An analysis of block-based programming environments for CS1

Luiz Carlos Begosso
Computer Science Department
Fundacao Educacional do Municipio de Assis – FEMA
Faculdade de Tecnologia de Assis - FATEC
Assis, Brazil
luiz.begosso@fema.edu.br

Luiz Ricardo Begosso
Computer Science Department
Fundacao Educacional do Municipio de Assis – FEMA
Assis, Brazil
begosso@fema.edu.br

Natalia Aragao Christ
Computer Science Department
Fundacao Educacional do Municipio de Assis – FEMA
Assis, Brazil
natalia.aragao.154@gmail.com

Abstract— **This Research Full Paper presents our experience in analyzing and selecting block-based programming environments to support the teaching of algorithms for the students starting the introductory courses of a Computer Science major. The teaching of algorithms and programming concepts to students of the first years of Computer Science and Engineering courses has been a major challenge because students often have difficulty understanding the logic and abstraction, leading to a high dropout rate. Some strategies have been conducted to further the mission of helping students understand better those basic concepts, but this topic still remains a major problem for students in the initial grades of those courses. In previous projects developed at our university, we have already proposed the use of learning objects and gamification, with very positive results. One of the questions that arise when we adopt new teaching approaches is to know how this new path will contribute to the student's learning. In this project, we conducted a study on eight block-based programming environments and sought to identify which aspects of those environments comply with the Computer Science reference curriculum. Our work was based on the joint task force on Computing Curricula conducted by the ACM and IEEE Computer Society CS2013 curriculum guidelines for undergraduate programs in Computer Science. We studied the virtual programming environments Alice, MIT App Inventor, Blockly Games, Code.org, Gameblox, Pencil Code, Microsoft MakeCode and Scratch. Then, we crossed the characteristics of each, identified the positive and negative points of each teaching environment in relation to the topics established by the guidelines. We have classified the main characteristics of those programming environments, establishing criteria such as: prior programming knowledge requirements; ease of interaction with users; programming language code; availability of documentation for learning; programming practices addressed by the environment; and ease of learning programming. We believe that this work can contribute to the selection process of a suitable programming environment to be adopted in an introductory course of computer programming. (*Abstract*)**

Keywords— CS1, block-based programming, reference curriculum, visual programming language.

This research was supported by Fundacao Educacional do Municipio de Assis.

I. Introduction

Students in the first year of Computer Science courses need to learn algorithms and how to program computers and one of the biggest challenges is to learn computational thinking, thought processes required in understanding problems and formulating solutions. Besides, first-year classes are usually large and heterogeneous, making the teaching process still more difficult for the teachers; also, it is known that several first-year students have high learning difficulties due to the nature of the subject, as most of them are exposed to algorithms for the first time in their lives [1]. In Brazil, most students start learning the concepts of algorithms construction only at university and this condition shows a failure in the educational system about the diffusion of computational thinking among students.

Computational thinking is associated to the problem solving, system designing and understanding of human behavior resources, based on the fundamental concepts of Computer Science [2]. Promoting computational thinking is an initiative that includes a range of mental tools that reflect the scope of Computer Science.

Several researchers have studied about the difficulties faced by Computer Science students in the initial years [3], [4], [5]. In general, novice students believe that computer programming is a difficult task, as it requires a lot of knowledge and skills to be learned at once. In order to minimize this condition, the literature presents some important initiatives and, in the context of this work, we will explore visual programming languages (VPLs), which are language environments that use graphical elements and figures to create a program, rather than by specifying it textually. If we can offer these languages to Computer Science first-year students, they will be allowed to learn programming concepts using graphic manipulation.

Many VPLs are based on the idea of "boxes and arrows", or "block fit" in which boxes or blocks are objects treated as entities, connected by arrows, or organized into blocks that represent computational relationships. Usually, VPLs are used as an additional tool for helping novice students learn programming concepts, especially in the context in which the student has limited knowledge of computer programming [6]. Unlike traditional text-based programming, block-based coding is not initially concerned with syntax and software quality issues. The goal is to demystify computer programming with beginner students.

We understand that visual-oriented programming is a good starting point for first-year students in Computer Science, but obviously it shouldn't be used all the time because it is not expected that students will learn real programming just by building block and visual code, but it is something addictive for novice students and that prepares them to learn and produce more difficult code.

It is known that the retention of Computer Science students has been a great concern in the academia. In Brazil, studies from the Brazilian Computer Society (SBC) show that in 2016 we had 22,463 new students for Computer Science and only 6,470 graduate students; in 2017 we had 22,444 new students and only 6,161 graduate students, as described in Table I [7].

TABLE I. EVOLUTION OF THE NUMBERS OF NEW STUDENTS AND GRADUATES IN BRAZIL

Course	**2016**		**2017**	
	New Students	Graduate Students	New Students	Graduate Students
Computer Science	22643	6470	22444	6161
Computer Engineering	11707	2114	10680	2267
Software Engineering	1518	144	2087	232

There are several reasons for the low retention in Computer Science courses, like socio-economic issues, vocation issues and learning issues. For the learning issue, we can identify mainly difficulties with mathematics and difficulties with programming concepts. In order to address the last argument, visual programming languages can be used to facilitate the process of learning programming concepts for novice students.

In this work, we studied eight VPLs that are commonly used to drag blocks around the screen, draw flow diagrams or use icons / representation without text. We investigated the features of block-based programming environments, with the objective of identifying and classifying the fundamental concepts for software development recommended by the joint task force on Computing Curricula conducted by the ACM and IEEE Computer Society CS2013 curriculum guidelines for undergraduate programs in Computer Science [8]. The results of our research about those eight block-based programming environments showed that some environments are highly favorable for pedagogical practice for teaching algorithms and problem solving for novice students.

In the next sections we will present the VPL environments, the research methodology, our results and discussion. Finally, we will present the conclusions.

II. BLOCK-BASED PROGRAMMING LANGUAGES

The theory of constructivism is directly associated to the concept inherent to the idea of visual programming languages (VPLs), emphasizing that the learners' understanding and knowledge are based on their own experiences, proposing that students actively participate in their own learning process, through experimentation, group research, the stimulation of questions and the development of reasoning, among other procedures. The learning process is different and unique for each individual and this learning does not occur in the same way, at the same pace and at the same time. We believe that, when investing in this study, we have the chance to make the contents taught more attractive and interactive, being essential for students with difficulties in solving problems, confidence, critical sense, protagonism and creativity. In this way, block-based tools were designed to produce features of accessible and attractive programming environments. The color and shape of the commands, the organization of the blocks and the mechanism for building the programs are easily navigable and displayed in order to help support students who are new to writing programs. In this perspective, we studied these VPLs for conducting our work: Alice, MIT App Inventor, Blockly Games, Code.org, Gameblox, Pencil Code, Microsoft MakeCode and Scratch, which will be briefly described in the next paragraphs.

A. Alice

Alice was developed at University of Virginia in 1994, it is a 3D object-based educational programming environment that was designed for beginners in programming. In Alice drag and drop software, students can create a three-dimensional animation to tell a story, design a game or a video. The environment is attractive for students, as it presents several 3D model elements, such as people, objects, animals and vehicles, which can be animated by the student [9], [10].

B. MIT App Inventor

App Inventor is described as an intuitive, visual programming environment projected for everyone – even children – to build fully functional apps for smartphones and tablets. The environment provides an intuitive programming metaphor and consists of two parts: a designer that allows the student to select the components of the application and a block editor to define the behavior of the application [11], [12].

C. Blockly Games

Blockly Games was developed for children and young students with no previous experience with computer programming. It was designed so that this audience can use the environment without an instructor, as its educational mechanics was designed to be self-taught. With Blockly Games the student receives instructions from a series of educational games and, at the end of the sequence of these games, it is expected that the student is ready to use conventional text-based programming languages [13].

D. Code.Org

Code.org is a nonprofit organization dedicated to expanding access to Computer Science in schools and increasing participation by women and underrepresented youth. The visual environment provides an opportunity for students to learn the principles of computer programming through drag and drop of blocks of code. The platform facilitates the understanding of programming assumptions, stimulating creativity and works with the following concepts: programming logic, clarity in expressions, conditional structures, variables, loops and functions [14].

E. Gameblox

Gameblox is a block-based programming environment for creating online games for the web and mobile devices. It consists of a block editor that allows students without prior knowledge of game programming to be able to build their own game [15].

F. Pencil Code

Pencil Code is a collaborative programming site for drawing, playing music, creating games, learning geometry, graphics, building web pages and algorithms. Pencil Code can also be used to explore and learn Java script, HTML and CSS [16].

G. Microsoft MakeCode

Microsoft MakeCode is a free, open source platform for engaging learning experiences in Computer Science. In MakeCode students may have access to several tools for helping their learning, such as the interactive simulator that provides them with immediate feedback on how their program is running and facilitates testing and debugging the code. MakeCode is a block editor, where students who are new to programming can start with colored blocks, drag and drop into their workspace to build their programs [17].

H. Scratch

Scratch is a visual and multimedia programming context based on Squeak. It aims to create animated sequences for learning computer programming in a simple and efficient way. This VPL has an intuitive, easy and understandable interface, so that novice students can learn without much difficulty. In Scratch, students can work with images, photos, music, create drawings, change appearance, make objects interact with each other and with the user, making programming entirely visual [18], [19].

III. METHODOLOGY

We adopted a descriptive approach to conduct this research, carrying out bibliographic survey followed by the study and evaluation of visual programming languages environments.

We defined and applied three steps to organize the study and evaluation of the applications of this project: first, we identified the state of the art in the area of block-based programming languages; second, we studied the selected environments; and finally, we evaluated which aspects of those environments comply with the Computer Science reference curriculum. To reach this last step, our work was based on the joint task force on Computing Curricula conducted by the ACM and IEEE Computer Society CS2013 curriculum guidelines for undergraduate programs in Computer Science. Figure 1 illustrates the steps used in our methodology:

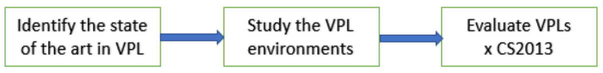
Identify the state of the art in VPL
Study the VPL environments
Evaluate VPLs x CS2013

Figure 1 – Steps used in the methodology

CS2013 establishes the importance of introducing the fundamental concepts of programming and advocates including the following study topics for novice students: syntax and semantics of a high-level programming language; variables and primitive data types (e.g., numeric, characters, and booleans); expressions and attributions; simple I/O; file I/O; conditional control structures; iterative structures (repetitions); functions and parameters passing; recursion concept. Based on these guidelines, we considered to identify and classify the CS2013 recommendations in view of the main characteristics of each of the eight VPL environments studied.

IV. RESULTS AND DISCUSSION

Based on the criteria established in the previous section, we developed Tables II and III, where it was possible to cross the characteristics of the visual programming environments with the respective recommendations of CS2013.

For the purpose of comparing all the studied environments, we agreed to detail some aspects of the CS2013 recommendations and obtained the following analysis criteria: basic syntax, block change to script, variables and primitive data types, expressions, assignments, I/O simple, conditional and iterative structures, functions and parameters passing, and concept of recursion.

TABLE II. RESULTS OF THE EVALUATED CRITERIA

	BS	BCS	VN	VC	VB	Exp
Alice	■		■	■	■	■
APP inventor	■		■	■	■	■
Blockly Games	■	■	■	■	■	■
Code.org	■		■	■		■
GameBlox	■	■	■	■	■	■
Pencil Code	■	■	■	■		■
MakeCode	■	■	■	■	■	■
Scratch	■		■	■	■	■

Table legend:

BS: Basic Syntax

BCS: Block Change to Script

VN: Variables numeric

VC: Variables characters

VB: Variables Boolean

Exp: Expressions and Attributions

TABLE III. RESULTS OF THE EVALUATED CRITERIA (CONT.)

	Assig	Simp I/O	CIS	Funct	Recurs
Alice	■	■	■	■	■
APP inventor	■	■	■	■	■
Blockly Games	■	■	■	■	
Code.org	■	■	■	■	■
GameBlox	■	■	■	■	■
Pencil Code	■	■	■	■	■
MakeCode	■	■	■	■	■
Scratch	■	■	■		

Table legend:

Assig: Assignments

Simp I/O: Simple I/O

CIS: Conditional and Iterative Structures

Funct: Functions and Parameters Passing

Recurs: Recursion

In tables II and III, the dark cells reflect that the tool meets that characteristic, while the light cells indicate that the tool does not meet that characteristic. In the next paragraphs we will describe which characteristics are not found in each VPL.

The tool Alice teaches object-oriented programming and programming in a 3D virtual graphic environment; however, it does not have the functionality of changing the block program into script programming code.

AppInventor assists in building mobile applications for the Android platform using a web-based graphical user interface builder and does not have the functionality of generating script programming code from the blocks created by the user.

Blockly Games uses pluggable graphic blocks to represent the concepts of programming code, but it does not have a resource that allows working with recursion concepts.

Code.org is a platform that provides resources to learn and teach concepts of Computer Science, through digital educational games, and does not allow the generation of script programming. It also does not have the functionality of representation of logical expressions, like "and" and "or".

Gameblox is a tool projected for game design. It provides an environment for creating fun games incorporating some more complex programming tasks such as physics content (collision and gravity) within the blocks. This VPL is quite complete, but the resources of assignment, functions and parameters, and concepts of recursion are used only when there is alternation of block programming for the script generated by the environment.

In Pencil Code it is possible to create routines with blocks of actions, the basis of programming logic. This environment emphasizes creativity, like drawing or creating music from mathematical thinking. Pencil Code does not have the functionality of representing a Boolean data type.

MakeCode covers all the basic programming concepts that help to develop creativity, computational reasoning and collaboration skills, through the block options available.

Finally, the Scratch platform, is a visual and multimedia programming environment that teaches how to produce animation, with or without sound, and helps in learning good programming structures. There are several initiatives in the literature with this VPL; however, it does not allow the generation of scripts, nor does it work with the concepts of recursion, functions and parameters passing.

V. Conclusion

In this work we had the objective to identify and classify block-based programming environments as pedagogical practices for the initial teaching of logics and computer programming. We analyzed eight visual programming environments (VPL): Alice, MIT App Inventor, Blockly Games, Code.org, Gameblox, Pencil Code, Microsoft MakeCode and Scratch.

We described the tools, the study methodology, and classified the fundamental concepts of each VPL studied according to the recommendations of the joint task force on Computing Curricula conducted by the ACM and IEEE Computer Society CS2013 curriculum guidelines for undergraduate programs in Computer Science.

The results point to an interesting horizon, as we found that there is a predominance of block-based languages that favors the development of games, which is very good, as the area of games tends to arouse interest in the group of young students of courses in computing.

To address a specific content that is better worked by a specific VPL, we suggest that teachers can use more than one programming teaching environment, since the block programming paradigm is common among all the environments studied in this work.

None of the explored languages requires prior knowledge of programming, since they are considered environments for teaching novice students. Most VPLs are already prepared for several foreign languages, which can be effective when we want to include, in all parts of the world, the largest number of young students in the art of computer programming.

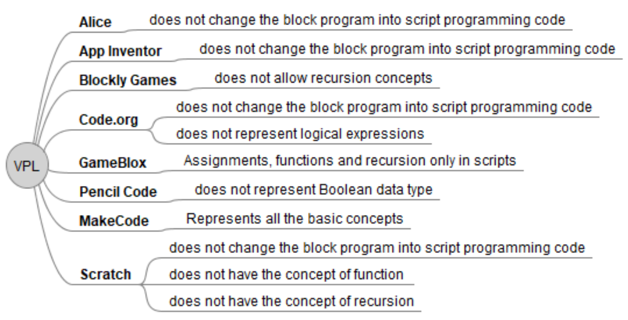
VPL
Alice
does not change the block program into script programming code
App Inventor
does not change the block program into script programming code
Blockly Games
does not allow recursion concepts
Code.org
does not change the block program into script programming code
does not represent logical expressions
GameBlox
Assignments, functions and recursion only in scripts
Pencil Code
does not represent Boolean data type
MakeCode
Represents all the basic concepts
Scratch
does not change the block program into script programming code
does not have the concept of function
does not have the concept of recursion

Figure 2 – Missing Characteristics in VPLs, according to CS2013

In summary, our proposal was to conduct a study on the characteristics of each VPL in relation to the CS2013 recommendations. Figure 2 summarizes the characteristics which we considered missing in the VPLs studied in this work.

Finally, we hope that this work can contribute to the development of teaching and learning processes of computer programming, presenting teachers with a perspective on several visual programming environments and an evaluation of which one can be used to reach different teaching objectives.

References

[1] E. Lahtinen, K. Ala-Mutka and H. Järvinen, "A study of the difficulties of novice programmers", in: Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, ITiCSE 2005, Caparica, Portugal, June 27-29, 2005, p. 14-18.

[2] J. M. Wing, "Viewpoint: Computational thinking", Communications of the ACM, vol. 49, Number 3, pp.33-35, March 2006.

[3] L. R. Begosso, L. C. Begosso, D. S. da Cunha, J. V. Pinto, L. Lemos and M. Nunes, "The Use of Gamification for Teaching Algorithms", in: Proceedings of the 2018 Federated Conference on Computer Science and Information Systems, FEDCSIS, Poznan, Poland, September 9-12, 2018, p. 225-231.

[4] L. R. Begosso, L. C. Begosso, R. H. Begosso, "An approach for the use of Learning Objects in teaching computer programming concepts", in: Proceedings of the 46th IEEE Frontiers in Education Conference, FIE, Eire, PA, USA, October 12-15, 2016, p. 1-8.

[5] S. Garner, P. Haden, A. Robins, "My program is correct but it doesn't run: a preliminary investigation of novice programmers' problems", in: Proceedings of the 7th Australasian Conference on Computing Education, ACE,(42), January 2005, p. 173–180.

[6] H. Nishino, "Ardestan: A Visual Programming Language for Arduino", in: The Adjunct Publication of the 32nd Annual ACM Symposium on User Interface Software and Technology (UIST '19) Association for Computing Machinery, 2019, New York, NY, USA, October, 2019, p.93-95.

[7] Brazilian Computer Society SBC, "Higher Education in Computing Statistics 2017", "Educação Superior em Computação Estatísticas 2017", https://www.sbc.org.br/documentos-da-sbc/category/133-estatisticas, 2017.

[8] Association for Computing Machinery ACM, IEEE Computer Society, "Curriculum Guidelines for Undergraduate Degree Programs in Computer Science", USA, 2013, 514 p.

[9] C. W. Herbert, "An Introduction to Programming Using Alice 2.2", 2nd ed., Boston, USA, 2011.

[10] W. P. Dann, S. P. Cooper and B. Ericson, "Exploring Wonderland: Java Programming Using Alice and Media Computation", Georgia Institute of Technology, USA, 2010.

[11] L. B. Logan, "Learn to Program with App Inventor", No Starch Press, San Francisco, USA, 2019.

[12] P. Beer and C. Simmons, "Hello App inventor!: Android programming for kids and the rest of us", Manning Pub., New York, USA, 2015.

[13] N. Brown, J. Monig, A. Bau and D. Weintrop, "Future Directions of Block-based Programming", in: Proceeding of SIGCSE '16 Proceedings of the 47th ACM Technical Symposium on Computing Science Education, 2016, Memphis, Tennessee, USA, March, 2016, p.315-316.

[14] F. Kalelioğlu, "A new way of teaching programming skills to K-12 students: Code.org", Computers in Human Behavior, vol. 52, pp. 200-210, November 2015.

[15] D. Bau, J. Gray, C. Kelleher, J. Sheldon and F. Turbak, "Learnable programming: blocks and beyond", Communications of the ACM, vol. 60, 6, May 2017, 72–80.

[16] D. Bau, D. A. Bau, M. Dawson, and C. S. Pickens, "Pencil code: block code for a text world", in Proceedings of the 14th International Conference on Interaction Design and Children, IDC' 15, Association for Computing Machinery, New York, NY, USA, June, 2015, pp. 445–448.

[17] T. Ball, A. Chatra, P. de Halleux, S. Hodges, M. Moskal, and J. Russell, "Microsoft MakeCode: embedded programming for education, in blocks and TypeScript", in Proceedings of the 2019 ACM SIGPLAN Symposium on SPLASH-E, SPLASH-E 2019, Association for Computing Machinery, New York, NY, USA, October 20-25, 2019, pp. 7–12.

[18] J. Woodcock, Coding Games in Scratch. New York: DK Publishing, 2016.

[19] G. A. B. Joshi, R. Pande, Advanced Scratch Programming: Learn to design programs for challenging games, puzzles, and animations, 1st ed. Scotts Valley: Create Space, 2016.