Article

Prototype of a Recommendation Model with Artificial Intelligence for Computational Thinking Improvement of Secondary Education Students

Raquel Hijón-Neira [1,*], Cornelia Connolly [2], Celeste Pizarro [3] and Diana Pérez-Marín [1]

[1] Computer Science Department, Universidad Rey Juan Carlos, 28032 Madrid, Spain; diana.perez@urjc.es
[2] School of Education, National University of Ireland Galway, H91 TK33 Galway, Ireland; cornelia.connolly@universityofgalway.ie
[3] Applied Mathematics Department, Universidad Rey Juan Carlos, 28032 Madrid, Spain; celeste.pizarro@urjc.es
* Correspondence: raquel.hijon@urjc.es

Abstract: There is a growing interest in finding new ways to address the difficult task of introducing programming to secondary students for the first time to improve students' computational thinking (CT) skills. Therefore, extensive research is required in this field. Worldwide, new ways to address this difficult task have been developed: visual execution environments and approaches by text programming or visual programming are among the most popular. This paper addresses the complex task by using a visual execution environment (VEE) to introduce the first programming concepts that should be covered in any introductory programming course. These concepts include variables, input and output, conditionals, loops, arrays, functions, and files. This study explores two approaches to achieve this goal: visual programming (using Scratch) and text programming (using Java) to improve CT. Additionally, it proposes an AI recommendation model into the VEE to further improve the effectiveness of developing CT among secondary education students. This integrated model combines the capabilities of an AI learning system module and a personalized learning module to better address the task at hand. To pursue this task, an experiment has been carried out among 23 preservice secondary teachers' students in two universities, one in Madrid, Spain, and the other in Galway, Ireland. The overall results showed a significant improvement in the Scratch group. However, when analyzing the results based on specific programming concepts, significance was observed only in the Scratch group, specifically for the Loop concept.

Keywords: computational thinking; programming; secondary education; preservice teachers; recommendation model with AI

check for updates

Citation: Hijón-Neira, R.; Connolly, C.; Pizarro, C.; Pérez-Marín, D. Prototype of a Recommendation Model with Artificial Intelligence for Computational Thinking Improvement of Secondary Education Students. *Computers* **2023**, *12*, 113. https://doi.org/10.3390/computers12060113

Academic Editors: Phivos Mylonas, Katia Lida Kermanidis and Manolis Maragoudakis

Received: 31 March 2023
Revised: 12 May 2023
Accepted: 23 May 2023
Published: 26 May 2023

1. Introduction

Teaching programming to secondary education students is doubtless very important nowadays, not just to learn how to program a computer but to foster many other key abilities related to learning how to code [1]. It has been a matter of study as to how to better introduce it to primary and secondary students [2,3]. When examining the training of teachers to undertake the complex and demanding task of teaching programming to secondary students, it is evident that they require additional computer science education [4]. Research suggests that incorporating more emphasis on computational thinking (CT) during their undergraduate degrees can effectively influence preservice teachers' CT in order to teach it afterward [5].

The number of educators qualified to teach computing is insufficient. In the USA, for instance, there are only about 2300 teachers who teach high school AP Computer Science, whereas there are around 25,000–30,000 high schools [6]. Although the numbers are higher across the globe [7], they are still insufficient. This insufficiency has sparked a worldwide interest in examining the impact of programming on fostering computational thinking

(CT) and improving higher cognitive skills [8]. Additionally, there is a growing focus on understanding human behavior based on computer science concepts [9]. This emphasis on CT stems from its significance as a crucial skill in today's world [10].

Papert said that by learning how to program, a child can acquire a better knowledge of prospective behavior and how things work [11]. Other authors explain that in order to acquire meaningful knowledge, the process has to affect the students [12]. Therefore, certain innovations in the materials and methods by which the subject is taught by their teachers may affect their students' feelings of success and self-confidence. This notion aligns with the principles of constructivism in teaching, as advocated by Piaget and Vygotsky [13–17].

To teach programming for the first time, many approaches and methodologies have been used, such as mobile devices [18] or pair programming [19]. Additionally, many studies deal with the misunderstanding and difficulties of the programming concepts learned [20]. Following the pace of learning programming, CT students' improvement has also been studied by higher education educators and researchers with different interventions [21] or using game approaches [22]. Therefore, it is recommended to continue improving and developing learning environments that foster CT, in preservice and in-service teachers facilitating their use, and in investigations for improving CT [23].

Trying to achieve the complex task of applying the learned programming grammar to produce a programming code that effectively solves a problem is what the authors [24] have proposed with an AI with a deep learning recommendation system. It was found that the university students who used the system for teaching programming and computational thinking performed better regarding learning achievement and computational thinking than those who did not. Thus, there can be an AI model built from the learning experience gathered without the need for external data [24].

Most studies applied AI to the analysis of data results. The authors propose a model for AI recommendations on the learning of programming for CT improvement in secondary education students. The model is based on the data results of the usage of the TPACK PrimaryCode VEE with their preservice teachers in two ways: block programming and text programming. This study was conducted at two universities: one in Ireland and one in Spain. The system already provides instant visual learning feedback to students, and the AI model this paper proposes would help with difficult learning tasks that preservice teachers have encountered.

This research paper presents the following hypothesis (H): it is possible to produce a recommendation model with artificial intelligence that recommends the best pace (order of concepts) and provides aid ad hoc to students when using a VEE with a text programming language or block-based programming. The aim is to introduce programming concepts to secondary education students and improve their CT skills. The hypothesis is based on their future teachers' performance using the VEE. For this study, we asked 23 preservice Computer Science teachers to follow a two-hour session on the TPACK PrimaryCode Visual Execution Environment for learning programming concepts at a university in Ireland with TPACK PrimaryCode VEE and block programming (Scratch) and in Spain with TPACK VEE PrimaryCode and text language (Java). Based on this hypothesis, there were four research questions:

RQ1: Did preservice teacher students significantly improve their understanding of programming concepts to foster their CT when using VEE PrimaryCode with a text programming language (Java)?

RQ2: Did preservice teacher students significantly improve their understanding of programming concepts to improve their CT when using the VEE PrimaryCode with block programming (Scratch)?

RQ3: If RQ1 is affirmative, were some concepts easier or better understood than others, and therefore should an AI recommendation model suggest the order and provide aids ad hoc to introduce them when using a text language to perform it, or maybe extra help is needed?

RQ4: If RQ2 is affirmative and some concepts are easier or better understood than others, should an AI recommendation model suggest the order and provide aids ad hoc to introduce them when using a block-based language to perform it, or maybe extra help is needed?

This paper is organized as follows: Section 2 reviews the theoretical framework for improving secondary students' and preservice secondary teachers' CT, visual execution environments, and AI recommendation models; Section 3 outlines the proposed AI recommendation model; Section 4 explains the research method followed, the pedagogical approach, participants, and measurement tool; Section 5 presents the results of the experiment, both overall and by programming concepts; Section 6 includes the discussion and limitations of the study; and Section 7 summarizes the conclusions and suggests lines of future work.

2. Theoretical Framework

2.1. Improving Secondary Students' CT

Children can learn skills such as abstract thought, problem-solving, pattern identification, and logical reasoning through the practice of algorithms and programming, which is the focus of computational thinking. Modern educational and infrastructure advancements, such as "CS for All" (https://www.csforall.org/ (accessed on 24 May 2023)), ISTE's Standards for Students in Computational Thinking (https://www.iste.org/explore/Solutions/Computational-thinking-for-all?articleid=152 (accessed on 24 May 2023)), and Concepts of Computational Thinking from the Computer Science Teachers Association (http://advocate.csteachers.org/2014/09/15/computational-think) (accessed on 24 May 2023) [25].

Students traditionally find programming challenging and difficult [26]. Even researchers have studied the improvement of mathematical ability, sequential, conditional, and analytical thinking related to the study of programming in high school [27]. Thus, programming allows for mastering multiple abilities.

Recent students have tried to find what programming paradigm would help better improve students' CT, such as block or text programming [28,29], robotics [30], or adaptive scaffolding in middle school students [31]. In K–12 students, educational robots (ER) have been shown to be effective at improving CT, with boys benefiting more than girls [32], and they are being used more frequently as pedagogical tools to encourage students to learn computer programming and CT. Research indicates that the approach of combining CT with ER has the potential to develop students' CT and programming skills [33].

2.2. Improving Preservice Secondary Teachers' CT

Furthermore, looking at the preservice teachers' training on CT, block programming for mobile devices with AppInventor has been reported satisfactory [34], as have Scratch [35] or unplugged approaches with storytelling such as [36]. Additionally, approaches using robotics allow the development of preservice teachers CT [37]. However, even though much has been improved in how to learn and teach CT, the best way to achieve it is still unclear, even though most of the studies suggest training in programming skills [38].

Due to the high dropout and failure rates among university students, a web application based on the Problem Analysis and Algorithmic Model (PAAM) was suggested. Results indicate that male students valued the web application in the programming (1) course more than female students did [39].

The level of computational thinking and digital competence among Spanish undergraduate students is compared in [40], and it is shown that there is a correlation between the two. However, women outperform men in terms of computational thinking. Additionally, computational thinking was incorporated into the design and application of the programming subject [41] with a programming course in undergraduate degrees. Results showed good reliability for Scratch and Tableau, and the use of visual and interactive programming pedagogies increased the interest of novice and non-technical students in learning technology [41].

2.3. Visual Execution Environment

The PrimaryCode (https://sites.google.com/view/primarycode-v3-english/inicio?pli=1 (accessed on 24 May 2023)) visual execution environment (VEE) proposed in this paper utilizes pre-established Java or Scratch programs for each of the proposed lessons. It is a Java-based application that can be installed on any PC (independently of its operating system). Using pre-established coding environments, such as this one, prevents syntax errors, and there is a guide to cover programming concepts incrementally (from easy to difficult). Such environments make novice programmers feel confident while learning.

According to the Fogg model [42], the three elements designed to change human behavior are motivation (towards acting either for rewards, recognition, fear, pleasure, etc.), skill (the difficulty level experienced in carrying out the deed), and the trigger (the agent that causes the action). This VEE provides for a dynamic where these three elements converge at the same time, a successful teaching–learning strategy for new concepts, which in this instance includes those related to an introduction to a programming course.

The Mishra and Koehler TPACK model [43] serves as a foundation for the VEE's integration of required knowledge and the creation of a practical tool for teaching programming concepts [44]. TPACK identifies the areas in which technology is consistently incorporated into instruction and the pupil's learning experience is improved. Three fields of knowledge converge in this area: content knowledge (programming ideas), pedagogical knowledge (visualization of the script execution and on the other parts of the PC, such as the screen, the memory, the files, etc.), and technological knowledge (executing scripts with Java and Scratch). TPACK is located at the confluence of the three areas, as shown in Figure 1.

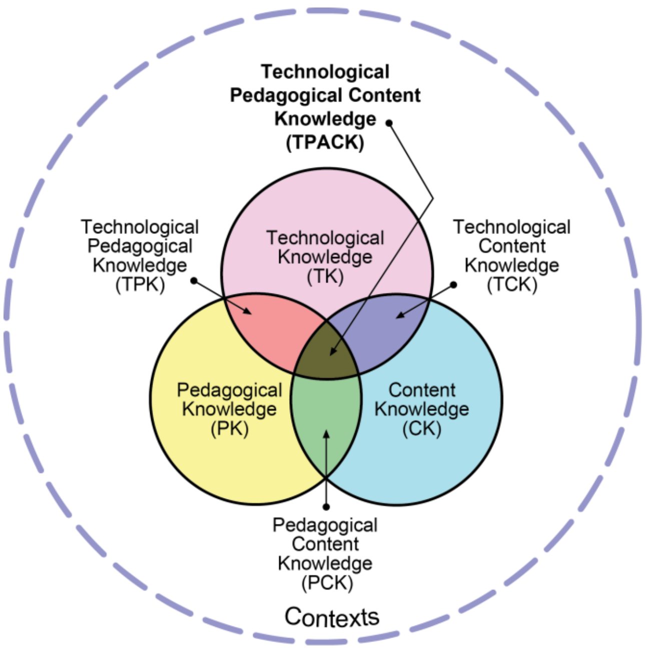

Technological Pedagogical Content Knowledge (TPACK)
Technological Pedagogical Knowledge (TPK)
Technological Knowledge (TK)
Technological Content Knowledge (TCK)
Pedagogical Knowledge (PK)
Content Knowledge (CK)
Pedagogical Content Knowledge (PCK)
Contexts

Figure 1. TPACK model [45].

2.4. AI Recommendation Models

Nowadays, artificial intelligence (AI) is a current trend in many different fields since it allows decision-making, prediction, and problem-solving [46,47].

For instance, it has been used in physics [48] and in data processing and speech and image recognition [49], also in medicine to prevent eye diseases [50], smart cities [51], policy-making [52], healthcare [53], agriculture [54], to learn from the emotional analysis of user reviews on gadgets bought [55], in computational chemistry [56], image recognition [57], language translation [58], in log analysis of student interactions to predict answers to difficult questions or grade of correctness [46], or to prevent student dropouts [59].

Artificial intelligence (AI) is a breakthrough technology, and in [60], it is suggested to use program coding scaffoldings to introduce high school students to the foundations and operation of two of the most well-known AI algorithms. It offers an alternative perspective on AI by having students work on partial Scratch code while learning about the computational reasoning that underlies AI processes.

The authors [24] have suggested using an AI with a deep learning recommendation system to attempt the challenging task of using the learned programming grammar to write programming code that successfully solves a problem. It was discovered that university students who participated in the system for teaching programming and computational thinking outperformed those who did not in terms of learning achievement and computational thinking. As a result, an AI model can be created using the learning experience amassed without the requirement for outside data [24]. The authors [24] have recommended using AI with deep learning to attempt the challenging task of using the learned programming language to develop programming code that successfully solves a problem.

3. Prototype of an AI Recommendation Model

In this study, a Java platform is used for developing the AI recommendation system. To enable Java scripts to acquire data from TPACK PrimaryCode VEE scripts written in Java, it is built on Java Learning Agents based on the TensorFlow Java framework, which can run on any JVM for building, training, and deploying machine learning models [61]. Recurrent neural networks (RNNs) will be used to model dynamic and sequence data because they can more precisely learn the characteristics of students and items to produce suggestions for supervised learning path recommendations (see Figure 2). To improve students' logic and application of related programming languages (text with Java or blocks with Scratch), the TPACK PrimaryCode VEE will combine AI technology to assess its relevance through the operation process of students and recommend alternative learning tasks for students who have difficulty understanding or find the learning process incomprehensible.

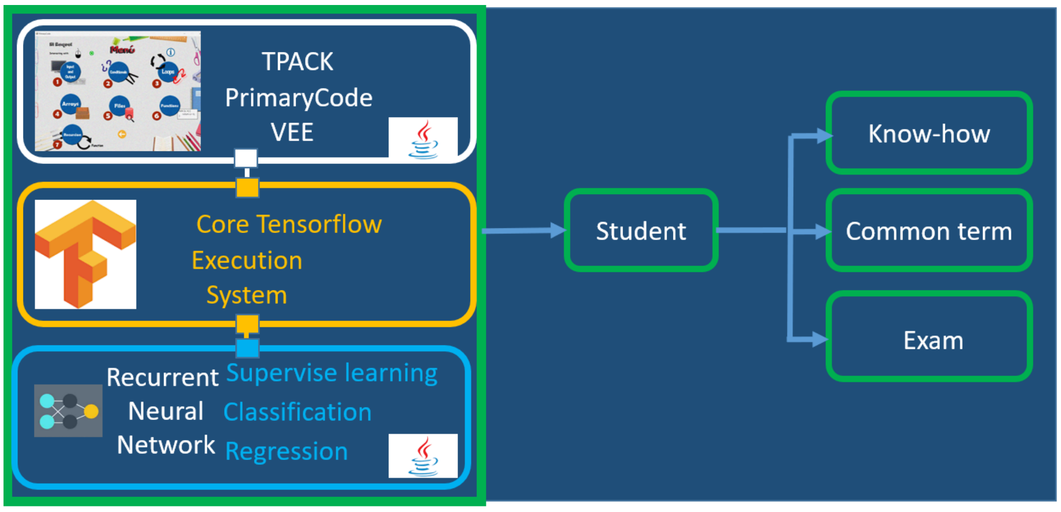


Figure 2. The structure and function of the AI recommendation system. In white the TPACK PrimaryCode VEE, in orange the Tensorflow Execution System, in blue Supervise Learning path recommendations, in green functions to the student.

The system will provide the following functions to the students as depicted in Figure 2 on the right, which illustrates the functions of the AI system [24]. It contains the following: (1) Know-How: According to their own learning path, students can select from a variety of lessons to read; (2) Common term: Provide students with commonly used programming language grammar (text or blocks); (3) Exam: Include questions for each instruction that are pertinent.

The AI system design is depicted in Figure 3, and it is primarily made up of a Java AI learning system module and a personalized learning module [24]. (1) An AI recommendation function that gives students more practice chances based on their learning process is included in the AI learning system module. (2) learning materials and lessons, which offer various AI learning tasks and materials for students to practice screenplay execution in accordance with the studied learning lessons. The personalized learning module consists of two parts: (1) a learning process that keeps track of how each student learns and operates; and (2) code hints, where pertinent prompt information is given to the students in accordance with the assignment.

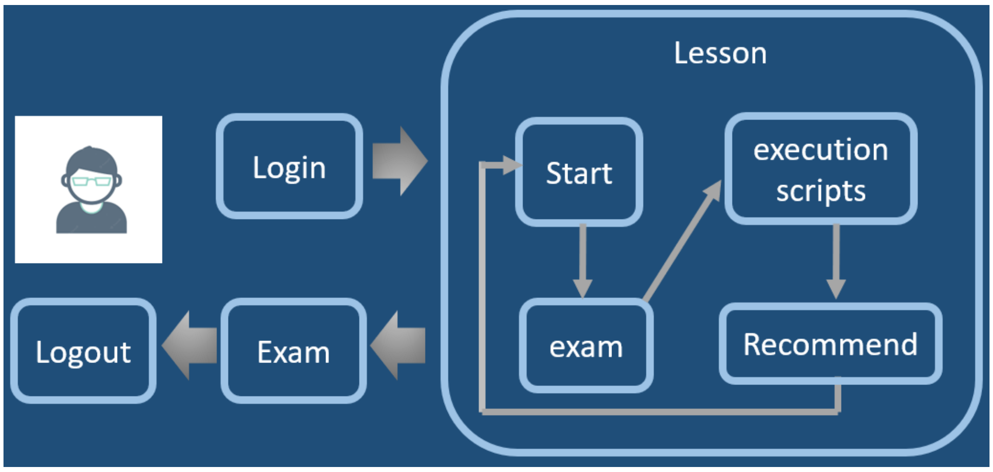


Figure 3. Student-operated learning scenario.

4. Research Method

Based on the principles of research-based learning, the didactic research method for computing education used in this study was created to foster students' foundational programming concepts for the development of CT skills. There was a quasi-experimental pre- and post-testing protocol used. The same evaluation measures were used for both the pre- and post-tests conducted to assess programming concepts in seven introductory programming lessons. However, in one group, the concepts were taught using block programming (Scratch) at the University of Galway, Ireland, while in the other group, text programming (Java) was utilized at the University Rey Juan Carlos of Madrid, Spain. After 10 weeks of learning programming in both block and text programming, in both settings, teachers' students took a pretest on their initial knowledge. Subsequently, they utilized the VEE PrimaryCode tool at their own pace for the duration of the learning period. Following the completion of the programming curriculum, the students took a post-test to evaluate their progress and understanding. Each test was completed individually by each student at their respective location. Figure 4 shows the diagram of the experimental design that followed.

4.1. Pedagogical Approach

The seven lessons corresponding to any introduction to programming course were grouped under a menu option, which allows coherent sequencing without mixing concepts. The TPACK PrimaryCode visual execution environment has pre-established programs, which include prefixed scripts allowing the students to choose from several inputs to practice incrementally on the scripts corresponding to the suggested lessons. This division enables the instructor to, if required, use a different sequencing method than the one suggested. Since some concepts require prior knowledge, it is essential, for example, to comprehend conditions in order to explain how loops work. Table 1 lists the lessons, topics, and number of scripts for each lesson, with an average of 1.5 min per script. The time for reviewing it all in Java or Scratch is also shown.

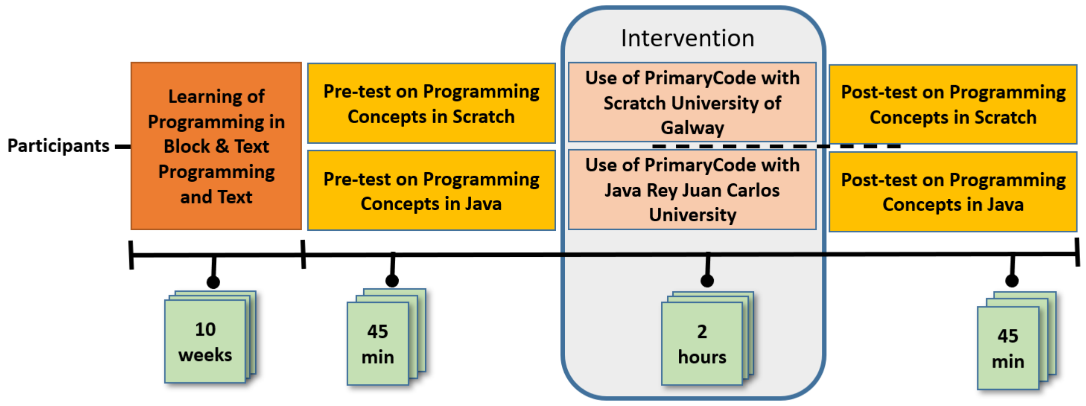

Participants
Learning of Programming in Block & Text Programming and Text
Pre-test on Programming Concepts in Scratch
Pre-test on Programming Concepts in Java
Intervention
Use of PrimaryCode with Scratch University of Galway
Use of PrimaryCode with Java Rey Juan Carlos University
Post-test on Programming Concepts in Scratch
Post-test on Programming Concepts in Java
10 weeks
45 min
2 hours
45 min

Figure 4. Diagram of the experimental Design.

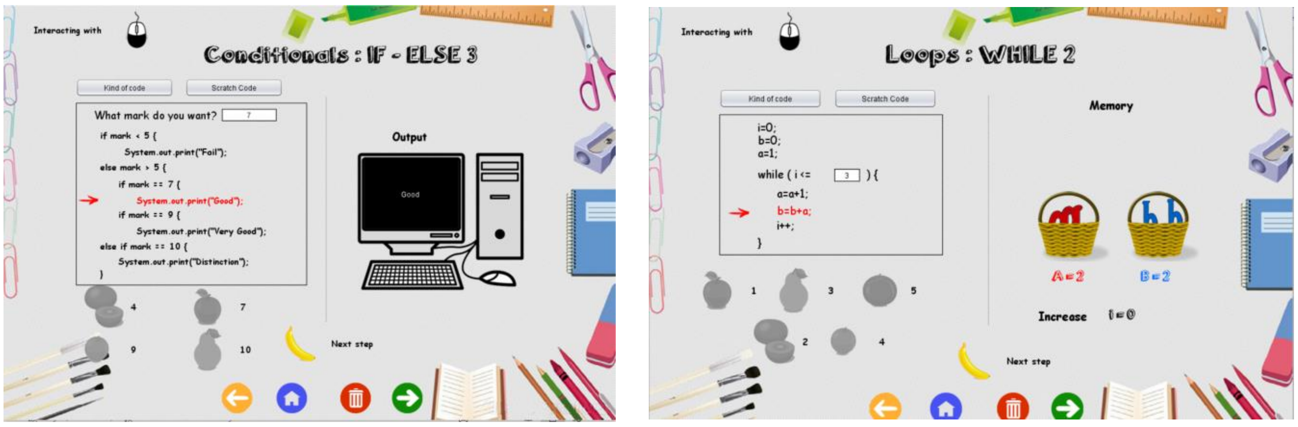
Table 1. Proposal for sequencing topics in the PrimaryCode VEE.

Lesson Number	Topic	Number of Scripts	Reviewing Time
Lesson 1.	Input/Output and Variables	5	7.5′
Lesson 2.	Conditionals	8	12′
Lesson 3.	Loops	18	27′
Lesson 4.	Arrays	12	18′
Lesson 5.	Files	11	16.5′
Lesson 6.	Functions	6	9′
Lesson 7.	Recursion	6	9′

Each lesson was meticulously designed with the intention of promoting effective concept assimilation. The instructional approach followed a sequence wherein the execution of the script was first demonstrated on the left side of the screen. This was followed by a visual representation on the right side, showcasing various elements such as memory boxes or trunks, the PC screen, and an array depicted as a fruit box, among others. Furthermore, the lessons incorporated multiple exercises and scripts, providing students with the opportunity to engage with different inputs. These inputs were conveniently accessible in the lower left corner of the screen. Figure 5 shows two examples of script executions in Java, for conditionals and loops, respectively. In both the script and the visual part, through the combined interaction with the different inputs, the scripts offer instant feedback, enabling assimilation and accommodating new ideas in Java (Figure 6) and Scratch (Figure 7).

4.2. Research Participants

The research participants are two cohorts of preservice computer science teachers' students, totaling 23. One cohort consisted of eight teacher students pursuing a Computer Science and Mathematics undergraduate degree at the University of Galway, Ireland. Their primary focus was on teaching Computer Science and Mathematics to secondary students. This cohort comprised individuals with ages ranging from 20 to 22 years old and Irish nationality. The second cohort comprised 15 teacher students enrolled in a Master of Education program at Universidad Rey Juan Carlos in Madrid, Spain. Their specialization was teaching Computer Science to secondary students. The students in this cohort ranged in age from 23 to 25 years old and held Spanish nationality. None of the participants in the experiment had previous experience teaching programming. All were enrolled in a "Didactics of Computer Science and Technology" course during the first semester of the academic year 2022–2023.



```
What mark do you want? 7
 if mark < 5 {
     System.out.print("Fail");
 else mark > 5 {
     if mark == 7 {
         System.out.print("Good");
     if mark == 9 {
         System.out.print("Very Good");
 else if mark == 10 {
     System.out.print("Distinction");
 }
```

```
i=0;
b=0;
a=1;
while ( i <= 3 ){
    a=a+1;
    b=b+a;
    i++;
}
```

Figure 5. The TPACK PrimaryCode VEE with examples of script execution in Java for conditionals (on the **left**) and loops (on the **right**).

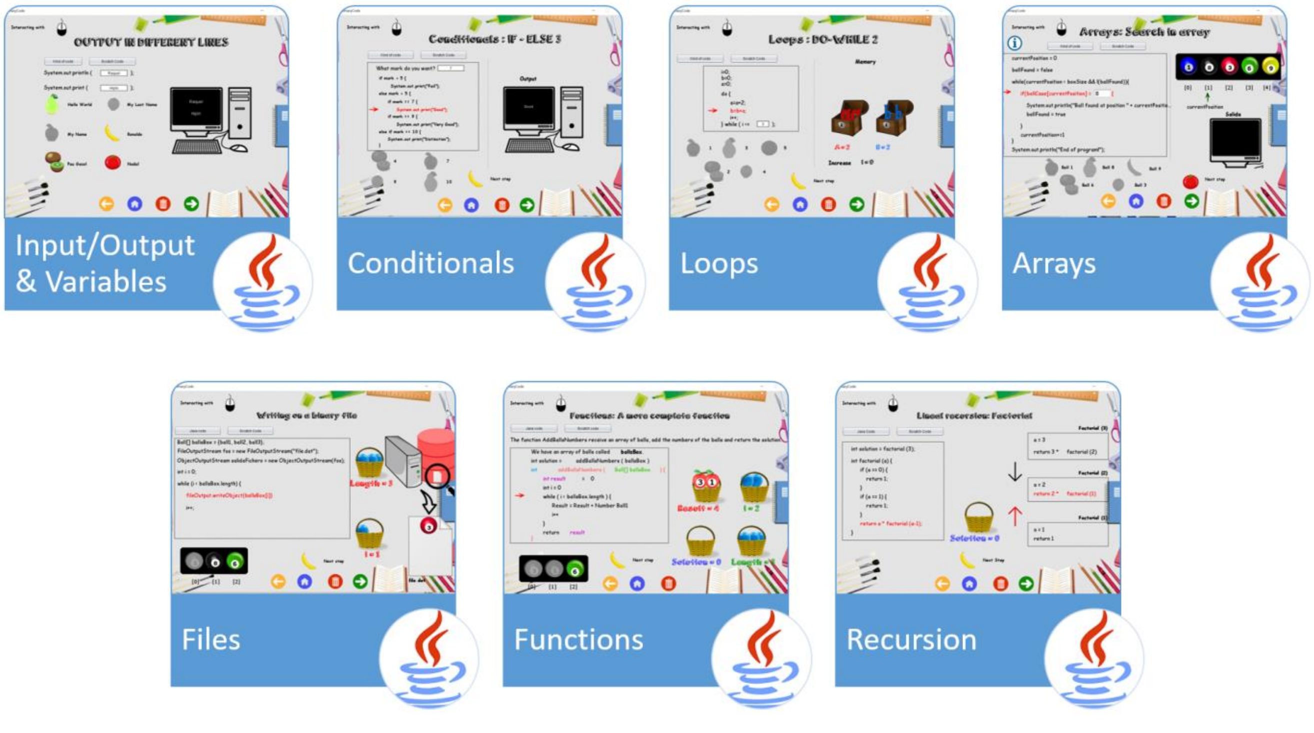

OUTPUT IN DIFFERENT LINES
Input/Output & Variables
Conditionals : IF - ELSE 3
Conditionals
Loops : DO-WHILE 2
Loops
Arrays: Search in array
Arrays
Writting on a binary file
Files
Functions: A more complete function
Functions
Lineal recursion: Factorial
Recursion

Figure 6. The TPACK PrimaryCode VEE with examples of script execution in Java (on the **left**) on each lesson (1–7) and its interactive visualization (on the **right**).

Out of the total 23 university students, 8 (34.7%) belonged to the Computer Science and Mathematics undergraduate degree program at the University of Galway, Ireland, and were part of the Blocks Group in Scratch. Among these 8 students, there were 4 male students (50%) and 4 female students (50%). The remaining 15 students (65.2%) were enrolled in the Master of Education program at Universidad Rey Juan Carlos in Madrid, Spain. They were part of the text group in Java, focusing on teaching Computer Science to secondary students. Out of these 15 students, 11 were male (73.3%) and 4 were female (26.7%). The tutor, who held a Ph.D. in Computer Science and more than 20 years of expertise instructing at the university level, taught the module to the teachers of both groups of teachers' students.

The introduction to teaching with the TPACK PrimaryCode VEE was made in the same manner for the two classes. Both groups of students have previously studied block programming with Scratch and text programming with different languages and backgrounds

(Python, Java, etc.). In addition to the time needed to complete the pre- and post-tests, the research study was completed in a single 2-h period.

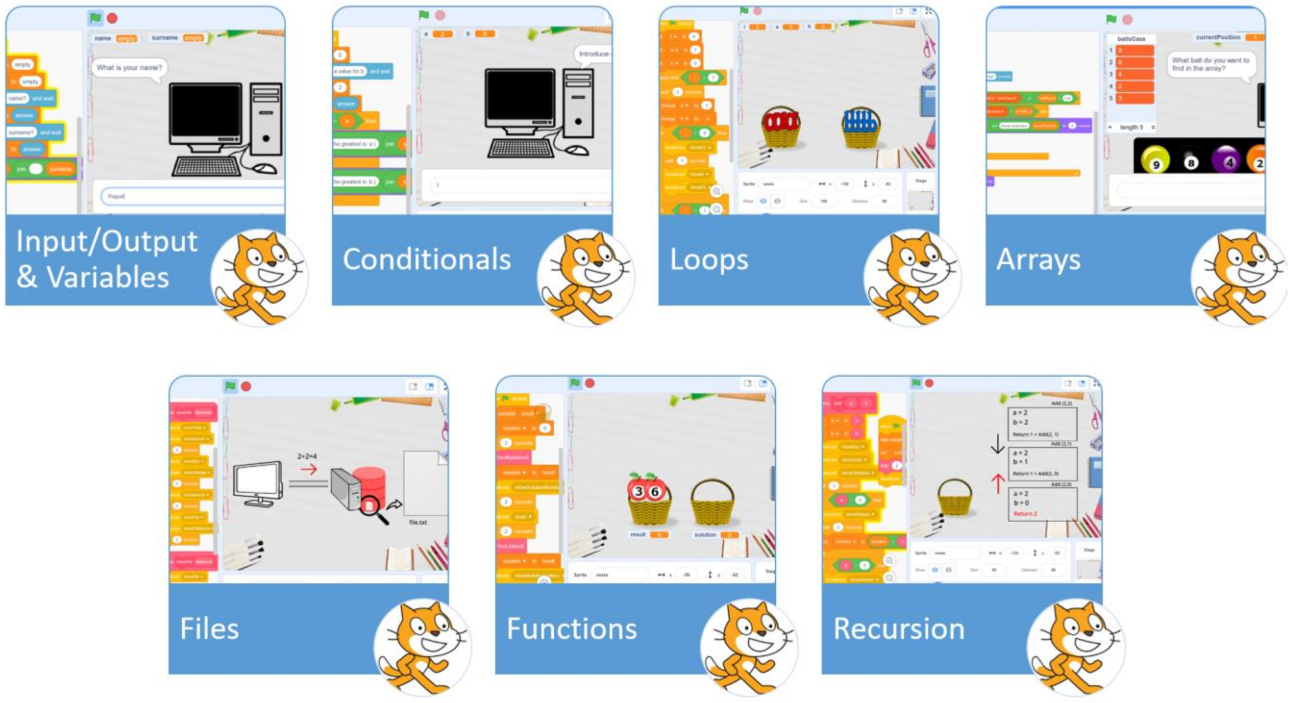


Figure 7. The TPACK PrimaryCode VEE with examples of script execution in Scratch (on the **left**) for each lesson (1–7) and its interactive visualization (on the **right**).

4.3. Measuring Tool

In this study, two similar questionnaires on programming concepts, but different in approach (block or test programming), pre- and post-tests were created to evaluate how learning exercises affected students' learning outcomes and programming skills. The pre- and post-test evaluations followed a consistent methodology for both the block/Scratch group and the text/Java group. Each group was assessed using the same methodology, consisting of 28 multiple-choice queries related to programming concepts. These concepts were specifically addressed in Table 2 within each paradigm (text or block programming). The total score for each test was 10 points. Experts with years of programming expertise created and revised the pre-test and post-test questions. The scoring rubric graded automatically each question that was answered correctly with 1 point and 0 when it was incorrectly answered. Moreover, there are links to the tests in Scratch (https://tinyurl.com/4vny5x7p (accessed on 24 May 2023)) and Java (https://docs.google.com/forms/d/e/1FAIpQLSc_MAvOxvgrz9wTrHbXEdn592t5FVjmhJa3GHl2LwWAYgBHZA/viewform (accessed on 24 May 2023)).

Table 2. Number of questions and concepts addressed on the multiple-choice pre- and post-test.

Lesson Number	Number of Questions	Concept Addressed
Lesson 1. Input/Output and Variables	4	Input, output, input, and output
Lesson 2. Conditionals	2	Conditional and switch
Lesson 3. Loops	9	While, do-while, for
Lesson 4. Arrays	3	Search, read, write
Lesson 5. Files	3	Binary and text files
Lesson 6. Functions	4	Parts, return value, inputs
Lesson 7. Recursion	3	Linear and tale recursion

In order to measure the reliability or internal consistency of the test used, Cronbach's α value is used [62]. The formula for Cronbach's α is calculated as follows:

$$\alpha = \frac{N\,\bar{c}}{\bar{v} + (N-1)\bar{c}}$$

where N is the number of items in the questionnaire, $\bar{c}$ is the average covariance between item pairs, and $\bar{v}$ is the average variance. Table 3 shows the internal consistency of the test is acceptable since all values are greater than 0.7.

Table 3. Values of Cronbach's alpha for Java and Scratch tests.

	Cronbach's Alpha Value	
	Pre-Test	**Post-Test**
Global	0.781	0.784
Scratch	0.818	0.874
Java	0.808	0.800

4.4. Variables

The dependent variables in pre- and post-tests were related to learning results from a general point of view and were named Java and Scratch (see Table 4), scaled from 0 to 10. Furthermore, Table 2 provides detailed information on seven new dependent variables that were introduced to measure different dimensions or lessons. These variables include input/output, loops, conditionals, functions, arrays, recursion, and files. Each of these variables corresponds to specific aspects of programming education. Additionally, the factor "group" is included as a binary variable, representing the utilization of either Scratch or Java programming languages in the study.

Table 4. Summary of variables, type (DV-dependent variables, IV-independent variable), name, and description.

Aspect	**Type**	**Variable**	**Name**
Learning programming	DV	Scratch	Scratch
	DV	Java	Java
Learning programming concepts	DV	Input/Output	I/O
	DV	Loops	Loops
	DV	Conditionals	Condit
	DV	Functions	Function
	DV	Array	Array
	DV	Recursion	Recursion
	DV	Files	File
Methodological factor	IV	Use of Scratch/Java	Group

5. Results

First, the study presents general results about possible differences in the scores using Java and Scratch. For that, scores in the pre-test and post-test are evaluated.

Second, the possible improvement is focused on the different programming concepts learned.

5.1. Overall Results

To measure the possible improvement, the differences between the pre-test and post-test for Java and Scratch are quantified. As a first approximation, Table 5 shows the main descriptive measures, mean and median, as centralization statistics, and standard deviation (SD), as scatter statistics, for each of them. As can be seen, both the mean and the median lower their values from pre-test to post-test in the case of Java (from 6.75 to 6.29 for the mean and from 7.24 to 6.55 for the median). The dispersion of the data in the case of Java remains similar, being, in both cases, very high, so it can be noted that it was a very heterogeneous group in terms of knowledge in the pre-test. In addition, the intervention with Java has not reduced this dispersion (there are still subjects with very different levels of learning). However, in the case of Scratch, the pre-test values are lower than those of Java (mean 5.56 and median 5.34) but increase in the post-test, becoming 7.49 for the mean and 7.07 for the median. In addition, an accomplishment observed in the case of Scratch is that despite the initial wide dispersion of scores in the pre-test (stemming from a heterogeneous group), the utilization of Scratch for learning purposes has significantly reduced this dispersion. Although the value in the table shows a slight increase from 1.86 to 1.97, it is important to note that this is influenced by an outlier in the post-test. Due to the small sample size, the outlier has not been eliminated from the analysis. Thus, it can be said that a more homogeneous level has been achieved throughout the class.

Table 5. Descriptive analysis in the pre- and post-tests for Java and Scratch.

	Pre-Test		**Post-Test**	
	Java	**Scratch**	**Java**	**Scratch**
Mean	6.75	5.56	6.29	7.49
Median	7.24	5.34	6.55	7.07
SD	1.98	1.86	1.96	1.97

Figure 8 shows box plots for the Java and Scratch groups, visually confirming what was said above.

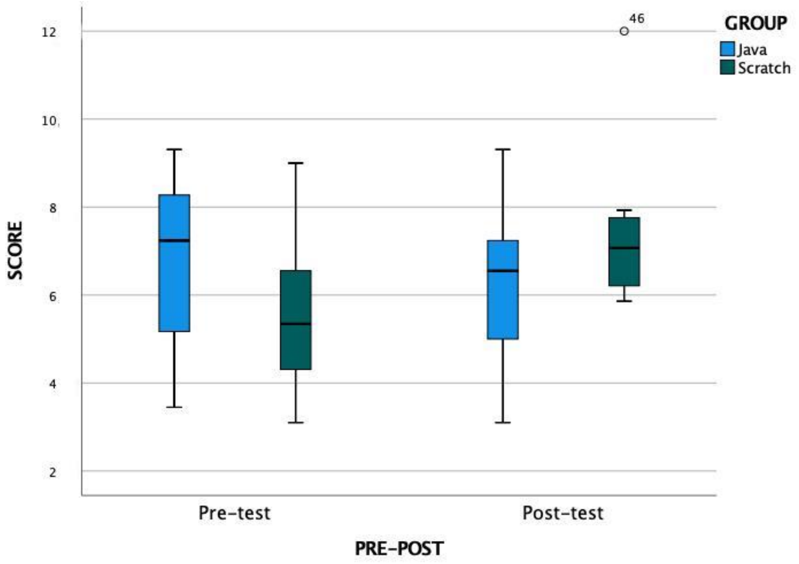


Figure 8. Box-plot for Java and Scratch groups in pre- and post-tests.

Now, let us see if these differences between the post-test and pre-test (negative in the case of Java and positive in the case of Scratch) are statistically significant. For this, and since the sample is small and lacks normality ($p < 0.01$ in the Shapiro–Wilk test for both samples), we will use non-parametric tests.

The Wilcoxon signed-rank test for paired samples is used, as it is the most powerful for this type of data. Z is the test statistic.

As can be seen in Table 6, in the case of Java, the *p*-value is non-significant (*p*-value > 0.05), so it can be concluded that there is no significant difference. In the case of Scratch, this *p*-value is significant (*p*-value = 0.012), so it can be concluded that, in the case of Scratch, there is a significant improvement.

Table 6. A non-parametric test (Wilcoxon test) for pre- and post-tests in the Java and Scratch groups.

	Pre-Post Java	**Pre-Post Scratch**
Z	−0.882	−2.524
p-value	0.372	0.012

5.2. Results by Programming Concepts

The study now focuses on studying the possible gains in scores by concepts (Table 6). The descriptive analysis presented in Table 7 reveals some noteworthy findings. Specifically, in the Java group, the centralization statistics such as mean and median exhibit a decrease in their value across various dimensions, except for I/O, where the values remain consistent. Additionally, there is a slight increase in the file dimension. The dispersion, on the other hand, does not have a fixed pattern, increasing or decreasing depending on the concept.

Table 7. Descriptive analysis of the sample by concepts.

		Pre-Test		**Post-Test**	
		Java	**Scratch**	**Java**	**Scratch**
I/O	Mean	9.16	5.93	9.16	7.81
	Median	10	7.50	10	7.50
	SD	1.81	2.29	1.21	1.6
Loop	Mean	6.44	4.16	5.70	6.25
	Median	7.78	3.88	5.56	6.11
	SD	3.31	1.76	2.94	2.14
Condition	Mean	9.33	9.37	10	10
	Median	10	10	10	10
	SD	2.58	1.77	0	0
Function	Mean	6.50	6.87	6	8.75
	Median	7.50	7.50	7.50	8.75
	SD	2.46	3.20	2.46	1.33
Array	Mean	7.77	5	6.22	5.41
	Median	10	5	6.67	5
	SD	3.01	3.08	3.31	3.53
Recursion	Mean	4.89	3.33	4	4.58
	Median	6.67	3.33	3.33	3.30
	SD	3.53	3.08	3.14	3.05
File	Mean	6.22	4.58	6.66	5.41
	Median	6.67	5	6.67	6.67
	SD	3.39	4.34	3.56	2.48

In the case of the Scratch group, the mean and median increase in all cases, especially in files, loops, and functions. The standard deviation has similar behavior in the Scratch group. Figure 9 visually represents these results, highlighting, in this case, the median, the measure of centralization chosen due to the asymmetry of the data.

Table 8 shows which of these changes between the pre-test and post-test of the Java and Scratch groups are statistically significant. Thus, using the non-parametric test based on a Wilcoxon test, it can be verified that only in the Scratch group, for the Loop concept, the improvement is statistically significant.

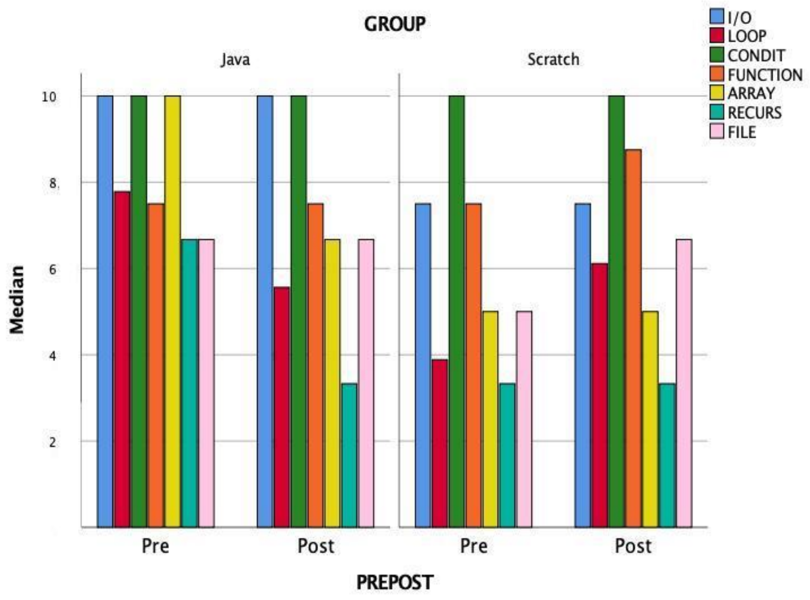


Figure 9. Bar chart for programming Java and Scratch groups in pre- and post-tests.

Table 8. Wilcoxon test between pre- and post-tests in the Java and Scratch groups.

	Java		**Scratch**	
	Z	***p*-Value**	**Z**	***p*-Value**
I/O	0.000	1.000	−1.511	0.131
Loop	−0.669	0.504	−2.388	0.017
Condition	−1.000	0.317	−1.000	0.317
Function	−0.542	0.588	−1.667	0.096
Array	−1.196	0.232	−0.322	0.748
Recurs	−1.192	0.233	−0.736	0.461
File	−0.135	0.892	0.461	0.398

6. Discussion

Due to its ability to facilitate decision-making, prediction, and problem-solving, AI is currently a popular trend in a wide range of industries [45,46]. For instance, it has been applied to physics [47], data processing, speech, and image recognition [48], medicine to prevent eye diseases [49], smart cities [50], policy-making, healthcare [52], agriculture [53], learning from the emotional analysis of user reviews on gadgets purchased [54], computational chemistry [55], image recognition [56], language translation [57], and log analysis of student interactions to predict answers to challenging questions [58].

It is advised in [59] to use program coding scaffolding to introduce high school pupils to the principles and workings of two of the most well-known AI algorithms. AI is a ground-breaking technology, and letting students work on incomplete Scratch code while learning about the computational reasoning that forms the basis of AI processes provides a different viewpoint on AI. Other studies [24] have suggested using an AI with a deep learning recommendation system to attempt the challenging task of using the learned programming grammar to write programming code that successfully solves a problem. It was discovered that university students who participated in the system for teaching programming and computational thinking outperformed those who did not in terms of learning achievement and computational thinking.

As stated before, most studies applied AI to the analysis of data results. In this paper, the authors propose a model for AI recommendation to enhance the learning of programming and improve CT skills among secondary education students. The model is based on the data results of the use of the TPACK PrimaryCode VEE with their pre-service teachers. This study investigates two approaches: block programming and text programming, implemented in two universities, one in Ireland and one in Spain. It is acknowledged as a limitation of the study that the number of preservice teachers' students is small, totaling 23 participants. Another limitation is the absence of secondary education students, who are the primary target of the AI recommendation model aimed at enhancing CT. These limitations will be addressed in future research endeavors to ensure a more comprehensive investigation.

7. Conclusions

The findings of this study, carried out with 23 preservice Computer Science teachers, provide positive confirmation of the formulated hypothesis (H): The results demonstrate that it is indeed feasible to develop an artificial intelligence recommendation model that suggests the optimal sequence of programming concepts and offers tailored assistance to students. This model utilizes a VEE with either a text programming language or a block-based programming approach. The ultimate objective is to introduce programming concepts to secondary education students and enhance their CT skills, building upon the performance of their prospective teachers. This paper introduces a proposition for the integration of an AI recommendation model into the VEE to enhance the development of CT among secondary education students. The AI system module offers several functions, including: (1) Know-How: According to their own learning path, students can select from a variety of lessons to read; (2) Common term: Provide students with commonly used programming language gram-mar (text or blocks); (3) Exam: Include questions for each instruction that are pertinent. Additionally, this study proposes a personalized learning module: (1) An AI recommendation function that gives students more practice chances based on their learning process is included in the AI learning system module. (2) learning materials and lessons, which offer various AI learning tasks and materials for students to practice screenplay execution in accordance with the studied learning lessons. The personalized learning module consists of two parts: (1) a learning process that keeps track of how each student learns and operates; and (2) code hints, where pertinent prompt information is given to the students in accordance with the assignment. Based on this hypothesis, there were four research questions:

The first one (RQ1) was: did preservice teacher students significantly improve their understanding of programming concepts to foster their CT when using the VEE PrimaryCode with a text programming language (Java)? The dispersion of the data in the case of Java remains similar, being, in both cases, very high, so it can be noted that it was a very heterogeneous group in terms of knowledge in the pre-test. In addition, the intervention with Java has not reduced this dispersion (there are still subjects with very different levels of learning). Additionally, in the case of Java, the *p*-value is non-significant (*p*-value > 0.05), so it can be concluded that there is no significant difference.

The second one (RQ2) was: did preservice teacher students significantly improve their understanding of programming concepts to improve their CT when using the VEE PrimaryCode with block programming (Scratch)? In the case of Scratch, the pre-test values are lower than those of Java but increase in the post-test for the mean and the median. In addition, an important achievement observed in the case of Scratch is the significant reduction in the dispersion of scores among students. Initially, the pre-test scores exhibited considerable variation due to the heterogeneous nature of the group. However, as the students engaged in learning through the Scratch platform, this dispersion was greatly reduced. Thus, it can be said that a more homogeneous level has been achieved throughout the class. In the case of Scratch, this *p*-value is significant (*p*-value = 0.012), so it can be concluded that, in the case of Scratch, there is a significant improvement.

Thus, the AI recommendation system should include more learning aids (more practice chances, learning materials, and lesson code hints) for students on the text programming side than on the block programming side.

The third one (RQ3) was: If RQ1 is affirmative, were some concepts easier or better understood than others, and therefore should an AI recommendation model suggest the order and provide aids ad hoc to introduce them when using a text language to perform it, or maybe extra help is needed? RQ1 was not affirmative; therefore, in almost all the concepts, the values decrease; only in the case of input/output are the values maintained, and in the case of files, they increase slightly. Therefore, as before, the AI recommendation system should include extensive aid in almost all programming concepts.

The fourth one (RQ4) was: If RQ2 is affirmative, were some concepts easier or better understood than others, and therefore should an AI recommendation model suggest the order and provide aids ad hoc to introduce them when using a block-based language to perform it, or maybe extra help is needed? In the case of the Scratch group, the mean and median increased in all cases, which means they improved in their learning of all programming concepts, but especially files, loops, and functions. Thus, it can be verified that only in the Scratch group, for the Loop concept, the improvement is statistically significant. Therefore, even if there are improvements in all the programming concepts, it is only statistically significant for loops in the Scratch group; thus, the AI recommendation system should include extensive aid in all the concepts, but probably for loops and, more importantly, according to the experimentation results, in recursion and arrays, where the improvement is smaller.

Further work should encompass a broader evaluation of the study's final target, which pertains to secondary students seeking to improve their CT abilities. Additionally, efforts should be made to validate the proposed AI recommendation model using a diverse cohort of secondary students from different countries. This comprehensive analysis would enable the identification and examination of strengths and weaknesses among the students. The AI recommendation model included on the TPACK VEE PrimaryCode will be offered to the community for free to be installed on any JVM, as it is now for this version.

Author Contributions: Conceptualization, R.H.-N.; methodology, R.H.-N., C.C., C.P. and D.P.-M.; software, R.H.-N.; validation, R.H.-N. and C.P.; formal analysis, C.P.; investigation, R.H.-N. and C.P.; resources, R.H.-N., C.C., C.P. and D.P.-M.; data curation, R.H.-N. and C.P.; writing—original draft preparation, R.H.-N., C.C., C.P. and D.P.-M.; writing—review and editing, R.H.-N., C.C., C.P. and D.P.-M.; supervision, R.H.-N.; funding acquisition, R.H.-N., C.P. and D.P.-M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Spanish Ministry of Universities "José Castillejo" Program for Mobility Stays Abroad for Young Doctors, grant number CAS21/00413 to Raquel Hijón-Neira.

Data Availability Statement: The data are not publicly available due to ethical problems.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Lau, W.W.; Yuen, A.H. Modelling programming performance: Beyond the influence of learner characteristics. *Comput. Educ.* **2011**, *57*, 1202–1213. [CrossRef]
2. Jovanov, M.; Stankov, E.; Mihova, M.; Ristov, S.; Gusev, M. Computing as a new compulsory subject in the Macedonian primary schools curriculum. In Proceedings of the 2016 IEEE Global Engineering Education Conference (EDUCON), Abu Dhabi, United Arab Emirates, 10–13 April 2016.
3. Ouahbi, I.; Kaddari, F.; Darhmaoui, H.; Elachqar, A.; Lahmine, S. Learning Basic Programming Concepts by Creating Games with Scratch Programming Environment. *Procedia-Soc. Behav. Sci.* **2015**, *191*, 1479–1482. [CrossRef]
4. Yadav, A.; Gretter, S.; Hambrusch, S.; Sands, P. Expanding computer science education in schools: Understanding teacher experiences and challenges. *Comput. Sci. Educ.* **2016**, *26*, 235–254. [CrossRef]
5. Yadav, A.; Mayfield, C.; Zhou, N.; Hambrusch, S.; Korb, J.T. Computational Thinking in Elementary and Secondary Teacher Education. *ACM Trans. Comput. Educ.* **2014**, *14*, 1–16. [CrossRef]
6. Margulieux, L.E.; Catrambone, R.; Guzdial, M. Employing subgoals in computer programming education. *Comput. Sci. Educ.* **2016**, *26*, 44–67. [CrossRef]
7. Yadav, A.; Connolly, C.; Berges, M.; Chytas, C.; Franklin, C.; Hijón-Neira, R.; Macann, V.; Margulieux, L.; Ottenbreit-Leftwich, A.; Warner, J.R. A Review of International Models of Computer Science Teacher Education. In Proceedings of the 2022 Working

Group Reports on Innovation and Technology in Computer Science Education (ITiCSE-WGR '22), Dublin, Ireland, 8–13 July 2022; Association for Computing Machinery: New York, NY, USA, 2022; pp. 65–93. [CrossRef]
8. Pea, R.D.; Kurland, D. On the cognitive effects of learning computer programming. *New Ideas Psychol.* **1984**, *2*, 137–168. [CrossRef]
9. Wing, J.M. Computational thinking. *Commun. ACM* **2006**, *49*, 33–35. [CrossRef]
10. Wing, J.M. Computational Thinking, 10 Years Later. 2016. Available online: https://www.microsoft.com/en-us/research/blog/computational-thinking-10-years-later/ (accessed on 24 May 2023).
11. Papert, S. *Mindstorms: Children, Computers, and Powerful Ideas*; Basic Books: New York, NY, USA, 1980.
12. Astin, W.A. *College Retention Rates Are Often Misleading*; Chronicle of Higher Education: Washington, DC, USA, 1993.
13. Piaget, J. *The Moral Judgement of the Child*; Penguin Books: New York, NY, USA, 1932.
14. Piaget, J. *Origins of Intelligence in Children*; International Universities Press: New York, NY, USA, 1952.
15. Vygotsky, L.S. *Thought and Language*, 2nd ed.; MIT Press: Cambridge, MA, USA, 1962.
16. Vygotsky, L.S. *Mind in Society: The Development of Higher Psychological Process; Harvard University Press: Cambridge, MA, USA, 1978.*
17. Vygotsky, L.S. The Genesis of Higher Mental Functions. In *Cognitive Development to Adolescence*; Richardson, K., Sheldon, S., Eds.; Erlbaum: Hove, UK, 1988.
18. Maleko, M.; Hamilton, M.; D'Souza, D. Novices' Perceptions and Experiences of a Mobile Social Learning Environment for Learning of Programming. In Proceedings of the 12th International Conference on Innovation and Technology in Computer-Science Education (ITiCSE), Haifa, Israel, 3–5 July 2012.
19. Williams, L.; Wiebe, E.; Yang, K.; Ferzli, M.; Miller, C. In Support of Pair Programming in the Introductory Computer Science Course. *Comput. Sci. Educ.* **2002**, *12*, 197–212. [CrossRef]
20. Renumol, V.; Jayaprakash, S.; Janakiram, D. *Classification of Cognitive Difficulties of Students to Learn Computer Programming*; Indian Institute of Technology: New Delhi, India, 2009; p. 12.
21. De Jong, I.; Jeuring, J. Computational Thinking Interventions in Higher Education. In Proceedings of the 20th Koli Calling International Conference on Computing Education Research, Koli, Finland, 19–22 November 2020.
22. Agbo, F.J.; Oyelere, S.S.; Suhonen, J.; Laine, T.H. Co-design of mini games for learning computational thinking in an online environment. *Educ. Inf. Technol.* **2021**, *26*, 5815–5849. [CrossRef]
23. Lee, I.; Martin, F.; Denner, J.; Coulter, B.; Allan, W.; Erickson, J.; Malyn-Smith, J.; Werner, L. Computational thinking for youth in practice. *ACM Inroads* **2011**, *2*, 32–37. [CrossRef]
24. Lin, P.-H.; Chen, S.-Y. Design and Evaluation of a Deep Learning Recommendation Based Augmented Reality System for Teaching Programming and Computational Thinking. *IEEE Access* **2020**, *8*, 45689–45699. [CrossRef]
25. Angeli, C.; Giannakos, M. Computational thinking education: Issues and challenges. *Comput. Hum. Behav.* **2019**, *105*, 106185. [CrossRef]
26. Jenkins, T. The motivation of students of programming. *ACM SIGCSE Bull.* **2001**, *33*, 53–56. [CrossRef]
27. Kurland, D.M.; Pea, R.D.; Clement, C.; Mawby, R. A Study of the Development of Programming Ability and Thinking Skills in High School Students. *J. Educ. Comput. Res.* **1986**, *2*, 429–458. [CrossRef]
28. Weintrop, W.; Wilensky, U. Comparing Block-Basedand Text-Based Programming in High School Computer Science Class-rooms. *ACM Trans. Comput. Educ.* **2017**, *18*, 1. [CrossRef]
29. Martínez-Valdés, J.A.; Velázquez-Iturbide, J.; Neira, R.H. A (Relatively) Unsatisfactory Experience of Use of Scratch in CS1. In Proceedings of the 5th International Conference on Technological Ecosystems for Enhancing Multiculturality, Cadiz, Spain, 18–20 October 2017.
30. Aristawati, F.A.; Budiyanto, C.; Yuana, R.A. Adopting Educational Robotics to Enhance Undergraduate Students' Self-Efficacy Levels of Computational Thinking. *J. Turk. Sci. Educ.* **2018**, *15*, 42–50.
31. Basu, S.; Biswas, G.; Kinnebrew, J.S. Learner modeling for adaptive scaffolding in a Computational Thinking-based science learning environment. *User Model. User-Adapt. Interact.* **2017**, *27*, 5–53. [CrossRef]
32. Zhang, Y.; Luo, R.; Zhu, Y.; Yin, Y. Educational Robots Improve K-12 Students' Computational Thinking and STEM Attitudes: Systematic Review. *J. Educ. Comput. Res.* **2021**, *59*, 1450–1481. [CrossRef]
33. Jawawi, D.N.A.; Jamal, N.N.; Halim, S.A.; Sa'Adon, N.A.; Mamat, R.; Isa, M.A.; Mohamad, R.; Hamed, H.N.A. Nurturing Secondary School Student Computational Thinking Through Educational Robotics. *Int. J. Emerg. Technol. Learn. (iJET)* **2022**, *17*, 117–128. [CrossRef]
34. Dodero, J.M.; Mota, J.M.; Ruiz-Rube, I. Bringing computational thinking to teachers' training. In Proceedings of the 5th Inter-national Conference on Technological Ecosystems for Enhancing Multiculturality, Cádiz, Spain, 18–20 October 2017.
35. Gabriele, L.; Bertacchini, F.; Tavernise, A.; Vaca-Cárdenas, L.; Pantano, P.; Bilotta, E. Lesson Planning by Computational Thinking Skills in Italian Pre-service Teachers. *Informatics Educ.* **2019**, *18*, 69–104. [CrossRef]
36. Curzon, P.; McOwan, P.W.; Plant, N.; Meagher, L.R. Introducing teachers to computational thinking using unplugged story-telling. In Proceedings of the 9th Workshop in Primary and Secondary Computing Education, Berlin, Germany, 5 November 2014; pp. 89–92.
37. Jaipal-Jamani, K.; Angeli, C. Effect of robotics on elementary preservice teachers' self-efficacy, science learning, and computa-tional thinking. *J. Sci. Educ. Technol.* **2017**, *26*, 175–192. [CrossRef]
38. Hsu, T.-C.; Chang, S.-C.; Hung, Y.-T. How to learn and how to teach computational thinking: Suggestions based on a review of the literature. *Comput. Educ.* **2018**, *126*, 296–310. [CrossRef]

39. Malik, S.I.; Tawafak, R.; Alfarsi, G.; Ashfaque, M.W.; Mathew, R. A Model for Enhancing Algorithmic Thinking in Programming Education using PAAM. *Int. J. Interact. Mob. Technol. (iJIM)* **2021**, *15*, 37–51. [CrossRef]
40. Esteve-Mon, F.M.; Llopis, M.; Adell-Segura, J. Digital Competence and Computational Thinking of Student Teachers. *Int. J. Emerg. Technol. Learn. (iJET)* **2020**, *15*, 29–41. [CrossRef]
41. Hou, H.-Y.; Agrawal, S.; Lee, C.-F. Computational thinking training with technology for non-information undergraduates. *Think. Ski. Creativity* **2020**, *38*, 100720. [CrossRef]
42. Fogg, B.J. A behavior model for persuasive design. In Proceedings of the 4th international Conference on Persuasive Technology, Claremont, CA, USA, 26–29 April 2009; pp. 1–7.
43. Mishra, P.; Koehler, M. Technological Pedagogical Content Knowledge: A Framework for Teacher Knowledge. *Teach. Coll. Rec.* **2006**, *108*, 1017–1054. [CrossRef]
44. Brennan, K.; Resnick, M. *New Frameworks for Studying and Assessing the Development of Computational Thinking*; American Educational Research Association: Vancouver, BC, Canada, 2012.
45. Mishra, P.; Koehler, M.J. *Introducing Technological Pedagogical Content Knowledge*; American Educational Research Association: Vancouver, BC, Canada, 2008; pp. 1–16.
46. Huang, N.-F.; Chen, C.-C.; Tzeng, J.-W.; Fang, T.T.; Lee, C.-A. Concept Assessment System Integrated with a Knowledge Map Using Deep Learning. In Proceedings of the 2018 Learning With MOOCS (LWMOOCS), Madrid, Spain, 26–28 September 2018; pp. 113–116.
47. Russell, S.; Norvig, P. (Eds.) *Artificial Intelligence: A Modern Approach*; Pearson: London, UK, 2010.
48. Baldi, P.; Sadowski, P.; Whiteson, D. Searching for exotic particles in high-energy physics with deep learning. *Nat. Commun.* **2014**, *5*, 4308. [CrossRef]
49. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436. [CrossRef]
50. Gulshan, V.; Peng, L.; Coram, M.; Stumpe, M.C.; Wu, D.; Narayanaswamy, A.; Venugopalan, S.; Widner, K.; Madams, T.; Cuadros, J.; et al. Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs. *JAMA* **2016**, *316*, 2402–2410. [CrossRef]
51. Bokhari, S.A.A.; Myeong, S. Use of Artificial Intelligence in Smart Cities for Smart Decision-Making: A Social Innovation Perspective. *Sustainability* **2022**, *14*, 620. [CrossRef]
52. Manias, G.; Mavrogiorgou, A.; Kiourtis, A.; Kyriazis, D. SemAI: A novel approach for achieving enhanced semantic interoperability in public policies. In Proceedings of the Artificial Intelligence Applications and Innovations: 17th IFIP WG 12.5 International Conference, AIAI 2021, Hersonissos, Crete, Greece, 25–27 June 2021; Proceedings 17. Springer: Cham, Switzerland, 2021.
53. Kleftakis, S.; Mavrogiorgou, A.; Mavrogiorgos, K.; Kiourtis, A. Digital Twin in Healthcare Through the Eyes of the Vitruvian Man. In *Innovation in Medicine and Healthcare: Proceedings of 10th KES-InMed 2022*; Springer: Singapore, 2022; pp. 75–85.
54. Sharma, A.; Georgi, M.; Tregubenko, M.; Tselykh, A.; Tselykh, A. Enabling smart agriculture by implementing artificial intelligence and embedded sensing. *Comput. Ind. Eng.* **2022**, *165*, 107936. [CrossRef]
55. Day, M.-Y.; Lin, Y.-D. Deep Learning for Sentiment Analysis on Google Play Consumer Review. In Proceedings of the 2017 IEEE International Conference on Information Reuse and Integration (IRI), San Diego, CA, USA, 4–6 August 2017; pp. 382–388.
56. Goh, G.B.; Hodas, N.O.; Vishnu, A. Deep learning for computational chemistry. *J. Comput. Chem.* **2017**, *38*, 1291–1307. [CrossRef]
57. Esteva, A.; Kuprel, B.; Novoa, R.A.; Ko, J.; Swetter, S.M.; Blau, H.M.; Thrun, S. Dermatologist-level classi_cation of skin cancer with deep neural networks. *Nature* **2017**, *542*, 115–118. [CrossRef] [PubMed]
58. Wu, Y.; Schuster, M.; Chen, Z.; Le, Q.V.; Norouzi, M.; Macherey, W.; Krikun, M.; Cao, Y.; Gao, Q.; Macherey, K.; et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv* **2016**, arXiv:1609.08144. Available online: http://arxiv.org/abs/1609.08144 (accessed on 24 May 2023).
59. Xing, W.; Du, D. Dropout Prediction in MOOCs: Using Deep Learning for Personalized Intervention. *J. Educ. Comput. Res.* **2018**, *57*, 547–570. [CrossRef]
60. Estevez, J.; Garate, G.; Grana, M. Gentle Introduction to Artificial Intelligence for High-School Students Using Scratch. *IEEE Access* **2019**, *7*, 179027–179036. [CrossRef]
61. Abadi, M.; Isard, M.; Murray, D.G. A computational model for TensorFlow: An introduction. In Proceedings of the 1st ACM SIGPLAN International Workshop on Machine Learning and Programming Languages (MAPL 2017), Barcelona, Spain, 18 June 2017; Association for Computing Machinery: New York, NY, USA, 2017; pp. 1–7. [CrossRef]
62. Cronbach, L.J. Coefficient alpha and the internal structure of tests. *Psychometrika* **1951**, *16*, 297–334. [CrossRef]