeper into the possibilities of the tool.

In order to pose and solve a problem, it is necessary to use the programming control structures. Programming involves understanding the programming language, and using it to write codes. In addition, to understand the basics of programming with Scratch, it is important to analyze the control structures that control the flow of program execution. They influence readability and ease of writing. Increasing the control that the programmer has over a program, and therefore growing reliability.

An algorithm is defined as "a step-by-step procedure for solving a problem or accomplishing some end" (Merriam-Webster, s.f). Furthermore, any algorithm can be designed and implemented using only three types of control structures. Böhm & Jacopini (1966) demonstrated the structured programming theorem that specifies that all algorithms can be built using sequential, conditional (selection) and repetitive control structures. Therefore, three types of logical structures can be combined to build programs that provide a solution to various issues that are made: sequence, selection or conditionals and iteration or cycles.

Analysis of control structures with flowchart

As it is shown in Figure 2 the sequential structure seen is the simplest of all, it simply indicates to the processor that must consecutively execute a list of actions (which can be, at turn, other control structures); to build a sequence of actions. Figures 3 and 4 show through representative schematics as pseudocode and flowcharts are useful to express programming structures, which are a schematic representation of the sequences of a program and graphically represent the algorithms. On the one hand, we have the conditional statement that is used when an algorithm wants to establish that an action or sequence of actions is only executed if a certain condition is met. Conditional statements can be of several types: single conditional statement, double or alternative conditional statement, or multiple conditional statements. Iterative or loop structure is used when the execution of an action or sequence of actions must be repeated several times. A condition will determine when the iterations should continue and determines the continuity of the loop that can be defined before or after the sequence of actions. Algorithms created to solve problems can consist of simple sequences of instructions (Rahman et al 2020).

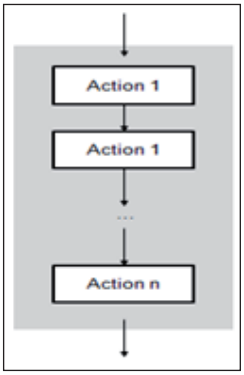


Figure 2: Sequential structure programming.

Selection or conditional structures control whether a statement or sequences of sentences are executed, depending on whether or not a condition or expression is fulfilled logic. The possible actions to be carried out are mutually exclusive; that is, it can only be run one at a time within the entire structure.

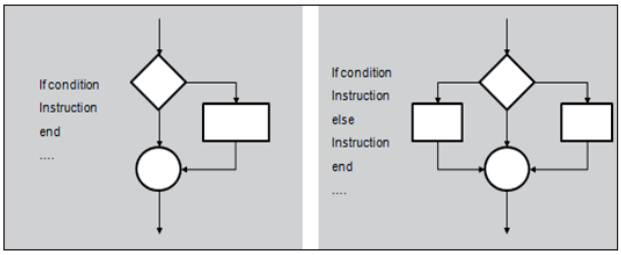


Figure 3: Sequential structure programming.

The repetitive or iterative structure allows, as its name suggests, repeating an action (or group of Actions); this repetition can be carried out a predetermined number of times or depend on the evaluation of a logical expression. There are three types of repetitive structures: from-to, while, and repeat-to (Figure 4).

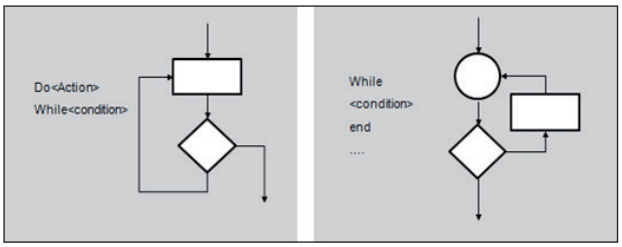


Figure 4: Repetitive structure programming.

Analysis of Control structures with Scratch

Scratch's role as a programming learning tool is very important since it allows you to create and share programs. As Scratch is very dynamic software, it is possible to work with the same control structures mentioned in the previous section, we detail the blocks to express the control structures. Table 2 shows the structures of control with Scratch, the use of colored blocks allows and facilitates many concepts expressed in sentences to become tangible, thereby facilitating the learning of abstract and complex concepts that are not easy to assimilate due to the lack of concrete references (Moreno-León et al. 2020). Detailed control structures and instructions, which can be associated with each object, are grouped into blocks according to their functionality. The blocks are snapped together to create the programs or scripts and the instructions are executed in order from top to bottom.

The graphical interface can change the background and a series of movable objects. Each object is called a sprite and can have one or more disguises; that is, a different aspect that the same object can take (for example, make a character walk). At the bottom of the scene is an area that shows the thumbnails of all the sprites in the project. The central panel of the graphical interface has three tabs: programs, costumes, and sounds. Selecting each of these tabs will change the panel on the right, where the user can see and modify the costumes (images), sounds, or behavior (the program) of the chosen sprite.

Analysis of a didactic game

In this example, the book series "Choose your own adventure" is emulated, where telling a story in which readers are the main protagonists and actively participate in decision-making, interacting with other characters, making choices, is emulated. We present the example developed by Fernández Casal (2016) where defines "Choose your own Adventure" as an educational proposal to develop in Scratch that is integrated with Language Practices and honors this kind of stories. This example focuses on the idea that students choose a story or tale from the language area. After that, they will modify the story to transform it into a format 'choose your own adventure'. The story must be analyzed using concept maps or other resources in order to know exactly decision taking moments in the story, the main and secondary characters. Then, it is possible to convert the story to graphic scenes with characters, dramatizing situations. Once located in this instance, the

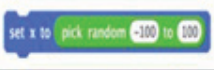
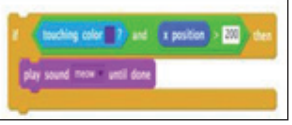
Concept	Explanation	Scratch
sequence	To create a program in Scratch, you need to think systematically about the order of steps.	
iteration (looping)	*forever* and *repeat* can be used for iteration (repeating a series of instructions)	
random	*pick random* selects random integers within a given range.	
conditional statements	*if* and *if else* check for a condition.	
boolean logic	*and*, *or*, *not* are examples of boolean logic	

Table 2: Control structures using Scratch.

idea is to design the bifurcations; that is, the different paths that the story can take. The choice of one path or another will be taken at runtime by another child or adult to whom we show the story we are putting together. In this way, children are designing and implementing human-computer interactions.

From the pedagogy of programming teaching, the game contributes to significantly improve its processes, to achieve the objectives set for an activity or program, and in the social construction of knowledge determined by aspects such as cognitive, affective and communicative. Conditional control structures, the use of variables and events are used to carry out the bifurcations. In the example, the program is modeled as a sequence of instructions whose execution produces a series of actions that change or transform the initial state of the environment, going through various intermediate states to reach a final state, which represents the solution to the problem.

Contextual analysis of Scratch

Advantages:

• It allows students to focus on the logic of programming by abstracting from the grammar of the language itself.

• It allows you to apply programming principles without having to worry about the syntax of a traditional programming language or having to face the intimidation of a professional programming language.

• It seeks that the instructions are at a visual level, by means of assembling blocks, which have different instructions so that programming is reduced to selecting and orderly assembling the instructions to be executed.

• It is a playful environment since it includes characters, sound and interactions that allow the generation of programs built with elements of structured basic programming.

• It favors student learning since it is based on the following characteristics: collaborative learning, critical thinking, the development of creative qualities, visualizing, expressing, exploring and understanding.

• It has simple interfaces, with which the interaction makes them accessible and is oriented to the rapid understanding of the students, also the environments are motivating, mainly oriented to learning to through the game.

• It promotes critical thinking in the context of exploration and discovery and independence of the skill level of students, both at the level of reading comprehension and computer experience.

• It allows students to achieve the formulation of simple problems and the construction of strategies for their resolution, including their decomposition into small parts, using ordered sequences of instructions, using creativity and experimenting with error as part of the process.

• It favors learning through its features and, considering that there are a wide variety of very complete tools, it is important to bear in mind that these types of resources allow the representation and understanding of different concepts related to programming.

Disadvantages

• It should be used according to the ages and the didactics to be applied according to the context.

• It is important to bear in mind the design of narratives of the didactic activities and adapt them to combine with language and the digital medium and build knowledge in a playful and creative framework.

• The number of sentences is reduced: although visual programming with blocks, the code is created by dragging blocks and placing it in the coding area, it is not necessary to write the sentences, it only requires moving the blocks, then the number of categories of blocks as the number of blocks within each category are reduced, so that only the most basic of the syntax of any programming language are preserved.

• To better understand the importance of each teaching material, it is necessary to take into account the benefits they provide for understand if they adapt to the needs, expectations and benefits, this assessment and perception depends on the teacher and the practical application they want to do as well as the context and technological resources that are available.

Discussion

The digital game through educational and didactic software has been implemented as a resource in different fields of knowledge, giving rise to a large number of contributions of great variety. The implementation of digital games for the teaching of programming allows us to understand and see the game as a crucial factor for improving the quality of education, due to the results and the change in attitude observed in the student in the face of the frustration that commonly generates learning the logic of programming.

With three types of basic instructions, not only statements or problems can be developed. It is possible to add and include the development of playful activities, as we saw through the construction of educational video games, reinforcing programming knowledge on the pillars that we mentioned, within a constructivist framework of reference for learning. As we can see in the example, the program that the students developed is a sequence of instructions whose execution produces a series of

actions that change or transform the initial state of the environment, going through various intermediate states to reach a final state, which represents the solution of the problem. Besides, as we have discussed, Scratch allows students to use the fundamental control structures for the writing of any algorithm in addition to being able to apply other concepts such as variables and events.

We can see through the analysis that Scratch, being a playful environment, allows students to concentrate on the logic of programming, abstracting from the grammar of the language itself and applying programming principles without having to worry about the syntax of a traditional programming language or having to face the intimidation of a professional programming language. Likewise, it seeks that the instructions are at a visual level, by means of assembleable blocks, which have different instructions so that programming is reduced to selecting and assembling the instructions to be executed in an orderly manner. In an accessory way, we can see that with the example being a playful environment they include characters, sound and interactions that allow generating programs built with elements of structured basic programming.

In this way, we can see as a consequence that it favors the learning of students since it is sustained in features like collaborative learning, critical thinking, the development of creative qualities, visualizing, expressing, exploring and understanding, through of a simple interface, with which the interaction makes them accessible and is oriented to the rapid understanding of the students, also the environments are motivating oriented to learning.

Conclusion

The students are considered as digital natives, since they were born and are immersed in an environment surrounded by technology, and are internalized in the use of computers and smartphones. However, just because a student is using technology does not mean that they know how to code. Programming is vital knowledge for a world ruled by digital, which is not a closed skill that only serves the purpose of creating code, but also serves to develop other skills and learn new ones.

Learning programming with Scratch is no different than other learning processes and the flexibility allows teachers to implement visual concept lessons. That is, the teacher can use the tool to create animations that help to visualize difficult concepts of physics, chemistry or mathematics, but also put together stories using scenarios and characters, bringing them to life and proposing different interactions. Educational software, designed for students to program, can be a means through which a contact between technology and the area of education can be established. The purpose of this research is to analyze and carry out a review of the different proposals that have as a strategy the learning of programming as pedagogical tools that support the improvement of cognitive and problem-solving skills in students who are in school. Moreover, being in an increasingly digitized world, it is no longer enough to teach the student to be a user of the available technology, knowing how to program helps to better understand the environment, having the fundamentals of structured programming regardless of the software tool used. The inclusion of programming in a transversal way requires a permanent update which is essential in the construction of an updated and relevant education to the times in which it is developed. We can foresee that current programming platforms will continue to evolve and greatly simplify the way they work, to make it easier to adapt and use by digital natives. We can think then, that these technological resources represent instruments to create situations and address content that allow students to experience changes and transformations. Finally, those who have more access to technology expand the possibility allowing them to have greater access to learning and, in fact, participate in what digital citizenship is. In future works, we will analyze other educational tools to address programming and computational thinking such as Lightbot, PilasEngine, Pilas Bloques and so on.

References

1. **Adell, J. S., Llopis, M. A. N., Esteve, M. F. M., y Valdeolivas, N. M. G. (2019).** *El debate sobre el pensamiento computacional en educación*. RIED. Revista Iberoamericana de Educación a Distancia, 22(1), 171-186. doi: http://dx.doi.org/10.5944/ried.22.1.22303
2. **Balladares Burgos, J., Avilés Salvador, M. y Pérez Narváez, H. O. (2016)** *Del pensamiento complejo al pensamiento computacional retos para la educación contemporánea.* Sophia: Colección de Filosofía de la Educación, 21,143-159. https://doi.org/10.17163/soph.n21.2016.06
3. **Böhm, C. & Jacopini, G. (1966).** *Flow diagrams, turing machines and languages with only two formation rules*. Communication ACM, 9, (5), 366–371. https://doi.org/10.1145/355592.365646
4. **Brennan, K. & Resnick, M. (2012).** *New frameworks for studying and assessing the development of computational thinking. Proceedings of the 2012 annual meeting of the American Educational Research, Vancouver, Canada*. Retrieved August 19, 2021, from https://bit.ly/3tV11wV
5. **Corradini, I., Lodi, M., & Nardelli, E. (2017).** *Conceptions and misconceptions about computational thinking among Italian primary school teachers*. In Proceedings of the 2017 ACM Conference on International Computing Education Research (ICER '17). Association for Computing Machinery, New York, NY, USA, 136–144. https://doi.org/10.1145/3105726.3106194
6. **Csizmadia, A., Curzon, P., Dorling, M., Humphreys, S., Ng, T., Selby,**

C., & Woollard, J. (2015). *Computational Thinking. A guide for Teachers*. Computing At School. http://community.computingatschool.org.uk/files/6695/original.pdf

7. **Fernández Casal, F. (April 29, 2016)** *Proyecto para aprender programando: Elige tu propia aventura*. Procomun. Retrieved August 19, 2021, from http://procomun.educalab.es/es/articulos/elige-tu-propia-aventura-con-scratch
8. **Gibbons, A. & Snake-Beings, E (2018)** *DiY (Do-it-Yourself) pedagogy: a future-less orientation to education*. Open Review of Educational Research, 5(1), 28-42. doi: 10.1080/23265507.2018.1457453
9. **Hu, Y., Chen, C.H., & Su, C.Y. (2021).** *Exploring the Effectiveness and Moderators of Block-Based Visual Programming on Student Learning: A Meta-Analysis*. Journal of Educational Computing Research, 58(8), 1467–1493. https://doi.org/10.1177/0735633120945935
10. **ISTE (Agust 12, 2021)** *Computational Thinking Competencies*. Integrate CT across disciplines, with all students: CT competencies for educators. Retrieved August 19, 2021, from https://bit.ly/3kqBj04
11. **Lye, S. Y. & Koh, J. (2014).** *Review on teaching and learning of computational thinking through programming: What is next for K-12?*. Computers in Human Behavior, 41, 51-61. https://doi.org/10.1016/j.chb.2014.09.012
12. **Merriam-Webster. (n.d.).** *Algorithm*. In Merriam-Webster.com dictionary. Retrieved August 19, 2021, from https://bit.ly/3EFt6Nx
13. **Moreno-León, J., Robles, G. and M. Román-González, M. (2020).** *Towards Data-Driven Learning Paths to Develop Computational Thinking with Scratch*. IEEE Transactions on Emerging Topics in Computing, 8(1), 193-205. doi: 10.1109/TETC.2017.2734818.
14. **Morris, D., Uppal, G., & Wells, D. (2017).** *Assessing pupil progress in computational thinking and coding.* In Teaching computational thinking and coding in primary schools (pp. 154-168). Learning Matters, https://www.doi.org/10.4135/9781529714647.n1
15. **Müller, A. (2020).** *What is constructivism?* Constructing Practical Reasons, 6–32. doi:10.1093/oso/9780198754329.003.0002
16. **Ortega Ruipérez, B., & Asensio Brouard, M. (2021).** *Evaluar el Pensamiento Computacional mediante Resolución de Problemas: Validación de un Instrumento de Evaluación*. Revista Iberoamericana De Evaluación Educativa, 14(1), 153–171. https://doi.org/10.15366/riee2021.14.1.009
17. **Phillips, P. (2008).** *Computational Thinking: A Problem-Solving Tool for Every Classroom*. Retrieved August 19, 2021, from https://bit.ly/3nU9RKc
18. **Rahman, M. M., Sharker, M. H. & Paudel, R. (2020)** *An Effective Approach to Teach an Introductory Computer Science Course with Computational Thinking and Flow-Chart Based Visual Programming.* 2020 IEEE Frontiers in Education Conference (FIE), 1-7. doi: 10.1109/FIE44824.2020.9273930.
19. **Rich, P. J., & Langton, M. B. (2016).** *Computational Thinking: Toward a Unifying Definition.* En J. M. Spector, D. Ifenthaler, D. G. Sampson, & P. Isaias (Eds.), Competencies in Teaching, Learning and Educational Leadership in the Digital Age (pp. 229-242). Cham: Springer International Publishing. doi: https://doi.org/10.1 007/978-3-319- 30295-9_14
20. **Rojas López, Arturo (2019).** *Escenarios de aprendizaje personalizados a partir de la evaluación del pensamiento computacional para el aprendizaje de competencias de programación mediante un entorno b-Learning y gamificación [Tesis Doctoral]*. Universidad de Salamanca.
21. **Soleimani, A. (2019).** *Computational Design Thinking and Thinking Design Computing*. In 2019 Reynolds Symposium: Education by Design. Portland, Oregon. https://doi.org/10.21428/f7d9ca02.b7daadcc
22. **Soria Valencia, E., & Rivero Panaqué, C. (2019).** *Pensamiento computacional: una nueva exigencia para la educación del siglo XXI*. Revista Espaço Pedagógico, 26(2), 323–337. doi:10.5335/rep.v26i2.8702
23. **Sykora, C. (April 23, 2021)** *Computational Thinking for All.* ISTE. Retrieved August 19, 2021, from https://bit.ly/3AySVfW
24. **UNICEF (2018).** *Aprendizaje a través del juego Reforzar el aprendizaje a través del juego en los programas de educación en la primera infancia, Editorial UNICEF.* Retrieved August 19, 2021, from https://uni.cf/39pdFLm
25. **Wing, J. (2006).** *Computational Thinking*. View Point. Comunication of ACM. 49, 3. 33-35. https://doi.org/10.1145/1118178.1118215
26. **Wing, J. (2010).** *Computational Thinking: What and Why?* Retrieved August 19, 2021, from https://bit.ly/3nU0RFa
27. **Zapata-Ros, M. (2015).** *Pensamiento computacional: Una nueva alfabetización digital*. Revista de Educación a Distancia (RED), (46). doi:10.6018/red/45/4
28. **Zapata-Ros, M. (2019).** *Pensamiento computacional desenchufado.* Education in the Knowledge Society (EKS), 20, 29. https://doi.org/10.14201/eks2019_20_a18