education sciences

Article

A Cross-analysis of Block-based and Visual Programming Apps with Computer Science Student-Teachers

Piedade João [1,*], Dorotea Nuno [1], Sampaio Ferrentini Fábio [2] and Pedro Ana [1]

[1] UIDEF, Institute of Education, University of Lisbon, 1649-004 Lisbon, Portugal
[2] Postgraduate Program in Informatics, Federal University of Rio de Janeiro, 21941-901 Rio de Janeiro, Brazil
* Correspondence: jmpiedade@ie.ulisboa.pt

Received: 17 May 2019; Accepted: 10 July 2019; Published: 12 July 2019

Abstract: In the last few years, it has been pointed out that teaching programming is a strong strategy to develop pupils' competences in computational thinking (CT). In the Portuguese context, the curriculum changes in 2018 made programming and CT compulsory for every pupil in primary and secondary education. Nowadays, there is an information and communication technology (ICT) subject, taught by a computer science teacher in each school grade. In Portugal, to become a computer science teacher in primary and secondary education, it is compulsory to have a master's degree in computer science education. This article reports on a pedagogical activity developed with student-teachers of a Master in Teaching Informatics at the University of Lisbon. Within the activities of the master's program, we developed a cross-analysis of the core characteristics of 26 block-based and visual programming applications (apps) used to teach computational thinking and programming in school classes. In order to organize the analysis, a framework with several dimensions was developed and used by student-teachers to register the characteristics of each app. The product of this work is a comparative matrix mapping the core characteristics of each of the 26 apps that student-teachers used to select the most appropriate one for teaching programming and computational thinking according to each grade, age group and other characteristics.

Keywords: computational thinking; programming; visual programming applications; computer science education; cross-analysis

1. Introduction

Computational thinking (CT) and programming have become a trending topic in an education context, as well as a very important trend in educational research. This concept was first defined by Wing [1], who stated that CT is about "solving problems, designing systems, and understanding human behaviour, by drawing on the concepts fundamental to computer science" (p. 6). Thus, CT should entail problem-solving activities designed for every citizen, including children, and not only for computer scientists, as an important way to understand the cognitive process associated with computer science. The same author pointed out that "computational thinking is using abstraction and decomposition when attacking a large complex task or designing a large complex system" (p. 32). The core principles of CT were inspired by the powerful constructionism ideas proposed by Papert [2,3]. One of these ideas concerns the use of "objects-to-think-with", where learners become aware of their own knowledge through construction and interaction with real world artefacts.

Over the last few years, computational thinking, computer science and programming have been integrated into the primary and secondary school curriculum in many countries around the world [4–8]. Normally, they are taught either as independent subjects or as a way to integrate the learning activities of other subjects [5]. According to the first perspective, for example, in Portugal, the latest curriculum changes made computer science and programming compulsory for every pupil in primary and secondary education [9]. The National Curriculum integrates a subject in the computer science area, between the 5th and 9th grades, taught by a computer science teacher. In primary grades, from the 1st to the 4th grade, computer science and information and communication technology (ICT) are part of other disciplinary areas (maths, Portuguese, science) and curricular guidelines are supposed to be implemented by the teacher in a multidisciplinary way. The curriculum states that pupils should learn about computational thinking, design algorithms, programming with blocks applications, programming robots and other tangible objects, and use digital technologies to create and innovate [9].

National curricular guidelines consider computer science standards developed by international institutions such as Association for Computing Machinery (ACM), Computer Science Teacher Association (CSTA), International Society for Technology in Education (ISTE) and the Computing at School Group (CAS). In 2011, the CSTA task force defined the strands and guidelines for computer science standards, organized in five core concepts: (i) computational thinking; (ii) collaboration; (iii) computing practice and programming; (iv) computers and communications devices; and (v) community, global, and ethical impacts. Recently in 2017, CSTA K–12 standards were revised, restructuring the five core concepts, and presenting a new definition: (i) computing systems; (ii) networks and the internet; (iii) data and analysis; (iv) algorithms and programming; and (v) impacts of computing. Both versions were organized in three levels and, for each one, a set of statements was defined.

In terms of teacher training, to become a computer science teacher at primary and secondary schools in Portugal, it is compulsory to hold a master's degree in teaching. The students should hold a first degree in informatics or computer science, and after the two years of the master's program in teaching, they complete their initial teacher training and are entitled to teach computer science or informatics. During the first year of the master's program, they learn about education sciences, research methods, curriculum and assessment, and didactic of informatics. In the second year, students are immersed as teachers in a real classroom where, for about two months, they must plan and implement a set of classes.

The experience reported in this article was developed with 11 student-teachers of the Master in Teaching Informatics at the University of Lisbon. In the Didactic of Informatics Course, students are involved in several pedagogical activities in the areas of computer science, computational thinking, programming, block-based and visual programming languages, as well as methodologies and strategies for teaching. We aimed to analyse the core characteristics of block-based and visual programming applications that have been used to teach programming concepts to children. To do so, a multidimensional matrix framework was created, and a set of criteria were then defined to support and organize the analysis process.

2. Background

In this section, we first discuss the major problems and difficulties related to learning to program, particularly by novice programmers or primary and secondary pupils. As previously referred to, nowadays, pupils are starting to learn programming early in schools and it is important to identify and to understand their difficulties and select the most adequate programming tool to support their learning. The literature has recurrently shown that block-based and visual programming environments are powerful tools to support novices and pupils in the process of learning to program. Thus, in the second part of this section, we analyse some aspects related to these environments and how they could help pupils overcome the difficulties of the traditionally text-based languages.

2.1. Major Problems in Learning Programming

Programming is a subject area that traditionally involves concepts related to computational, algorithmic and logical thinking, identifying problems, design and coding solutions, understanding the syntax, semantics and complexity of languages, and mastering a set of programming paradigms. Although this is an important area of computer science, it is also very difficult for novices to learn [10–12]. Through a literature review on this topic, we aim to organize and systematize the main difficulties into four dimensions of analysis: (i) subject and complexity of languages; (ii) technologies and applications; (iii) teachers and teaching methodologies; and (iv) pupils' skills.

2.1.1. Subject and Complexity of Languages

Computer programming entails an understanding of theoretical background and practice of a series of semantic and syntactic knowledge, programming skills and algorithmic thinking. These characteristics make learning complex and difficult for many pupils [10,13–15].

The nature of programming involves high levels of abstraction, generalization and critical thinking, and has very complex and extensive syntax with many syntactic details to memorize [1]. Another problem is the programming language chosen to teach computer science pupils. In many cases, the programming language used for teaching is not actually designed for teaching but instead for professional and normally more complex purposes [16], and it has a sharp learning curve.

2.1.2. Technologies and Applications

There are many aspects to consider when choosing the most adequate programming tool to be used for teaching programming, particularly to newcomers [15]. Teachers have a dilemma, whether they should select a professional tool, mostly text-based languages, or select a more pedagogical tool, like block-based programming environments. Professional applications are an authentic technological tool that can support learning [16,17], but they are also usually too complex for many of the pupils [10]. In addition, teachers tend to choose the programming language and tool that they master well enough and are confident to use in the classroom. However, some of these languages are not the best choice for the first approach to pupils' programming learning. In this sense, Sáez-López et al. [8], quoting Maya et al. [18], state that "some teachers believe that the only computing experiences are those related to programming languages, such as Java, or C++" (p. 3).

According to Kelleher and Pauschin [19], there are many programming tools that could be of interest to support a variety of styles and paradigms of programming, programming constructs, coding actions or coding representations, while also offering different levels of learning ability.

2.1.3. Teachers and Teaching Methodologies

Teachers' beliefs, practices and, especially, methodologies that they select for use in programming instructional activities, have a strong impact on novice programming pupils. "Alternative methodologies provide educators with the opportunity to deal with the complexity of each formal or informal class" [15] (p. 5). Following this line of thought, the same authors point out that creative instructional approaches based on constructivism and constructionism principles could be very important for involving pupils actively in their learning process. However, Groover and Pea [20] mentioned that guidance activities are important for the pupils' understanding of cognitive aspects of computing practices, improving their capacity to reflect on their experiences.

The importance of teacher practices is referred to by Gomes and Mendes [10] (p. 3) when they state: "teachers are sometimes more concentrated on teaching a programming language and its syntactic details, instead of promoting problem solving using a programming language". In fact, sometimes teachers tend to give more importance to the semantic and syntax of a specific programming language (e.g., Java, C, C++, Python), which is related to their beliefs and confidence levels in teaching computing. Mostly, teachers' strategies and approaches do not provide strong support for all the learning styles and tend to ignore their previous background [10,15].

In addition, teachers should support pupils in learning and achieving their goals due to the complexity of languages and concepts. This support and feedback can improve pupils' understanding of various difficulties in computer science concepts and practices [21–24].

2.1.4. Pupils' Skills

Several studies have recurrently identified major difficulties in learning programming, related to the pupils' skills, background and motivation [10,11,25]. Mostly, pupils have difficulties in problem solving activities with logical reasoning and mathematical thinking; they use inadequate studying methods and do not work hard enough to develop programming competences [10,15].

Interest and motivation to learn programming are important variables that could improve a pupils' learning and achievement and, for some authors [11,24], the absence of these characteristics can drive many pupils to give up computer science [26,27].

According to Robins et al [28], cited by Cheng [29] (p. 362), "one of the major challenges facing novice programming pupils was that they knew the syntax and semantics of each single statement in the programming language, but they did not understand how to combine the statements to create a valid computer program". Another problem in learning programming is the pupils' difficulty to translate a solution into an algorithm and coding it to solve a problem [25,29,30]. For many pupils it is very difficult to understand the knowledge and skills necessary to create a computer program [31].

Another important issue to discuss is gender in learning programming. Previous research has pointed out that female pupils are generally less confident in their ability to learn computer science and have a less positive attitude towards programming [32–35]. However, in some studies, the differences between male and female pupils were not significant and could not be generalized to all pupils [29]. In the study of Cheng [29] with 431 pupils (296 boys and 135 girls) from 38 primary schools, the author identified gender differences in pupils' perceptions towards the visual programming environment and on their intention to use it. Therefore, although of low significance, gender is still an important variable to consider in computer science education to increase the number of female pupils learning computing.

2.2. Visual and Block-based Programming Environment

The visual programming environment (VPE) and block-based programming environment have been referred in many research studies as important tools to learn programming and to solve many problems taught in primary schools. In fact, visual programming with blocks is a great support in introducing programming and saves pupils from the difficulties of traditionally complex text-based languages [27,36].

In 1990, Myers [37] defined visual programming as "any system that allows the user to specify a program in a two (or more) dimensional fashion (p. 2)" and visual languages as "all systems that use graphics, including visual programming and program visualization systems (p. 3)". According to Myers's perspective, in 'visual programming' applications, the graphics are used to create the programs, and in 'program visualization applications', the program is created in the traditional text-based way and the graphics are used to visualize the program execution and outputs. In this paper, we understand VPE to be a block-based or visual text-based or hybrid (block + text) programming environment, where the focus is on the core characteristics of each environment in order to improve the pupils´ learning.

Block-based programming is one form of visual programming and refers to a software or application where pupils can select the appropriate blocks of code and snap them together to create a program [29]. In this kind of introductory programming tool, blocks of coding are organized in coloured categories that can help pupils understand how to select the correct block, thus lowering some barriers to programming [38]. Programming in these environments takes the form of drag-and-drop blocks that can be snapped together to create programs; if two blocks cannot be joined to form a valid syntactic statement, the interface prevents them from snapping together [39].

The most popular of these environments used in early programming education is Scratch, but there many others available like App Inventor, Alice, and PencilCode. While these VPEs differ in many aspects and characteristics, they have similar programming principles and purposes.

This kind of programming environment is recognized by different researchers as an interesting way to preventing errors and reducing the cognitive overload to novice programmers [40]. Most common syntax errors found in text-based languages are avoided or they cannot occur [41].

However, according to Bau et al. [40], " ... plain text as the representation of program text is still the norm for proficient and professional programmers and the accepted educational goal for programming instruction in schools in many countries (p. 1)", which means that students will face a transition from blocks to text-based programming somewhere in their school life, but the move to new text-based environments is still a challenge.

Another problem attributed to block-based programming is that it becomes difficult to understand/debug programs when they start having many blocks of code. The diversity of colors of the blocks and the need to move the page from top to bottom (and vice versa) in search of execution errors, ends up confusing the programmers [21].

In recent years, a new type of environment has emerged, joining the two models of programming, block-based and text-based. In these environments, pupils have the possibility to switch between the block-based and text-based versions of their programs. This strategy tends to be used by newcomers when using these environments to learn to program [42,43].

Using these VPEs, pupils can learn and practice the main concepts of programming, such as instructions, containers (variables, constants and lists), conditional statements, loops, logical operators and input/output data. In addition, some studies concluded that block-based programming, together with efficient pedagogy, can promote a strong path for developing computational skills and prepare pupils for computer science education [44].

Nowadays, there are many visual programming environments available as tools to introduce programming in primary and secondary schools. All these applications share the same principles of programming, but different core characteristics, forms of programming, system requirements, and types of processing, among others. In this article we report a cross-analysis of 26 selected VPEs (for example Scratch, App Inventor, Alice, Pencil Code, Tynker, Kodu, Code Studio, Thunkable, m-Block) focusing on their pedagogical potentialities and characteristics.

3. Research Purpose

The research purpose of the pedagogical activity proposed to student-teachers was both to carry out a cross-analysis of core characteristics of a set of 26 applications of visual and block-based programming, as well as to develop the student-teachers' skills in the pedagogical analysis of programming environments. The objectives were to:

- Identify the typology of the programming language;
- Identify the typology of processing or execution;
- Recognize the core programming concepts that can be taught;
- Evaluate the potentialities in terms of deepening programming learning;
- Analyse the possibility of connecting to other areas of knowledge;
- Analyse the typology of the programming;
- Analyse the possibility of programming multiple scenarios, actors and sprites;
- Evaluate their suitability by age or grade;
- Analyse the functionalities available for teachers;
- Identify the technical system requirements and price.

As output of the analysis developed, in Appendix A Table A1 we present a matrix framework that maps the main characteristics of the VPEs, which could be used to support the teachers' choices of the best application to teach programming and computational thinking in different contexts.

4. Methods

4.1. Context

The experience described in this article was developed in the Didactics of Informatics Course in the Master in Teaching Informatics. Didactics of Informatics is the first of five didactics units and, in the first semester, it takes place in a face-to-face session of three hours for about 12 weeks. This course aims to introduce future teachers to the study of pedagogical and didactic aspects of teaching computer science in primary education. The syllabus of this course is organized around three core themes: (i) Initiatives related to ICT and computer science in the curriculum in national and international contexts; (ii) computational thinking and programming in primary education; and (iii) programming of tangible objects and robotics.

In the first theme, students study ICT in the curriculum and computer science in school initiatives around the world comparing them to national initiatives and curriculum guidelines. The second theme is centered on the pedagogical aspects of computational thinking, the programming concepts, and visual or block programming environments. Finally, in the third theme, students study programming of tangible objects, and robotics, as a pedagogical way to develop computational thinking and programming skills. The pedagogical and research activity presented in this article was carried out within the second theme and was implemented with student-teachers of the 2018–2019 class. Two pedagogical tasks were proposed to students in the second theme: (i) a cross-analysis of visual or blocks programming environments, to be developed in pairs; and (ii) a peer-teaching workshop preparation and implementation on the environment they analyzed. In the first task, pupils examined the characteristics of the programming environments using a framework previously defined by the teachers of the course. This activity aimed to develop knowledge about the characteristics of a set of visual programming applications that can help teachers in selecting the most appropriate environment to teach programming considering, for example, pupils' age, grade, and objectives of the lesson.

4.2. Participants

This activity involved 11 Master in Teaching Informatics students who attended the Didactics of Informatics Course. The group of students was predominantly female (9) and aged between 22 and 56 years old. All students had a previous degree in computer science and were willing to become entitled as computer science and informatics teachers in primary and secondary schools.

4.3. Procedures

The pedagogical activity was designed to be implemented for four weeks and aimed to involve the students in the analysis of visual programming environments (VPE's). For that purpose, an analytic framework was created by the teachers and 26 VPEs were selected. The selection of VPEs was decided taking into account the programming typology (block-based, text-based or both), the programming concepts that can be taught, the possibility of mobile and tangible objects programming, and the adaptation to the different age groups of pupils (preschool, k1 to k12). The VPE's list was generated by using the Google search engine with a set of query terms like "block-based programming apps", "block coding apps", "coding apps for kids", "visual programming apps", "coding game apps" and "coding apps for beginners". We selected the most popular VPEs presented in the first five pages searched. In addition, we selected some VPEs included in the "Hour of Code" initiative and some of the most popular for educational purposes like Scratch, Alice, App Inventor and Snap!

VPEs were randomly distributed among the students using a randomizing online application. In the pedagogical activity, student-teachers could choose to work in groups or individually. They worked in five groups of two elements and three students opted to work individually. However, during the activities, one group of two students left the class. As a consequence, the authors decided to carry out the analysis of the three environments that would be the responsibility of these students (Alice, Snap! and Pencil Code). This decision was based on the fact that these three VPE´s are widely used to introduce programming to pupils.

Each group was responsible for the analysis of four VPEs, two students analysed two VPEs each, and one student analysed three VPEs. The framework was made available to the students using an online Google spreadsheet, which allowed them to have access to peer analysis.

At the end of the activity, each group presented and discussed the main characteristics and functionalities to the classmates and the matrix of analysis was validated by the teachers.

4.4. Categories of Analysis

The multidimensional matrix framework designed to support the cross-analysis of the visual programming environments was organized in five main dimensions. To identify these dimensions, we took into account the Portuguese curricular guidelines for computational thinking and programming in primary and secondary education (k1 to k12), the international standards defined by ISTE and CSTA, the pedagogical adequacy and the system requirements. Each of these dimensions has categories that consider the objectives defined for the pedagogical activity as well as some of the characteristics mentioned in the literature. Accordingly, the dimensions and categories were defined and organized in the following topics and tables. In all items with yes/no options, the yes option is registered in the table with an X.

Programming Aspects: This dimension (Table 1) presents some aspects provided by the environment in terms of programming.

Table 1. Categories of analysis in the programming aspects dimension.

Categories	Possibilities
Programming (typology)	Blocks; Lines of code; Both
Language	Symbolic; Text; Both
Translate blocks to text	Yes; No
Programming structures	Arithmetic operators; Relational and logical operators; Variables and constants; Decision structures; Loops; Functions.
Type of processing	Sequential; Parallel
User can evolve in programming skills	Yes; No

The dimension allows one to analyse the core programming aspects of the environments considering the typology of programming (block-based, text-based or both) and the programming concepts that can be taught. During the process of analysis, a new type of environment has emerged, joining the two models of programming, block-based and text-based, that can translate blocks to text and help students in the transition for text-based languages. It was only considered the category “translate blocks to text” because none of the analyzed applications allow the conversion of text to blocks. It is also important to analyse the possibilities that the environment allows in terms of evolution in learning programming, for example, using the environment, the student can solve more complex problems, create new scenarios and learn new programming concepts. In the "user can evolve in programming skills" category, we intend to analyze the potential of each application to learn more complex programming concepts, such as nested cycles, chain decision structures, lists, functions and procedures, and recursion.

Requirements: This dimension (Table 2) provides information about the computer requirements needed to run the software/environment.

Table 2. Categories of analysis in the requirements dimension.

Categories	Possibilities
Requires continuous internet connection	Yes; No
Requires Login or Classroom Code for younger children	Login; Classroom Code
Operating system	Browser; Windows; IOS; Android; Mac; Other
Equipment	Computers; Tablets; Smartphones
Available in Portuguese	Yes; No
Available in English	Yes; No
Price	Free; Partially free; Paid

The dimension intends to analyse each programming environment considering the system requirements and the equipment necessary for their use for programming activities, as well as the language and the costs. The costs presented in the price category were calculated at the date of paper publication and represent the monthly value or total value of the app. The cost of Micro: bit only includes the hardware, as the software is free to use.

User Interaction: this dimension (Table 3) presents aspects related to the interface provided by the software.

Table 3. Categories of analysis in the user interaction dimension.

Categories	Possibilities
Stimulating environment for kids	Yes; No
Activities (typology):	Pre-defined; Open; Both
Scenarios	None; Only one; Multiple; Creation of new ones
Multimedia	Audio; Image; Video; Pre-defined by the software
Actors/sprites	One; Many; Creation of new sprites
Programming actors/sprites	None; One; Two or more
Creating drawings	Yes; No
User can enter data	Yes; No
Stimulating environment for kids	Yes; No
Activities (typology):	Pre-defined; Open; Both

In this dimension, the VPEs are analysed considering the possibilities of interaction with the user (pupils) and the typology of the activities. It is important to analyse the quality of the environment that can motivate the pupils to learn programming, and the possibilities of creating new scenarios, new actors and sprites, and integration of the multimedia elements. When using data types, all applications only allow the use of the type appropriate to the actions and variables used. In applications whose programming typology is textual, the application visually alerts that there is an error of undue use of the data type. In the category "stimulating environment for kids", we intend to analyse the capabilities of each application to keep the user motivated to continue to use it. The main capabilities considered are design, user experience, more complex and stimulating challenges as the user evolves and gamification features.

Programming Tangible Objects: This dimension (Table 4) tells us about tangible objects that can be programmed by the user.

Table 4. Categories of analysis in the programming tangible objects dimension.

Categories	Possibilities
Programming robots	Yes; No
Programming drones	Yes; No
Programming for mobile devices	Yes; No

Some of the block-based environments offer the possibilities of programming tangible objects. As tangible objects, we consider robots with different typologies, drones and mobile devices (e.g., smartphones). This category intends to analyse the VPEs considering their capacity for programming these objects.

Pedagogical Adequacy: This dimension (Table 5) presents the age adequacy of the VPE and the tools for teachers to monitor pupils' progress.

The choice of the adequate VPE to teach programming can improve the pupils learning and achievement. Thus, the pedagogical characteristics are relevant aspects to consider when teachers choose one of these characteristics. For example, some VPEs are more suitable for the pre-school pupils who still cannot read and write, and others more suitable for 7th to 9th graders who can use a hybrid environment (blocks + text). In the same way, some VPEs are more interesting and suitable for solving articulating programming problems with other curriculum domains. It is also relevant to analyse the tools related to the possibilities for teachers to create and manage groups, monitor the activities developed by pupils and the possibilities to provide feedback to them.

Table 5. Categories of analysis in the pedagogical adequacy dimension.

Categories	Possibilities
Age/ school grade suitability	Pre-school; k1–k4; k5–k6; k7–k9; k10–k12
Adequate to articulate with other curricular domains	Area of Society and Citizenship; Area of Expression and Communication (motor abilities; artistic abilities; oral and Written abilities; mathematical abilities); K1–k12; Mathematics; Language; English; Sciences; History; Geography; Arts and Physical Education; Society and Citizenship;
Tools for teachers	Manage or create groups; Monitoring activities; Share with others; Tutorial and Lesson Plans;

After the definition of the first version of the framework, we discussed the dimensions and criteria with three computer science educators to validate the core dimensions and to improve the framework quality. The results of this validation process were important to redefine some dimensions and to clarify the detail of the core criteria. After that, the framework was tested with three types of VPEs (block-based, text-based and both). The final version of the framework was presented and detailed to the student-teachers prior to the beginning of the evaluation of the environments.

5. Findings

The findings of this activity were organized into two outcomes presented by the student-teachers. The first outcome was the matrix that includes the analysis of the main characteristics of the VPEs; the second was the design and implementation of a workshop of one of the environments, analysed by each group through a peer teaching activity in the classroom.

According to the main goals of the activity, we present the list of the 26 VPEs analysed and a short description including the URL for the online support webpage in Table 6. As stated in Section 4.3, the VPEs were selected by the teacher of the Didactics Course and presented to the students as part of the activity proposal.

The results of the analysis process of the VPEs were organized in each dimension of the framework described in Section 4.4. For an integrated view of the results, the matrix framework with the complete analysis is presented in Appendix A Table A1 and is available online on http://ftelab.ie.ulisboa.pt/framework.pdf. This matrix presented the core characteristics of each of 26 VPE's and allows computer science teachers to choose the most suitable environment according to pupils age and grade, pedagogical aspects, programming concepts to teach, user interaction possibilities, system requirements and programming tangible objects.

Table 6. List and short description of the 26 visual and block programming environments (as mentioned in Section 4.3, the last three environments were analyzed by the authors of this paper).

Visual or Block Programming Environment	Short Description	Available in
CodeMove PT	Environment providing games with different levels where, as a team, students have to program to achieve the goals. This app was developed by the Portuguese Movement "CodeMove PT" to improve the programming skills of Portuguese students.	https://codemove.pt/
Run Marco (All Can Code)	An adventure game with different levels through which you learn to code in a similar way to Hour of Code.	https://www.allcancode.com/hourofcode
Bee-Bot Emulator	Emulates the bee-bot robot on the screen. You can have different scenarios to control the Bee-bot, basically using four commands (forward, backward, left, right).	https://www.bee-bot.us/emu/beebot.html
Blockly Games	Series of games that teach programming to children without prior experience in computer programming.	https://blockly-games.appspot.com/
Cargo-Bot	A set of puzzle games that challenges users while helping them to learn programming concepts.	https://itunes.apple.com/pt/app/cargo-bot/id519690804 https://www.microsoft.com/pt-br/p/cargobot/9nblggh4r05c
Code for Life Rapid Router	Uses a coding game to teach the first principles of computer programming that are covered in the National Computing curriculum (UK).	https://www.codeforlife.education/
Code Monkey	Uses line programming (CoffeeScript) to teach the basics of programming to children while they play a game. Students can apply their acquired skills to develop their own games.	https://www.playcodemonkey.com/
Code Studio	Environment providing games with different levels where students have to program to achieve the goals.	https://studio.code.org/courses
Lego Bits and Bricks	A simulated robot has to solve some challenges using a visual language, similar to ScratchJr.	https://www.lego.com/en-us/kids/games/bits-and-bricks-2ca484b751a946559fe6ebf0ecb10e66
Kodu	Children can create games on the PC, via a simple visual programming language. There is a community on the web which could be used to share programs.	https://www.kodugamelab.com/
Lightbot	A simulated robot has to be programmed in order to solve different puzzles.	http://lightbot.com/
App inventor	Visual programming environment that allows users to build fully functional apps for smartphones and tablets.	http://appinventor.mit.edu/explore/
Micro:bit	Computer board and IDE environment created by the BBC to learn programming using a visual language or JavaScript.	https://microbit.org/
Kodular	A programming environment based on App Inventor where you can create mobile apps. Experts in programming can use its IDE and develop apps in Java.	https://www.kodular.io/
Thunkable	An environment to create mobile apps using a visual programming language.	https://thunkable.com/
mBlock	An IDE environment where you can learn how to code (using visual code blocks), exploring the objects already available or creating new ones, and program robots and drones.	http://www.mblock.cc/
Scratch	An IDE environment where you can learn how to code (using visual code blocks), exploring the sprites already available or creating new ones.	https://scratch.mit.edu/
ScratchJr.	A simplified version of Scratch developed for young children.	https://www.scratchjr.org/
The FooS	Young students solve some puzzles using a visual computer language.	https://codespark.com/hour-of-code
Tynker IDE	An IDE environment where you can learn to code (using visual code blocks), exploring the games already available or creating new ones.	https://www.tynker.com/ide/
Tynker Hour of Code	A subset of Tynker IDE.	https://www.tynker.com/hour-of-code/
Code Combat	A game with different levels that teaches programming (typed code) to learners of all ages. The community can add new features to the environment.	https://codecombat.com/
Coding with Chrome	Google project providing an educational coding environment (IDE) that runs in the Chrome browser and offline. Users can create programs with output for Logo Turtle and/or toys such as the Sphero, SPRK+, mBot and Lego Mindstorms.	https://codingwithchrome.foo/
Pencil Code	Pencil Code is a collaborative programming site for drawing art, playing music, and creating games. It is also a place to experiment with mathematical functions, geometry, graphing, webpages, simulations, and algorithms.	https://pencilcode.net/
Alice	Alice is a block-based programming environment that makes it easy to create animations, build interactive narratives, or program simple games in 3D.	https://www.alice.org/
Snap!	An IDE environment where you can learn how to code (using visual code blocks), exploring the sprites and actors already available or creating new ones.	https://snap.berkeley.edu/site/

6. Discussion and Conclusions

This article presented the work developed by 11 students in the Didactics of Informatics Course, as part of a Master in Teaching Informatics provided by the Institute of Education of the University of Lisbon.

Throughout the course, the students had to evaluate a set of visual programming environments, generating a matrix that mapped the main characteristics of 26 VPEs. We believe this matrix framework can be useful to help teachers choose which environments best fit their pedagogical activities to teach pupils about programming and computational thinking concepts.

In fact, a relevant factor to mitigate the pupils' difficulties in learning programming is this selection of an environment that best fits teaching code, depending on the different ages and backgrounds [15,18]. The use of an adequate VPE can improve pupils' learning and outcomes and provide a strong way to develop computational thinking skills and creativity. However, we cannot ignore the teacher´s knowledge about the software chosen as well as the pedagogical strategies that they feel most comfortable to work with.

All the environments analysed share a common core of characteristics and principles but preserve some different features, which are related to pupils' age or grade, pedagogical adequacy, tools for teachers and possibilities for user interaction. For example, to develop programming activities with preschool pupils, currently the best choices are Scratch Jr., Lego Bits and Bricks, Bee-bot Emulator and some activities of Tynker Hour of Code. These VPEs have a simple and appealing design and have an easy task accomplishment, despite having a good challenging level. Moreover, they allow one to work with very basic concepts of programming, associated with curricular elements of pre-school education. Finally, these programming environments use blocks with symbolic language to allow pupils to create programs, even if they do not know how to read and write.

Most of the VPEs selected are free or partially free and work in many platforms and equipment. The majority of them are web-based and need a continuous internet connection, which may be a problem in some educational contexts.

In terms of tools for teachers, some VPEs provide a set of resources and tools to support teacher's practices as creation of groups, support tutorials, activities monitoring, assessment and feedback tools and sharing options.

According to the analysis, currently the most appropriate and flexible environments to learn programming concepts would be Scratch, Coding with Chrome, Tinker IDE, m-Block and Code Monkey. The analysis showed us that Scratch presents characteristics that make it an excellent VPE to learn basic concepts of programming as well as to improve the development of computational thinking skills. Recent studies have pointed out that Scratch is an important tool to learn programming, especially for novice programmers [8,38,45]. For programming activities with robots, teachers can choose from six VPEs, of which the Micro:bit and mBlock stand out as good alternatives. Depending on robot typology and architecture, Tinker IDE allows one to program drones, while App Inventor, Thunkable and Kodular allow the development and programming of applications for mobile devices. Regarding high school students, Code Combat is the best solution to work on advanced programming concepts in textual and lines of code, as the language used is Python or JavaScript and the application presents challenging activities in a game environment. Analysis shows that Alice is an interesting and stimulating environment to introduce programming in k12 education, which engages pupils in 3D game-based programming. In fact, Alice has been referred to as an excellent tool to introduce programming to pupils [46,47] and to support the transition to text-based languages like Java [48].

Moreover, it was possible to identify that many of the VPEs' programming activities use games and puzzles, adopting gamification methodologies of problem solving. This type of strategy has been pointed out as relevant for the design and implementation of programming activities for pupils with different ages and backgrounds [49].

This work, in particular the cross-analysis matrix, is relevant in different contexts. Specifically, it can be used in computer science teacher education by new groups of student-teachers to support their decisions about the choice of the best tool to teach programming in certain contexts. Alternatively, it can be used by in-service teachers to support their decisions about the tool to teach in their programming classes, or it can even be used by other professionals and institutions with interest in this field.

To conclude, we consider that, despite the possible limitations of the analysis, this activity systematized a set of relevant characteristics of the selected VPEs that can help to understand their potentialities and limitations.

7. Future Work

The development of this activity provided a framework tool that can allow future-teachers to choose appropriate VPEs to teach programming and computational thinking. As future work, it would be relevant to examine how the student-teachers will use this framework in their future practices in the classroom. It is also relevant to analyse in more detail the use of the VPEs in k–12 education through a literature review, focusing on computer science concepts and pedagogical aspects.

Author Contributions: The authors contributed equally to this work.

Funding: This article was prepared within the Project Technology Enhanced Learning @ Future Teacher Education Lab funded by Fundação para a Ciência e a Tecnologia I.P. under contract PTDC/MHC-CED/0588/2014.

Acknowledgments: The authors would like to thank Ana Pires, Ana Rodrigues, Anabela Morouço, Alexandrina Gonçalves, Andrea Resendes, Dália Pereira, Diogo Soares, Joana Pardal, Jorge Silva, Odette Paulo and Rita Martins, Master in Teaching Informatics students of the Institute of Education of Lisbon University, for their work and support in analysing the visual programming applications reported in this article.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

Table A1. A Cross-analysis of 26 Block-based and Visual Programming Apps.

Categories		**CodeMove PT**	**Run Marco (All Can Code)**	**Bee-Bot Emulator**	**Blockly Games**	**Cargo-Bot**	**Code for LifeRapid Router**	**Code Monkey**	**Code Studio**	**Coding With Chrome**	**Kodu**	**Lego Bits and Bricks**	**Lightbot**	**app Inventor**	**micro:bit**	**Kodular**	**Thunkable**	**mBlock**	**Code Combat**	**Scratch**	**ScratchJr**	**The FooS**	**TynkerHour of Code**	**Alice**	**Tynker IDE**	**Pencil Code**	**Snap!**
Programming Aspects																											
Programming (typology)	Blocks	X	X			X			X		X	X	X	X		X	X			X	X	X					X
	Lines of code			X				X											X								
	Both				X		X			X					X			X					X	X	X	X	
	Translate blocks to text				X		X			X					X			X					X	X	X	X	
Language	Symbolic		X	X		X			X		X	X	X			X	X			X	X	X					X
	Text	X			X			X		X								X	X					X			
	Both						X								X								X		X	X	
Programming structures	Arithmetic operators		X		X		X	X	X	X				X	X	X	X	X	X	X			X	X	X	X	X
	Relational and logic operators		X		X	X	X	X	X	X				X	X	X	X	X	X	X			X	X	X	X	X
	Variables and constants		X		X		X	X	X	X				X	X	X	X	X	X	X				X	X		X
	Decisions structures	X	X		X		X	X	X	X	X	X	X	X	X	X	X	X	X	X			X	X	X	X	X
	Repetitions structures (Loops)	X	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X	X
	Functions				X		X	X	X	X				X	X	X	X		X	X				X	X	X	X
Type of processing	Sequencial	X	X	X	X	X	X	X	X	X	X	X	X	X								X			X	X	
	Parallel																			X	X						X
	Both															X	X	X	X					X			
User can evolve in programming skills	Weak	X																					X				
	Reasonable		X	X		X	X				X	X	X	X	X						X					X	X
	Good				X			X	X	X						X	X	X	X	X				X	X		
Requirements																											
Requires continuous internet connection		X	X	X	X	X	X	X	X	X		X	X		X		X		X		X	X	X		X	X	
Requires Login or Classroom Code for younger children	Login (L) Classroom Code (C)							L or C								L	L										

Table A1. *Cont.*

		Requirements																									
Categories		**CodeMove PT**	**Run Marco (All Can Code)**	**Bee-Bot Emulator**	**Blockly Games**	**Cargo-Bot**	**Code for LifeRapid Router**	**Code Monkey**	**Code Studio**	**Coding With Chrome**	**Kodu**	**Lego Bits and Bricks**	**Lightbot**	**app Inventor**	**micro:bit**	**Kodular**	**Thunkable**	**mBlock**	**Code Combat**	**Scratch**	**ScratchJr**	**The FooS**	**TynkerHour of Code**	**Alice**	**Tynker IDE**	**Pencil Code**	**Snap!**
Operating system	Browser	X	X	X	X		X	X	X			X		X		X			X	X			X		X	X	X
	Windows									X	X				X			X	X	X		X		X			
	IOS		X			X				X		X	X		X		X	X			X	X					X
	Android		X									X	X	X	X	X	X	X			X	X					X
	MAC																		X	X				X			
Equipment's	Computers	X	X	X	X		X	X	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X
	Tablets	X	X	X	X	X	X	X	X			X	X	X	X	X	X	X		X	X	X			X	X	X
	Smartphones	X	X		X			X	X				X	X	X	X	X	X		X	X	X			X		X
Available in Portuguese (PT, BR)		X	X	X	X				X		X			X		X		X	X	X	X	X		X			X
Available in English			X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Price	Free	X	X	X	X	X	X		X	X	X	X		X	X	X		X	X	X	X			X	X	X	X
	Partially free							X					X				X						X				
	Paid																					X					
	Minimum amount at publication date in $USD Total (T) or per Month (M)							4 (M)					2.69 (T)		15 only board		20 (M)					7.99(M)	10 (M)				
		User Interaction																									
Environment stimulating for children	Weak															X											
	Reasonable	X	X	X	X	X	X	X			X	X	X										X			X	X
	Good								X	X				X	X		X	X	X	X	X			X	X		
Activities (typology)	Pre-defined	X	X	X	X	X		X	X		X	X	X			X						X	X				
	Open													X										X		X	
	Both						X			X					X		X	X	X	X	X				X		X
Scenarios	Do not permit	X												X												X	
	Only one					X		X			X		X														
	Multiple		X	X	X				X		X	X					X			X	X	X	X	X	X		X
	Creation of a new ones						X			X						X	X	X	X	X	X				X		X

Table A1. *Cont.*

User Interaction																											
Multimedia	Audio									X				X		X	X		X	X	X			X			X
	Image									X				X		X	X		X	X	X			X	X		X
	Video													X		X	X		X	X				X			X
	Pre-defined by the software		X	X	X	X	X	X	X		X	X	X					X		X		X					X
Actors/Sprites	Only one	X		X		X						X	X										X			X	
	Multiples		X		X		X	X	X		X						X			X	X	X		X			X
	Creation of a new sprites									X							X	X	X	X	X						X
	Actors pre-defined by the application																					X			X		
Programming Actors/Sprites	One		X	X	X	X			X			X	X									X			X	X	
	Two or more	X						X		X	X						X	X	X	X	X	X		X			X
	Not applicable													X													
Create drawings					X				X	X	X				X	X	X	X	X	X				X	X	X	X
User can enter data							X	X	X	X				X	X	X	X	X	X	X				X	X		X
Programming Tangible Objects																											
Programming robots				X						X					X	X		X							X		
Programming Drones																		X							X		
Programming for Mobile Devices														X		X	X										
Pedagogical Adequacy																											
Age/ school grade suitability	Preschool			X					X			X	X								X		X				
	K1- K4	X	X	X			X		X		X	X	X	X					X	X	X	X	X		X		X
	K5 - K6	X	X		X		X	X	X		X			X	X		X		X	X			X	X	X	X	X
	K7- K9	X	X		X	X	X	X	X	X	X			X	X	X	X	X	X	X				X	X	X	X
	K10 - K12					X		X	X					X	X	X	X		X	X				X			X

Table A1. *Cont.*

	Preschool																										
Adequate to articulate with other curricular domains	Area of Society and Citizenship			X					X											X	X	X					X
	Area of Expression and Communication								X											X	X						X
	- Motor abilities											X								X	X						X
	- Artistic abilities								X											X	X	X				X	X
	- Oral and Written abilities			X					X											X		X					
	- Mathematical abilities			X					X			X								X		X				X	
	Areas of Knowledge			X					X											X	X						X
	K1–k12																										
	Mathematics	X	X	X	X	X	X	X	X	X		X	X	X	X	X	X	X	X	X				X	X	X	X
	Language			X					X					X		X	X		X	X	X			X			X
	English	X	X	X			X	X		X		X	X	X	X	X	X		X	X			X	X	X		X
	Sciences, History, Geography, etc.			X	X				X					X	X	X	X			X	X			X	X		X
	Arts and Physical Education	X						X	X	X	X	X			X	X	X			X	X	X		X	X		X
	Society and Citizenship		X	X												X	X			X	X			X			X
Tools for the teacher	Create groups						X	X			X								X	X			X				
	Monitoring activities						X	X								X			X	X			X		X		
	Share with others							X			X						X		X	X				X	X	X	
	Tutorials, Lesson Plans					X	X	X	X			X			X	X	X	X	X	X	X		X	X	X	X	X

References

1. Wing, J.M. Computational thinking. *Commun. ACM* **2006**, *49*, 33–35. [CrossRef]
2. Papert, S. *Mindstorms: Children, Computers, and Powerful Ideas*; Basic Books: New York, NY, USA, 1980.
3. Papert, S.; Harel, I. *Situating Constructionism*; MIT Press: Cambridge, MA, USA, 1991.
4. Bell, T.; Tymann, P.; Yehudai, A. The Big Ideas in Computer Science for K–12 Curricula. *Eur. Assoc. Theor. Comput. Sci.* **2018**, *124*, 2–12.
5. Heintz, F.; Mannila, L.; Färnqvist, T. A review of models for introducing computational thinking, computer science and computing in K-12 education. In Proceedings of the Frontiers in Education Conference (FIE), Erie, PA, USA, 12–15 October 2016; pp. 1–9. [CrossRef]
6. Hubwieser, P. Computer Science Education in Secondary Schools—The Introduction of a New Compulsory Subject. *Trans. Comput. Educ.* **2012**, *12*, 16:1–16:41. [CrossRef]
7. Hubwieser, P.; Armoni, M.; Giannakos, M.N. How to Implement Rigorous Computer Science Education in K-12 Schools? Some Answers and Many Questions. *Trans. Comput. Educ.* **2015**, *15*, 5:1–5:12. [CrossRef]
8. Sáez-López, J.M.; Román-González, M.; Vázquez-Cano, E. Visual programming languages integrated across the curriculum in elementary school: A two year study using "Scratch" in five schools. *Comput. Educ.* **2016**, *97*, 129–141. [CrossRef]
9. DGE. *Aprendizagens Essenciais para da Disciplina de TIC*; Direção-geral da Educação, Ministério da Educação de Portugal: Lisboa, Portugal, 2017.
10. Gomes, A.; Mendes, A.J. Learning to program—Difficulties and solutions. In Proceedings of the International Conference on Engineering Education—ICEE, Coimbra, Portugal, 3–7 September 2007.
11. Martins, S.W.; Mendes, A.J.; Figueiredo, A.D. Diversifying Activities to Improve Student Performance in Programming Courses. *Commun. Cogn.* **2013**, *46*, 39–58.
12. Jenkins, T. On the difficulty of learning to program. In Proceedings of the 3rd Annual Conference of LTSN-ICS, Loughborough, UK, 23 August 2002.
13. Katai, Z.; Toth, L. Technologically and artistically enhanced multi-sensory computer-programming education. *Teach. Teach. Educ.* **2010**, *26*, 244–251. [CrossRef]
14. Wang, Y.; Li, H.; Feng, Y.; Jiang, Y.; Liu, Y. Assessment of programming language learning based on peer code review model: Implementation and experience report. *Comput. Educ.* **2012**, *59*, 412–422. [CrossRef]
15. Garneli, V.; Giannakos, M.N.; Chorianopoulos, K. Computing education in K–12 schools: A review of the literature. In Proceedings of the 2015 IEEE Global Engineering Education Conference (EDUCON), Tallinn, Estonia, 18–20 March 2015; pp. 543–551.
16. Navarrete, C.C. Creative thinking in digital game design and development: A case study. *Comput. Educ.* **2013**, *69*, 320–331. [CrossRef]
17. Webb, H.C.; Rosson, M.B. Exploring careers while learning Alice 3D: A summer camp for middle school girls. In Proceedings of the 42nd ACM Technical Symposium on Computer Science Education, Dallas, TX, USA, 9–12 March 2011; ACM: New York, NY, USA, 2011; pp. 377–382.
18. Maya, I.; Pearson, J.N.; Tapia, T.; Wherfel, Q.M.; Reese, G. Supporting all learners in school-wide computational thinking: A cross-case qualitative analysis. *Comput. Educ.* **2015**, *82*, 263–279.
19. Kelleher, C.; Pausch, R. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Comput. Surv. CSUR* **2005**, *37*, 83–137. [CrossRef]
20. Grover, S.; Pea, R. Computational thinking in K-12: A review of the state of the field. *Educ. Res.* **2013**, *42*, 38–43. [CrossRef]
21. Lewis, C.M. Is pair programming more effective than other forms of collaboration for young students? *Comput. Sci. Educ.* **2011**, *21*, 105–134. [CrossRef]
22. Sengupta, P.; Farris, A.V. Learning kinematics in elementary grades using agent-based computational modeling: A visual programming-based approach. In Proceedings of the 11th International Conference on Interaction Design and Children, Bremen, Germany, 12–15 June 2012; ACM: New York, NY, USA, 2012; pp. 78–87.
23. Sengupta, P.; Kinnebrew, J.S.; Basu, S.; Biswas, G.; Clark, D. Integrating computational thinking with K-12 science education using agent-based computation: A theoretical framework. *Educ. Inf. Technol.* **2013**, *18*, 351–380. [CrossRef]

24. Webb, H.C.; Rosson, M.B. Using scaffolded examples to teach computational thinking concepts. In Proceedings of the 44th ACM Technical Symposium on Computer Science Education, Denver, CO, USA, 6–9 March 2013; ACM: New York, NY, USA, 2013; pp. 95–100.
25. Rahmat, M.; Shahrina, S.; Latih, R.; Yatim, N.F.M.; Zainel, N.F.A.; Rahman, R.A. Major Problems in Basic Programming that Influence Student Performance. *Procedia Soc. Behav. Sci.* **2012**, *59*, 287–296. [CrossRef]
26. Kordaki, M. A drawing and multi-representational computer environment for beginner learning of programming using C: Design and pilot formative evaluation. *Comput. Educ.* **2010**, *54*, 69–87. [CrossRef]
27. Wilson, A.; Moffat, D.C. *Evaluating Scratch to Introduce Younger Schoolchildren to Programming*; PPIG, 2010; pp. 1–12. Available online: http://scratched.media.mit.edu/sites/default/files/wilson-moffatppig2010-final.pdf (accessed on 6 May 2019).
28. Robins, A.; Rountree, J.; Rountree, N. Learning and teaching programming: A review and discussion. *Comput. Sci. Educ.* **2003**, *13*, 137–172. [CrossRef]
29. Cheng, G. Exploring factors influencing the acceptance of visual programming environment among boys and girls in primary schools. *Comput. Hum. Behav.* **2019**, *92*, 361–372. [CrossRef]
30. Eckerdal, A. Novice Programming Students' Learning of Concepts and Practice. Ph.D. Thesis, Uppsala University, Uppsala, Sweden, 2009. Available online: http://uu.diva-portal.org/smash/get/diva2:173221/FULLTEXT01.pdf (accessed on 11 April 2019).
31. Yang, T.-C.; Hwang, G.-J.; Yang, S.J.H.; Hwang, G.-H. A Two-Tier Test-based Approach to Improving Students' Computer-Programming Skills in a Web-Based Learning Environment. *Educ. Technol. Soc.* **2015**, *18*, 198–210.
32. Baser, M. Attitude, gender and achievement in computer programming. *Middle East J. Sci. Res.* **2013**, *14*, 248–255.
33. Carter, J.; Jenkins, T. Gender and programming: What's going on? *ACM SIGCSE Bull.* **1999**, *31*, 1–4. [CrossRef]
34. Korkmaz, Ö.; Altun, H. Engineering and CEIT student's attitude towards learning computer programming. *Int. J. Soc. Sci.* **2013**, *6*, 1169–1185. [CrossRef]
35. Sullivan, A.; Bers, M.U. Girls, boys, and bots: Gender differences in young children's performance on robotics and programming tasks. *J. Inf. Technol. Educ. Innov. Pract.* **2016**, *15*, 145–165. [CrossRef]
36. Franklin, D.; Skifstad, G.; Rolock, R.; Mehrotra, I.; Ding, V.; Hansen, A.; Weintrop, D.; Harlow, D. Using upper-elementary student performance to understand conceptual sequencing in a blocks-based curriculum. In Proceedings of the 2017 ACM SIGCSE Technical Symposium Computer Science Education, Seattle, WA, USA, 8–11 March 2017; ACM: New York, NY, USA, 2017; pp. 231–236.
37. Myers, B. Taxonomies of visual programming and program visualization. *J. Vis. Lang. Comput.* **1990**, *1*, 97–123. [CrossRef]
38. Lye, S.; Koh, J. Review on teaching and learning of computational thinking through programming: What is next for K-12? *Comput. Hum. Behav.* **2014**, *41*, 51–61. [CrossRef]
39. Weintrop, D.; Wlensky, U. How block-based, texto-based and hybrid block/text modalities shape novice programming practices. *Int. J. Child Comput. Interact.* **2018**, *17*, 83–92. [CrossRef]
40. Bau, D.; Gray, J.; Kelleher, C.; Sheldon, J.; Turbak, F. Learnable Programming: Blocks and Beyond. *Commun. ACM* **2017**, *60*, 72–80. [CrossRef]
41. Kolling, M.; Brown, N.; Altadmri, A. Frame-Based Editing. *J. Vis. Lang. Sentient Syst.* **2017**, *3*, 40–67. [CrossRef]
42. Weintrop, D.; Holbert, N. From blocks to text and back: Programming patterns in a dual-modality environment. In Proceedings of the 48th ACM Technical Symposium on Computer Science Education, Seattle, WA, USA, 8–11 March 2017; ACM: New York, NY, USA, 2017.
43. Matsuzawa, Y.; Ohata, T.; Sugiura, M.; Sakai, S. Language migration in non-cs Introductory programming through mutual language translation environment. In Proceedings of the 46th ACM Technical Symposium on Computer Science Education, Kansas City, MI, USA, 4–7 March 2015; ACM Press: New York, NY, USA, 2015; pp. 185–190.
44. Grover, S.; Pea, R.; Cooper, S. Designing for deeper learning in a blended computer science course for middle school students. *Comput. Sci. Educ.* **2015**, *25*, 199–237. [CrossRef]
45. Salant-Meerbaum, O.; Armoni, M.; Ben-Ari, M. Learning computer science concepts with scratch. *Comput. Sci. Educ.* **2013**, *23*, 239–264. [CrossRef]

46. Johngard, K.; McDonald, J. Using Alice in overview courses to improve success rates in programming I. In Proceedings of the 2008 21st Conference on Software Engineering Education and Training, Charleston, SC, USA, 14–17 April 2008; pp. 75–79.
47. Mullins, P.; Whitfield, D.; Conlon, M. Using Alice 2.0 as a first language. *J. Comput. Sci. Coll.* **2009**, *24*, 136–143.
48. Dann, W.; Cooper, S.; Pausch, R. *Learning to Program with Alice*; Prentice Hall Press: Upper Saddle River, NJ, USA, 2011.
49. Freitas, S.; Liarokapis, F. Serious Games: A New Paradigm for Education? In *Serious Games and Edutainment Applications*; Ma, M., Oikonomou, A., Jain, L., Eds.; Springer: London, UK, 2011.