JCSE Journal of Computing Science and Engineering

Regular Paper
Journal of Computing Science and Engineering,
Vol. 13, No. 1, March 2019, pp. 1-10

A Methodology for the Analysis of Block-Based Programming Languages Appropriate for Children

Radoslava Kraleva* **and Velin Kralev**
Department of Informatics, South-West University “Neofit Rilski”, Blagoevgrad, Bulgaria
rady_kraleva@swu.bg, velin_kralev@swu.bg

Dafina Kostadinova
Department of Germanic and Romance Studies, South-West University “Neofit Rilski”, Blagoevgrad, Bulgaria
dafinakostadinova@swu.bg

Abstract
Nowadays, the interest of young people in programming is decreasing steadily on a global scale. This, however, is becoming a problem for global economic development. The dynamic development of technologies requires implementation of new teaching and learning methods. As a result, new Computer Science courses related to programming in primary education have been introduced. Pupils learn the basics and the programming skills using new visual programming languages known as block-based programming languages that allow the design of programming algorithms (program logic) using drag-and-drop of program chunks, named blocks. This makes the programming languages easy to use even by young children. The lack of a reasonable argument for the choice of block-based programming languages based on their functional characteristics, interface and children’s preference prompted this investigation. This article discusses some of the modern block-based programming languages. Research into the state-of-the-art scientific publications on this issue has been done. The criteria for comparing and analyzing these programming languages have been defined. As a result, the block-based programming languages that best meet the criteria have been identified. Two languages (Scratch and Code.org) have been selected based on the proposed methodology. These languages were used for two weeks by pupils in the 3rd and 4th grades in Bulgaria. The main goal of this study is to determine the degree of similarity between block-based and traditional programming languages, as well as discuss the opportunity for their use in the Bulgarian primary school. The proposed methodology can be easily adapted and used in other countries. An important factor in this research is the support available in the pupils’ native language for the integrated development environment and programming languages.

Category: Compilers / Programming Languages

Keywords: Block-based programming languages; Programming language for children; Human-computer interaction; Computational thinking; Computer science education

I. INTRODUCTION

The fast-paced, over-technological lifestyle has revealed new aspects of modern programming. Today, the textual programming environments, which until recently have been the main mode of programming, have paved the way to visual programming languages. They have evolved into environments that allow programming based on the

Open Access http://dx.doi.org/10.5626/JCSE.2019.13.1.1 http://jcse.kiise.org

Received 10 October 2017; **Revised** 26 January 2019; **Accepted** 27 January 2019
*Corresponding Author

use of readily available program blocks and in some cases, the real source code remains hidden. These new visual programming languages are known as block-based programming languages (BBPLs). The integrated development environments in which they are used are called block-based programming environments (BBPEs). BBPLs allow the development of a computer program by dragging, dropping and snapping program chunks that are organized into different categories. Thus, people who never encountered application development can better understand the basic concepts of programming and the creation of algorithms [1]. Despite these innovations in software development, the need for computer specialists is increasing. For Larson [2], the shortage of IT staff in many countries around the world has become a serious problem. To solve this problem, governments and education ministries in many countries have changed their curricula by introducing subjects belonging to the computer science. These curricula aim not only at introducing information technology terms to children, but also the basic concepts of programming and the creation of computer programs and computer games.

In 2015, the Ministry of Education and Science in the Republic of Bulgaria added a new compulsory computer discipline named “Computer Modeling” for primary school students (grades 3 and 4) [3], which will start in 2019. According to the curriculum of this discipline, children should be able to write simple algorithms and develop computer applications, computer games and animated objects using visual environments and programming languages.

After investigating various literary sources, it was found that there was a lack of reasonable arguments in the methodologies for the selection of appropriate visual programming languages, based on their functionalities and interface. Analyses related to the children’s preferences are also very limited due to the intensive research into computer programming for children only within the last few years. All this motivated us to start working in this field of study and present the results of our investigation in this paper.

The paper is structured as follows: overview of the state-of-the-art studies, associated with BBPLs for children; several types of criteria used to determine the benefits and disadvantages of multiple BBPLs and their comparison with traditional programming languages; analysis of the possibility for use of BBPLs by pupils; analysis of the opinions of the pupils in a primary school regarding some of the BBPLs presented; and study conclusions.

II. RELATED WORK

Modern computer technologies define the new lifestyle of people, and children are no exception to that definition. From an early age, children start to use mobile devices (smartphones and/or tablets) to play, make phone calls, and for entertainment. But they can do much more with the help of computer technologies. Young children can visualize their ideas through computer drawings, animations, or computer games that they develop themselves.

The use of modern computer technologies by children as a learning tool is not a novelty. In the 80's of the 20th century, “Turtle”, the computer-controlled cybernetic animal, was created at MIT [4]. The “Turtle” is controlled by a computer language LOGO, which is the first computer language appropriate for children. A study of Papert [4] can be considered as fundamental to human-computer interaction as children are no longer seen as ordinary users but as a part of the computer program development team.

Thus, the development of comprehensible and intuitive programming languages has become a priority for all software engineers. The focus of interest in this paper is to study not only the way children interact with programming languages, but also their features and capabilities, the intuitive understanding of the developmental environment, and its similarity to classical object-oriented programming languages such as C++. The present study is the result of a 1-year work on the problem that was examined in [5] and [6].

A number of researchers offered original methodologies for teaching and learning computing programming for young students. In most cases, their proposals adhere to generally accepted practice and standards in their countries. An example of such a study can be seen in [7], investigating some game development environments, comparing their features, and the types of games that can be created with them are listed. Hayes and Games [7] argue that special attention should be paid to the development of thinking when designing learning games. Teaching computer programming to children aged 3 to 6 years in England is discussed by Manches and Plowman [8]. The two authors take into account the lack of scientific research related to pedagogical theory, generally accepted practices, and the scarcity of achievements and/or results that allow researchers and practitioners to obtain an overall assessment after introducing computer science training (programming or coding) in the “early years” of children.

It is not only the great capabilities of BBPLs that are of essential importance when used by primary school children. A study related to the development of algorithmic thinking among seven children over the age of 6 who used the *PiktoMir* online environment for 8 weeks is rendered in [9]. All the participants in the experiment ‘found *PiktoMir* fun to use’. Furthermore, children aged 5 to 11 develop their algorithmic thinking and apply it in practice [10]. According to Gibson [11], it is needless for them to wait until adolescence to study the creation and use of simple algorithms for mathematical computing or computer programming. He suggests that children can begin to learn computer science before they even know how to read and write. In another study published earlier, Gibson [11] presents the idea, "that the best way of introducing children to computer science does not require a computer: it requires the teaching of rigorous (formal) reasoning about computations and algorithms". He stated that the first formal methods can be taught to children as young as seven years and thus children can learn fundamental algorithm concepts easily. His methodology is based on games with real objects such as cubes, cardboards, etc.

The development of algorithmic thinking is crucial for mastering the basic concepts in programming and is facilitated by the modern BBPLs. However, it is questionable whether the children can understand how to use program chunks (blocks). The way children in the 4th grade understand and read an existing code using the visual cues provided by block-based programming is presented in [11, 12]. Dwyer et al. [12] recognize the fact that different pupils understand the program blocks differently. In addition, pupils find it hard to understand how the computer programs are designed and they encounter difficulties when developing simple algorithms. This study confirms once again Gibson's idea that the study of programs initially requires acquisition of knowledge related to building an algorithmic sequence of actions to achieve the respective purpose.

Another problem found in the literature is the difficulty young children encounter in understanding the concepts of abstraction in the programming languages, even those that are block-based. For example, Armoni [13] presents a study on the different opinions for and against studying computer science by children in the kindergarten, as well as the lack of detailed research into this issue. The author of this paper concludes that children must be at least 7 years old to be able to cope with the abstract concepts used in programming.

In the course of the present study, some articles reporting positive results associated with the use of BBPLs were found. According to [14], the pupils acquired basic knowledge of text-based programming when using such languages. Matsuzawa et al. [14] state that the language interface of the developmental environments, the availability of learning materials and a user-friendly interface are of vital importance. If all these features are available, the pupils can focus only on the development of algorithms and/or applications without handling exceptions and debugging. The authors also confirm the fact that there is still a lack of sufficient information concerning the way students perceive BBPLs.

Bearing in mind the contemporary literary sources reviewed above, it can be concluded as follows:

- Children must be at least 7 years of age in order to clearly understand the abstract programming languages;
- Prior to introducing programming languages, including BBPLs, it is necessary to acquaint pupils with the formal concepts and the basics of mathematical logic. This will help them to learn to build simple algorithms;
- BBPLs help to build algorithmic thinking;
- BBPLs help to further study the traditional programming languages.

III. A METHODOLOGY FOR THE ANALYSIS OF SOME BLOCK-BASED PROGRAMMING LANGUAGES

Block-based programs most often consist of stacked block elements that resemble puzzle elements. The program is run only by pressing the RUN button. BBPE that uses a BBPL lacks debugging and handling exceptions. The main purpose of this type of programming language is not simply to arrange program chunks. Instead, its purpose is to let children acquire knowledge related to the logical organization of an algorithm, so that they can solve a certain task. Objects and splines from the real world and background images are most commonly used in these programming platforms to design the computer program or game.

The authors of [8] point out six main areas of computer training: (1) understand the algorithms; (2) create and debug the program; (3) logical thinking; (4) work (create, store, organize, retrieve and delete) with digital content; (5) use information technology beyond school; and (6) ensure safety and keeping personal information private. All these indicators will be used in the development of our comparative methodology of BBPLs for children.

The main purpose of this study is to determine the degree of similarity between the block-based and the traditional programming languages, and the possibility of their use in the Bulgarian schools.

Similar analyses have been made in several scientific publications, e.g., [19, 21, 22] among others. The common feature shared by these analyses is that they investigated no more than two BBPLs or programming platforms. Moreover, the criteria used compare them only in terms of their capabilities and mode of application in the learning/teaching process. No clear set of criteria are available to assess the capabilities of the investigated languages compared with the traditional programming languages. The advantage of the methodology presented here is that it corresponds to and satisfies the conclusions of the study involving the various literary sources and provides an accurate assessment of the actual capabilities of a BBPL.

The criteria for analyzing some of the more common BBPLs are divided into four categories (groups): the first category refers to the user interface of the programming platform; the second one relates to the availability of teaching materials; and the third contains the main features of traditional programming languages. The proposed criteria can be used to analyze programming languages different from the ones used in this article.

Table 1. Analysis of some programming languages based on their usability and support

Name	Website	Age (yr)	Price	OS and other software	Language support	Science publications
Code.org	http://studio.code.org	4+	Free	All modern browsers	English, Bulgarian etc.	[23, 24]
ScratchJr	http://www.scratchjr.org	5+	Free	iOS, iPad, Android	English, Spanish	[25, 26]
Scratch	https://scratch.mit.adu	8+	Free	iOS, iPad, Android, Windows, Mac, Linux	English, Spanish, Bulgarian, etc.	[16, 24, 27, 28, 29, 30]
Tynker	https://www.tynker.com	7+	Free and payment	All modern browsers	English	[31]
Kodu Game Lab	http://www.kodugamelab.com	8+	Free	Windows	English	[15, 19, 32, 33]
GameStar Mechanic	http://gamestarmechanic.com	8+	Free and payment	All modern browsers	English	[34]
Hopscotch	https://www.gethopscotch.com	8+	Free	iOS, iPad, iPod, iPhone	English	[35]
Alice	http://www.alice.org	10+	Free	Windows	English	[27, 36, 37, 38, 39]
Snap!	http://byob.berkeley.edu	13+	Free and payment	All modern browsers	English, Bulgarian etc.	[40]

This methodology provides an independent evaluation of the programming languages investigated.

(1) Usability and support (Table 1)

- Age: Appropriate user age is determined according to the websites of the programming language and related scientific publications.
- Price and license
- Operating systems (OS) and other software: The requirements of the operating system and/or additional software used are included under this criterion. This criterion is needed because some of the BBPLs are web programming platforms and do not need installation, which is an advantage as it guarantees easier and wider accessibility.
- Language support: The Bulgarian language support is of great significance. This is a focal point of the present study. All the investigated BBPs and their programming platforms are adapted to the English language, and very few are supported in the Bulgarian language.
- Scientific publications

(2) Teaching materials (Table 2): The availability of a detailed help system, easy-to-test examples, and a well-structured curriculum are essential since these programming languages are intended for children's education. It is of vital importance that these materials are made available in the native language of pupils. Well-structured resources will assist children to better understand the abstract concepts used in the BBPLs. However, this article aims at showing only the availability of teaching materials as necessary elements enabling the use of BBPLs and their platforms for pupils' education. Their quality is not the subject of research in this article as they are a subjective factor and depend on the educational system in the country concerned. This category contains the following criteria:

- Curriculum
- Structured learning contents
- Video tutorials
- Text documents
- Teaching materials in pupils' native language

(3) Capabilities of programming languages (Table 2): This group includes the following criteria:

- Games: Capabilities to develop games;
- Animations: Capabilities to develop animations;
- Robotics and Drones: Capabilities to manage robotic devices or drones;
- Program sharing: Capabilities to share the created application;
- Real program code: Capabilities for visualization of a real program code.

(4) Features of the traditional programming languages (Table 3): This category is needed to determine the extent to which the language can equip the child with real capabilities in a traditional programming language. This will allow easy acquisition of the basic programming knowledge and the programming skills involving traditional languages. The criteria that fall here are:

- Data types, variable type and constants
- Arithmetic and Boolean operators
- Conditional operators
- Loops

Table 2. Analysis of some programming languages based on teaching materials and programming language capabilities

Category	Code.org	ScratchJr	Scratch	Tynker	Kodu Game Lab	Gamestar Mechanic	Hops-cotch	Alice	Snap!
Teaching materials									
Curriculum	√	√	√	√	√	√	√	√	
Structured learning content	√		√	√					
Video tutorials	√	√	√	√	√	√	√	√	
Text documents	√	√	√	√	√	√	√	√	√
Teaching materials in Bulgarian	√		√						
Capabilities of programming languages									
Games	√	√	√	√	√	√	√	√	√
Animations	√	√	√		√		√	√	
Robotics and Drones				√					√
Program sharing	√	√	√	√		√	√	√	√
Real program code	JavaScript			JavaScript, Python				Java	XML, Python
Total evaluation based on learning materials and programming language capabilities (%)	91	55	82	73	36	45	55	64	55

Table 3. Features of traditional programming languages integrated into block-based programming languages and platforms appropriate for children

Name	Code.org	ScratchJr	Scratch	Tynker	Kodu Game Lab	Gamestar Mechanic	Hopscotch	Alice	Snap!
Data types, variable type and constants	√		√	√			√	√	√
Arithmetic and Boolean operators	√		√	√			√	√	√
Conditional operators	√		√	√	√	√	√	√	√
Loops	√	√	√	√	√	√	√	√	√
Functions and procedures	√		√	√			√	√	√
Arrays and list	√		√	√				√	√
Pointers and data structures	√		√	√				√	
File handling	√	√	√	√				√	√
Units and modules								√	
Object-oriented programming	√			√				√	√
Data and database								√	√
Events	√	√	√	√	√	√	√	√	√
Tools for painting and drawing	√		√	√		√		√	√
Sound use	√	√	√	√	√	√	√	√	√
Template and sprite control	√	√	√	√	√	√	√	√	√
Input with keyboard and mouse	√	√	√	√	√	√	√	√	√
Save	√	√	√	√	√	√	√	√	√
Total value (%)	88	41	82	88	41	47	59	100	88

- Functions and procedures
- Arrays and list
- Pointers and data structures
- File handling
- Units and modules
- Object-oriented programming
- Data and database
- Events
- Tools for painting and drawing
- Sound use
- Template and sprite control
- Input with keyboard and mouse

The BBPLs that are intended for analysis with the proposed methodology were selected after a detailed overview of the literature discussed in the second section. There are other programming languages and environments that are appropriate for children.

To complete the goals set in this study several BBPEs such as Move the turtle (http://movetheturtle.com), PiktoMi (https://piktomir.ru), and Daisy the Dinosaur (http: // www.daisythedinosaur.com) among others, which are suitable for children aged 5+ and allow the creation of simple programs, were found with Google Search. The common feature shared by all of them is the lack of complex functionality that is characteristic of programming languages. Therefore, they will not be considered in this article.

We chose the programming platforms Code.org, ScratchJr, Scratch, Tynker, Microsoft's Kodu Game Lab, Gamestar Mechanic, Hopscotch: Learn to Code Through Creativity, Alice, and Snap! (Table 1). The capabilities of both the environments and their programming languages are analyzed.

The high-level primitives of Kodu allow writing of very easy computer programs. A study of various strategies for the practical application of the Kodu Game Lab within a curriculum for young children educated “into the habit of reasoning about programs” is presented in [15]. The results reported by Touretzky et al. [15] show that many of the pupils find it difficult to understand the basic logical rules when creating complex programming actions. Difficulty involving the development of mental simulation and analytical reasoning among children is noticed. In conclusion, the authors point out that in order to understand Kodu, the children should first study other simple languages such as *Scratch* or *Python*.

ScratchJr, Tynker and Hopscotch programming platforms stand out among others. What is common between them is that they can be installed on a device with a mobile operating system. The ScratchJr programming way is more intuitive and easier, but its functionalities are limited. In contrast, Tynker and Hopscotch both provide many programming options as they provide an opportunity to use different backgrounds, characters and objects, and so on.

All the studies devoted to the use of BBPLs as a programming tool for primary school children noticed that *Scratch* was the most widely used. The practical use of Scratch in the learning/teaching process is presented in [16].

Wilson et al. [17] examined the possibility of using *Scratch* to create computer games by children (4th to 7th grades) in Scotland. Their article presents pedagogical approaches to the introduction of game-based training and provides an opportunity for assessment of children's programming skills. Another study, related to the use of *Scratch* by pupils for one semester, is presented in [18]. According to Kobsiripat [18], 60 children in the 4th grade obtained the knowledge and skills to use media objects, and thereby develop their creative skills in the Scratch environment.

Based on the first three categories presented in Tables 1 and 2, it can be seen that the most suitable BBPLs for primary school pupils in Bulgaria are Code.org (91%), followed by Scratch (82%) and Tynker (73%). Scratch, followed by Alice and Kodu were the predominant programming languages in our literature survey analyses.

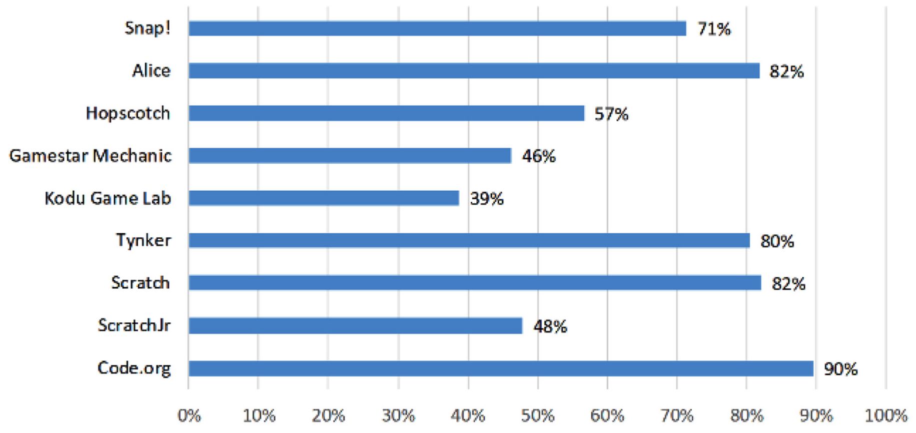

Fig. 1. Evaluation of the block-based programming languages investigated, and their programming platforms.

All of the scientific works listed in Table 1 were retrieved by the search strategy involving Scopus and Google Scholar scientific databases without any limitation concerning their publishing period.

Taking into consideration the results presented in Table 3, it can be concluded that based on the BBPLs investigated and analyzed, the closest to the traditional programming languages were Alice (100%), Code.org (88%), Tynker (88%), Snap! (88%) and Scratch (82%).

The average scores of each of the BBPLs analyzed and listed in Tables 2 and 3 are shown in Fig. 1. These values can be considered as average evaluations of BBPLs.

Based on the methodology presented and the research conducted, it can be concluded that Code.org (90%) and Scratch (82%) are the most suitable for the primary school children in Bulgaria. These programming languages also correspond to the requirements of the MES under the subject "Computer Modeling" for primary school (3rd and 4th grades) [3]. It must be noted that Alice (82%) is comparable to *Scratch*, but it is difficult to use in the initial training/learning due to the expected language barrier that children may encounter when using it.

IV. CHILDREN'S OPINIONS ABOUT SOME BLOCK-BASED PROGRAMING LANGUAGES

When talking about programming appropriate for children, one should not forget that they have a limited set of knowledge and skills. They should be considered as ordinary users, and not as programmers with special skills. Therefore, programming environments (platforms) for children should be easy to use and easy to understand. The program chunks should be easy to manage. They should use natural language for the development of the computer programs/games. The platform should provide easy-to-understand examples and multiple learning resources.

Based on the results obtained by the BBPLs surveyed in the previous section, the opinions of 19 children have been studied: 11 girls and 8 boys, aged from 8 to 10 years, and enrolled in the primary school 3rd and 4th grades in Bulgaria. The study was done with parents' permission and with the children's consent.

The children used Code.org and Scratch every day for 20 minutes for 2 weeks except for the weekends. The choice of Code.org and Scratch was based on the methodology presented and the results obtained after application in nine BBPLs.

During the experiment, the children received brief instructions for the use of the individual features and their application in simple algorithms. It must be pointed out that the children encountered additional difficulties when they used Scratch compared to Code.org. They also found it very easy to complete the simple tasks developed with Code.org because the program chunks used were in Bulgarian.

Table 4. Opinions of 8- to 10-year-old children about Scratch and Code.org

Age (yr)	Sex	Scratch		Code.org		Total
		Like	Unlike	Like	Unlike	
8	Girl	0	2	2	0	2
	Boy	0	1	1	0	1
9	Girl	2	4	5	1	6
	Boy	1	3	4	0	4
10	Girl	1	1	2	0	2
	Boy	1	3	4	0	4
Total		5	14	18	1	19

At the end of the work period, the children were asked to provide their opinions about the programming environment and the language they preferred the most. They indicated the language of preference. The results of this study are presented in Table 4. The most preferred BBPL for the children was Code.org.

Our study proved that Scratch was more difficult to understand and use by children. The presence of interactive multimedia objects in the programming platform Code.org, such as a few favorite cartoon characters proved to be one of the key factors influencing the children.

The results obtained in this investigation lead to one more conclusion: Code.org with its BBPL is the programming platform that best meets the requirements of the Ministry of Education and Science in Bulgaria for the school subject "Computer Modeling", as well as the criteria set in this article and the children's preferences.

V. CONCLUSIONS

The use of BBPLs based on drag-and-drop technology is the first step that children can take in the field of computer science. Therefore, many governments around the world introduce computer programming classes at primary school. Thus, pupils must study specific BBPLs using an appropriate programming platform. Therefore, it is important that the programming language and its environment are correctly selected, easy to understand, with adequate teaching material for free used in the teaching/learning process.

In this article, a detailed scientific analysis of the publications over the past few years related to the use of BBPLs and their development environments has been made. This analysis resulted in several generalized conclusions. (1) Children find it difficult to understand the abstract concepts used in programming languages. (2)

Children need to be at least 7 years of age to understand programming concepts. (3) The next important step in the study of programming is that children should acquire prior knowledge about formal concepts and the stages of simple algorithms. (4) The advantages of using BBPLs to develop algorithmic thinking and build the foundation for subsequent studies of traditional programming languages were also taken into consideration.

Based on the analysis of the various literary sources, groups of criteria for analysis and comparison of BBPLs were outlined. The main points of these criteria include the availability of learning materials and the features that bring the block-based languages closer to the classical programming languages. The maintenance of native language in the programming environment is also of importance for young pupils. These criteria were used to investigate nine programming languages. The evaluation identified the possibility of using some of these languages for teaching primary school pupils in Bulgaria.

According to the proposed methodology, Scratch and Code.org were selected for practical research. These BBPLs were used for two weeks by 19 children aged 8–10 years. After completion of the experiments, children's opinions were collected and analyzed and their preferences concerning the programming platform and block-based languages were discussed.

One of the advantages of this study is that the methodology used provides an independent evaluation of block-based languages, and can be used as an indicator of their application in early education. Moreover, this methodology may serve as a starting point for the development of new BBPLs.

REFERENCES

1. J. F. Pane, "A programming system for children that is designed for usability," PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 2002.
2. S. Larson, "Schools aren't teaching kids to code; here's who is filling the gap, in web," 2013, https://readwrite.com/2013/10/18/kids-learn-code-programming/.
3. Ministry of Education and Science in the Republic of Bulgaria, "Ordinance No 5 of 30.11.2015 on general education," http://www.mon.bg/?h=downloadFile&fileId=8661.
4. S. Papert, *Mindstorms: Children, Computers, and Powerful Ideas*. London: Harvester Press, 1980.
5. V. Kralev and R. Kraleva, "Methods and tools for rapid application development," *International Scientific and Practical Conference World Science*, vol. 1, no. 4, pp. 21-24, 2017.
6. R. Kraleva, V. Kralev, and D. Kostadinova, "Investigating some programming languages for children to 8 years," *International Scientific and Practical Conference World Science*, vol. 5, no. 9, pp. 4-6, 2016.
7. E. R. Hayes and I. A. Games, "Making computer games and design thinking: a review of current software and strategies," *Games and Culture*, vol. 3, no. 3-4, pp. 309-332, 2008.
8. A. Manches and L. Plowman, "Computing education in children's early years: a call for debate," *British Journal of Education Technology*, vol. 48, no. 1, pp. 191-201, 2017.
9. I. Rogozhkina and A. Kushnirenko, "PiktoMir: teaching programming concepts to preschoolers with a new tutorial environment," *Procedia - Social and Behavioral Sciences*, vol. 28, pp. 601-605, 2011.
10. J. P. Gibson, "Teaching graph algorithms to children of all ages," in *Proceedings of the 17th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, Haifa, Israel, 2012, pp. 34-39.
11. J. P. Gibson, "Formal methods: never too young to start," in *Proceedings of the Formal Methods in Computer Science Education (FORMED)*, Budapest, Hungary, 2008, pp. 151-160.
12. H. Dwyer, C. Hill, A. Hansen, A. Iveland, D. Franklin, and D. Harlow, "Fourth grade students reading block-based programs: predictions, visual cues, and affordances," in *Proceedings of the 11th Annual International Conference on International Computing Education Research (ICER)*, Omaha, NE, 2015, pp. 111-119.
13. M. Armoni, "Teaching CS in kindergarten: how early can the pipeline begin?," *ACM Inroads*, vol. 3, no. 4, pp. 18-19, 2012.
14. Y. Matsuzawa, Y. Tanaka, and S. Sakai, "Measuring an impact of block-based language in introductory programming," in *Stakeholders and Information Technology in Education*. Cham: Springer, 2016, pp. 16-25.
15. D. S. Touretzky, C. Gardner-McCune, and A. Aggarwal, "Semantic reasoning in young programmers," in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, Seattle, WA, 2017, pp. 585-590.
16. S. Dasgupta and B. M. Hill, "Scratch community blocks: supporting children as data scientists," in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, Denver, CO, 2017, pp. 3620-3631.
17. A. Wilson, T. Hainey, and T. M. Connolly, "Using scratch with primary school children: an evaluation of games constructed to gauge understanding of programming concepts," *International Journal of Game-Based Learning*, vol. 3, no. 1, pp. 93-109, 2013.
18. W. Kobsiripat, "Effects of the media to promote the scratch programming capabilities creativity of elementary school students," *Procedia - Social and Behavioral Sciences*, vol. 174, pp. 227-232, 2015.
19. F. Hermans, K. T. Stolee, and D. Hoepelman, "Smells in block-based programming languages," in *Proceedings of 2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Cambridge, UK, 2016, pp. 68-72.
20. M. Ichinco, K. J. Harms, and C. Kelleher, "Towards Understanding successful novice example use in blocks-based programming," *Journal of Visual Languages and Sentient Systems*, vol. 3, pp. 101-118, 2017.
21. D. Weintrop and U. Wilensky "To block or not to block, that is the question: students' perceptions of blocks-based programming," in *Proceedings of the 14th Annual International Conference on Interaction Design and Children*, Boston, MA, 2015, pp. 199-208.
22. S. M. Taheri, M. Sasaki, J. O. Chu, and H. T. Ngetha, "A study of teaching problem solving and programming to children by introducing a new programming language," *International*

Journal of E-Learning and Educational Technologies in the Digital Media, vol. 2, no. 1, pp. 31-36, 2016.

23. F. Kalelioglu, "A new way of teaching programming skills to K-12 students: Code.org," *Computers in Human Behavior*, vol. 52, pp. 200-210, 2015.
24. D. Weintrop and U. Wilensky, "Bringing blocks-based programming into high school computer science classrooms," in *Proceedings of Annual Meeting of the American Educational Research Association (AERA)*, Washington, DC, 2016.
25. K. D. Leidl, M. U. Bers, and C. Mihm, "Programming with ScratchJr: a review of the first year of user analytics," in *Proceedings of the International Conference on Computational Thinking Education (CTE),* Hong Kong, 2017, pp. 116-121.
26. A. Strawhacker, M. Lee, C. Caine, and M. U. Bers, "ScratchJr demo: a coding language for kindergarten," in *Proceedings of the 14th International Conference on Interaction Design and Children (IDC)*, Boston, MA, 2015, pp. 414-417.
27. X. Basogain, M. A. Olabe, J. C. Olabe, and M. J. Rico, "Computational thinking in pre-university blended learning classrooms," *Computers in Human Behavior*, vol. 80, pp. 412-419, 2018.
28. D. Franklin, C. Hill, H. A. Dwyer, A. K. Iveland, and D. B. Harlow, "Initialization in scratch: seeking knowledge transfer," in *Proceedings of the 47th ACM Technical Symposium on Computer Science Education (SIGCSE)*, Memphis, TN, 2016, pp. 217-222.
29. M. Dorling and D. White, "Scratch: a way to logo and Python," in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE)*, Kansas City, MO, 2015, pp. 191-196, 2015.
30. M. Resnick, J. Maloney, A. Monroy-Hernandez, N. Rusk, E. Eastmond, K. Brennan, et al., "Scratch: programming for all," *Communications of the ACM*, vol. 52, no. 11, pp. 60-67, 2009.
31. L. Pawloski and C. Wall, *Maker Literacy: A New Approach to Literacy Programming for Libraries*. Santa Barbara, CA: Libraries Unlimited, 2017.
32. T. Fristoe, J. Denner, M. MacLaurin, M. Mateas, and N. Wardrip-Fruin, "Say it with systems: expanding Kodu's expressive power through gender-inclusive mechanics," in *Proceedings of the 6th International Conference on Foundations of Digital Games*, Bordeaux, France, 2011, pp. 227-234.
33. M. B. MacLaurin, "The design of Kodu: a tiny visual programming language for children on the Xbox 360," *ACM SIGPLAN Notices*, vol. 46, no. 1, pp. 241-246, 2011.
34. R. Joseph and J. Diamond, "iDesign designing and implementing a culturally relevant game-based curriculum," in *Culture, Learning, and Technology: Research and Practice*. New York, NY: Routledge, 2017, pp. 151-164.
35. R. O. Welsh, "School hopscotch: a comprehensive review of K-12 student mobility in the United States," *Review of Educational Research*, vol. 87, no. 3, pp. 475-511, 2017.
36. D. Slater, E. Brown, and W. Dann, "Mapping Alice curriculum to standards: a BOF for the Alice community," in *Proceeding of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, Seattle, WA, 2017, pp. 731-731.
37. L. Werner, J. Denner, M. Bliesner, and P. Rex, "Can middle-schoolers use Storytelling Alice to make games?: results of a pilot study," in *Proceedings of the 4th International Conference on Foundations of Digital Games (FDG)*, Orlando, FL, 2009, pp. 207-214.
38. C. Kelleher, R. Pausch, and S. Kiesler, "Storytelling Alice motivates middle school girls to learn computer programming," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, San Jose, CA, 2007, pp. 1455-1464.
39. S. Cooper, W. Dann, and R. Pausch, "Alice: a 3-D tool for introductory programming concepts," *Journal of Computing Sciences in Colleges*, vol. 15, no. 5, pp. 107-116, 2000.
40. D. Garcia, L. Segars, and J. Paley, "Snap! (build your own blocks): tutorial presentation," *Journal of Computing Sciences in Colleges*, vol. 27, no. 4, pp. 120-121, 2012.

Radoslava Kraleva

Radoslava Kraleva is an assistant professor of Computer Science at the Faculty of Mathematics and Natural Sciences, South-West University "Neofit Rilski", Blagoevgrad, Bulgaria. She defended her Ph.D. thesis on "Acoustic-Phonetic Modeling for Children's Speech Recognition in Bulgarian" in 2014. Her research interests include child-computer interaction, speech recognition, mobile app development and computer graphic. She is an editorial board member of *the International Journal of Advanced Computer Research* and *Perspectives of Innovations, Economics and Business*. She is a reviewer of *the International Journal on Advanced Science, Engineering and Information Technology* (iJET), *Computer Standards & Interfaces*, *Journal of King Saud University - Computer and Information Sciences*, and many others.

Velin Kralev

Velin Kralev is an assistant professor of Computer Science at the Faculty of Mathematics and Natural Sciences, South-West University "Neofit Rilski", Blagoevgrad, Bulgaria. He defended his Ph.D. thesis in 2010. His research interests include database systems development, optimization problems of the scheduling theory, graph theory, and component-oriented software engineering. He is an Editorial Board member of the *International Journal of Advanced Computer Research* (IJACR).

Dafina Kostadinova

Dafina Kostadinova is currently an associate professor at the Department of Germanic and Romance Studies at the Faculty of Philology of the South-West University "Neofit Rilski" since 2000. She has taught General English, Business English, Translation, Specialized Translation, Contrastive Analysis, Academic Writing, and Introduction to General Linguistics to students at the Faculties of Philology, Economics, and Pedagogy. In 2012, Kostadinova defended her Ph.D. thesis on "Structural Interferences in the Production of English by Bulgarians". She co-authored a textbook titled Specialized Translation (Selected English and Bulgarian Texts for Translation) published in 2015. She is a member of the Editorial Board of the journal, *Orbis Linguarum*, published by the South-West University "Neofit Rilski".