Improving CS1 Programming Learning with Visual Execution Environments

Raquel Hijón-Neira *, Celeste Pizarro, John French, Pedro Paredes-Barragán, Michael Duignan

Posted Date: 21 August 2023

doi: 10.20944/preprints202308.1390.v1

Keywords: Programming; Visual Execution Environment; Java; Visualization; Contextualization

Article

Improving CS1 Programming Learning with Visual Execution Environments

Raquel Hijón-Neira [1,*] Celeste Pizarro [2,] John French [3], Pedro Paredes-Barragán [1,*] and Michael Duignan [3]

[1] Computer Science Department, Universidad Rey Juan Carlos, 28032 Madrid, Spain; pedro.paredes@urjc.es
[2] Applied Mathematics Department, Universidad Rey Juan Carlos, 28933 Móstoles, Madrid, Spain; celeste.pizarro@urjc.es
[3] Department of Computer Science & Applied Physics, Atlantic Technological University Galway, Ireland; john.french@atu.ie, michael.duignan@atu.ie
* Correspondence: raquel.hijon@urjc.es

Abstract: Students in their first year of computer science (CS1) at universities typically struggle to grasp programming concepts. This paper covers research with a a Java programming language-guided visual execution environment (VEE) to teach CS1 students about programming concepts. The topics covered include input and output, conditionals, loops, functions, arrays, recursion, and files, all of which are covered in an introductory programming course. The VEE walks beginner programmers through the fundamentals of programming, utilizing visual metaphors to explain and direct interactive Java tasks. This study's goal is to determine whether a group of 105 CS1 students from four different groups who are enrolled in two academic institutions, one in Madrid, Spain, and the other in Galway, Ireland —can advance their programming abilities under the guidance of the VEE. Second, does the improvement vary depending on the programming concept? The findings demonstrate that students' programming knowledge has greatly increased. This improvement is significant pertaining to all coding topics, while it is more pronounced for some topics than others, like operators, conditionals, and loops. Additionally, it exemplifies how students had little prior understanding of files and recursion. The most well-known concept to them was the sequence concept.

Keywords: programming; visual execution environment; Java; visualization; contextualization

1. Introduction

The development of 21st-century competencies including innovation, analytical thinking, problem-solving, teamwork, social-intercultural competence, productivity, leadership, and responsibility is crucial [1]. Games or Scratch have been mentioned in studies from numerous nations [2–4]. Therefore, it is yet unknown how to deliver programming concepts to CS1 students in the most effective order. Teaching fundamental ideas like program design [5], loops [6], control structures, and algorithms [7] is challenging. Teachers require some direction to approach this activity effectively since problems could result from a lack of or even inadequate teaching methodology [8,9].

A youngster who can program a computer, according to Papert, will be capable of develop a practical grasp of probability-based behaviour since they will be exposed to empowering knowledge about how things function through this activity [10]. In his opinion, programming allows one to express ideas precisely and formally and test if they are sound. This, in turn, allows one to link a coder to cognitive research. In a world where computers are abundant, Papert advised people to "look at programming as a source of descriptive devices" [10] and predicted that "computer languages that simultaneously provide a means of control over the computer and offer new and powerful descriptive languages for thinking will... have a particular effect on our language for describing ourselves and our learning" (p. 98). Learning to code involves a variety of factors, including the educational process, instructional materials, technology employed, and metacognitive aspects. Others have underlined that engaging lessons and activities only have value when they have an impact on the pupils [11]. Additionally, it was claimed that using teaching aids and innovative

 doi:10.20944/preprints202308.1390.v1

teaching techniques may increase students' feelings of success [12] and aid in their development of confidence, which is consistent with constructivist teaching methods and the theories of Piaget and Vygotsky [13–17].

Numerous strategies have been used, for instance leveraging Mobile technology [18] or other techniques, for example, pair programming [19], to assist students learning programming for the first time. Traditionally, students have encountered challenges and misperceptions regarding the concepts studied [20].

The contribution of this work is a rigorous investigation of how to teach fundamental programming concepts at the CS1 level, and how this influences the learning progress of the pupils in response to this proposition. In this research, a Java Visual Executing Environment (JVEE) is evaluated as a teaching, learning, and practice tool for computer programming that was created for CS1 students. There are two research questions that aim to address this.

RQ1: Can this cohort of CS1 students benefit from a Java Visual Execution Environment to enhance their understanding of programming concepts?

RQ2. Which programming principles are typically easier to understand, furthermore which are challenging?

The ideas covered here can be found in a typical "Introduction to Programming" course, including input and output structures, conditionals, loops, arrays, functions, recursion, and files.

This paper examines whether a group of 63 CS1 undergraduates from two universities, one in Galway, Ireland, and the other in Madrid, Spain who are registered for an Java-based introductory programming course in their respective universities can develop their programming skills under the guidance of the JVEE. Second, it looks into whether different programming concepts have varying degrees of progress. The same lessons and order of concepts was followed on both university sites, and 4 very experienced and coordinated tutors taught the module to the CS1 students in each university: 2 groups were experimental groups, used the JVEE and the other two were control groups, did not use the JVEE. The process included an introduction to each concept being taught and worked through initially using the Java VEE. The JVEE provided interactive exercises with visuals illustrating the execution of the code in steps, alongside giving context for the concepts using pre-made, on-the-spot exercises based on metaphors for each idea and practice with suggested activities in Java. The system was used for the whole of the 1st semester of 2021-22 in Madrid, from September 13th to December 22nd, and secondly in Galway, Ireland, in the 2nd semester of 2022-23, from January 23rd to May 5th. The improvement in the pupils' learning was evaluated with a test before and after, which had 28 multiple-choice questions that cover the programming fundamentals, was the same regarding the pre- and post-tests. The findings reveal that students' programming knowledge did significantly increase. This development is significant for all aspects of programming except conditionals. The improvement is particularly pronounced for some concepts, such as loops, recursion, files, arrays and functions.

This paper is organised as follows: Section two examines the theoretical framework for enhancing programming learning, complementary methods for teaching programming, and Visual Execution Environments. Section 3 describes the research design, pedagogical strategy, research participants, and instrument for measuring. The experiment's findings are presented in Section 4, both generally and by programming ideas. Section 5 discusses the limits of the study. Section 6 summarizes the results and suggests areas for further research.

2. Theoretical Framework

2.1. Learning How to Programme

Programming is purportedly an extremely demanding topic or ability, thus it is understandable that pupils appreciate it difficult [21]. Some have looked into what aspects of students' mathematical aptitude, processing speed, analogical thinking, conditional reasoning, procedural reasoning along with temporal thinking [22] can demonstrate. Good programmers have therefore been proficient in many different areas. Beginner programmers frequently struggle with basic ideas like variables,

loops, and conditionals. According to research, individuals have trouble grasping the syntax and semantics of programming languages, which can result in logical and syntax problems. [23,24]. When presented with programming obstacles, many students have trouble coming up with efficient problem-solving techniques. They might place more emphasis on syntax than algorithm design, resulting in hard to read and maintain code [25].

According to Brooks [26], "I think the difficult one of developing software is the creation of this intellectual construct, specifically its specification, design, and testing, not the work of representing it and evaluating the fidelity of the representation. We still make syntactic mistakes, for sure, but they pale in comparison to the conceptual flaws found in the majority of systems. If this is the case, developing software is and always will be challenging (p. 182). Many of the difficulties highlighted by Brooks are likely to be faced by initial computer science students (CS1) during their initial term of programming instruction. Novice programmers must learn abstract conceptualizing, the computer language they will be using, and as well as the environment in which they will be working in addition to understanding the algorithm design process and algorithm construction.

Several CS1 students learn about the most basic software applications, such as "hello world," only where the fundamental ideas of the primary function and the system output are shown. Over the course of the term, more elements are added, and by the end, pupils have progressively been exposed to the key constructs of the language. In order to develop their ability to write programs to specific requirements, students are expected to practice using programming ideas through a variety of tasks [27]. It is crucial to keep students motivated and interested during the whole CS1 course. Demotivation can cause a decline in persistence and interest in programming learning [28].

The five components are specification, algorithm, design, code, and test which make up the sequential method of turning converting system design requirements into usable programming code. The specification, which is frequently rewritten in a detailed and close-to-implementation manner, helps the students grasp the issue domain and develop an acceptable method. It is typically stated in simple language. The algorithm transforms in programming ideas through design, followed by actual code, drawing significantly on abstraction. This step shouldn't be difficult with a good design, and it depends on the programming language. Testing comes before the program is put into action, which is the last stage. This order is ideal for creating an effective computer program, although students frequently skip the specification and design phases in favor of the last "code" component. Since such methods are frequently strengthened in the way the subject is offered through books and talks, numerous inexperienced programmers have a tendency to focus on syntax [29].

Conceptualization, problem-solving, mathematical logical thinking, procedure, evaluation, bug-fixing, and career advancement are only a few of the abilities that are used in programming [22], and they cannot be used alone. It might be difficult for students to generalize and abstract programming principles. Their capacity to apply newly acquired knowledge to novel issues is hampered by this [30]. They are used in the context of a specific issue or problem region. The undergraduate degree programme A pupil is pursuing will frequently dictate the quantity of programming is done, the language used, and the educational setting.

2.2. Complementary Methods for Teaching Programming

Many different academic backgrounds are represented in CS1 classes, and some students might not have had any programming experience before. It can be difficult for instructors to cater to this wide range of knowledge levels [31]. The instructional methods and programming languages used have a big impact on how well pupils learn. It is crucial to strike the proper balance between theoretical ideas, practical programming, and real-world applications [32]. There are a variety of methods that have been found to be effective in assisting pupils who are taking classes on computer initial experiences learning to program, including making use of an adaptive virtual reality platform [33], problem-solving in artificial intelligence [34], simulation games [35], serious games [36,37], using robots [38], and comparisons between block and text programming [39,40]. Also, programmers must have strong debugging abilities, however, beginners may find it difficult to locate and successfully correct flaws in their code. They might not have a methodical debugging strategy [41]. For pupils

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 August 2023 doi:10.20944/preprints202308.1390.v1

who are having trouble, proper help and resources are essential. Learning results for struggling pupils can be enhanced by early detection and additional support [42].

2.3. The Visual Execution Environment

The PrimaryCode (https://tinyurl.com/2s334mfe (accessed on 30 July 2023)) visual execution environment (VEE) suggested in this article makes use of pre-existing programmes in Java for every single of the suggested lessons. Every computer (regardless the operating system) can have the Java-based program loaded on it. There is a guide to teach programming ideas gradually (from simple to complex), and utilising predefined programming environments, this instance, avoids using syntax problems. Such settings give new programmers a sense of security while they learn.

The Fogg model [43] states that the three factors intended to alter human behaviour are skill (the degree of difficulty faced when performing the deed), trigger (the agent that initiates the action), and desire (to behave out of incentive, awe, or fright, pleasure, etc.). An effective learning and instruction technique for fresh ideas that in this case include the ones connected with a specific introductory coding course, is made possible by this VEE, which creates a dynamic where these three parts all come together simultaneously.

The Mishra and Koehler TPACK model [44] is the basis for the integration of necessary information into the VEE and the development of a useful tool for instructing programming ideas. Through the continual incorporation of technology into instruction, TPACK pinpoints the areas where students' learning experiences are enhanced. In this area, knowledge from three different fields intersects: understanding of the subject (programming ideas), pedagogical (displaying execution of scripts along with other PC activity, the display, RAM, data, etc.), & technical expertise (using J2EE and Scratch to run scripts). As depicted in Figure 1, TPACK is situated where the three regions converge.

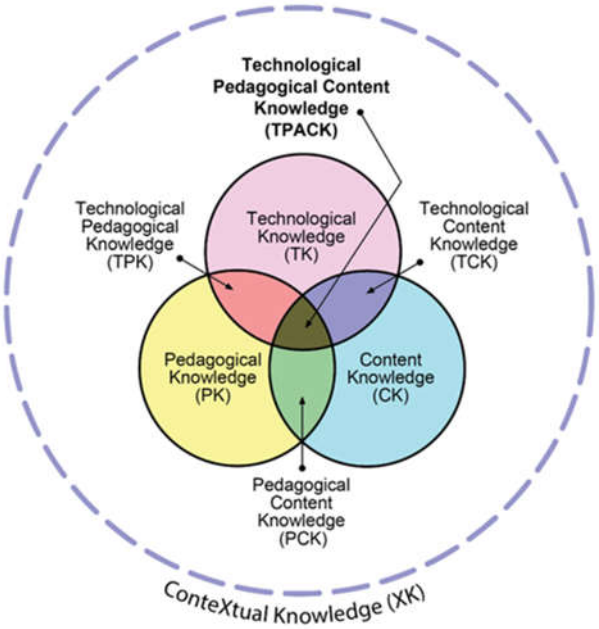

Figure 1. TPACK model [45].

3. Research Design

The method of didactic research for computer instruction employed is based on the concepts of research-based learning, this study was created to support fundamental coding ideas to pupils in the development of programming knowledge. Pre- and post-testing were conducted according to an experimental technique. It included tests before and after the procedure utilized to assess the seven introductory programming lessons' coverage of programming principles using the same evaluation

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 August 2023 doi:10.20944/preprints202308.1390.v1

metrics. In both universities: Atlantic Technological University Galway in Ireland and Spain's Universidad Rey Juan Carlos, their CS1 participants completed a starting literacy pretest, then used VEE for Java to introduce each concept being taught to teach programming to their test groups, and did not use it on their control groups. Following fifteen weeks of instruction in all the programming concepts a CS1 introduction to programming course should include, the students completed a post-test to gauge their understanding and progress after completing the programming curriculum. Each student took the test on their own at their assigned place. The schematic illustrating the subsequent experimental strategy is shown in Figure 2.

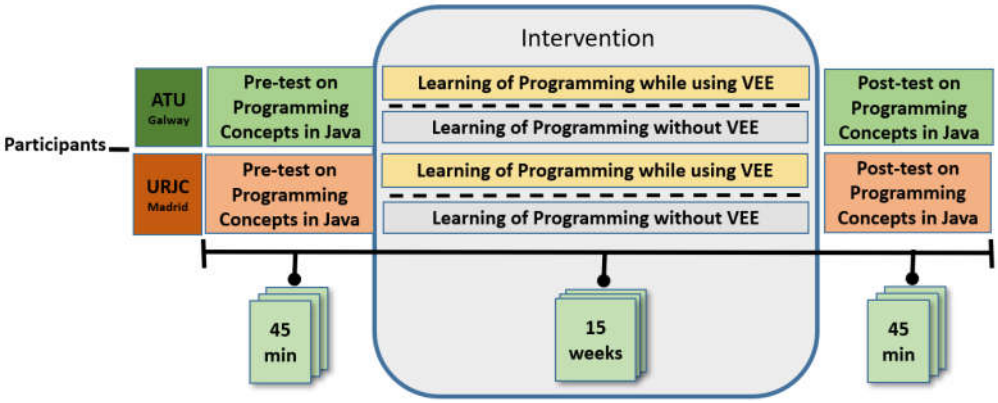

Figure 2. Schematic illustrating the experimental strategy.

3.1. Pedagogical Strategy

A menu item was created that contained all seven lessons necessary for any introductory programming course. This arrangement allows for logical ordering with no confusing topics. The Java TPACK PrimaryCode VEE offers built-in applications using preloaded scripting that let learners select about a variety of suggestions for the scripts' practise sessions gradually as they follow the suggested classes. The instructor can utilize a different sequencing strategy than the one indicated thanks to this division, if necessary. Because certain ideas need earlier understanding, this is crucial, as an illustration, being aware of conditions to be able to describe about loops function. With an average screenplay length of 1.5 minutes, Table 1 provides the topics, description, and quantity of codes for each topic.

Table 1. A suggestion for the Guided Scratch VEE's topic order.

Topic Num.	Description	Num. of Codes
Topic 1.	Input/Output and Variables	5
Topic 2.	Conditionals	8
Topic 3.	Loops	18
Topic 4.	Arrays	12
Topic 5.	Files	11
Topic 6.	Functions	6
Topic 7.	Recursion	6

Each lesson was thoughtfully created with the goal of encouraging successful concept assimilation. The chronology of the instructive method involved first showing how to execute the script towards the leftwards of the screen. Then, towards the right, a graphic representation appeared, showing numerous components including memory chests or containers, the computer monitor, and an array that looked like a carton of vegetables, among other things. Additionally, the sessions included a variety of activities and scripts, giving pupils the chance to interact with various data. All of the data included is easily available throughout the screen's lower left corner. Figure 3 displays two instances of Java codes run, one for conditionals and the other for loops. The scripts

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 August 2023 doi:10.20944/preprints202308.1390.v1

provide immediate feedback with the combined interaction with the script and the visual portion, various data, facilitating assimilation, and welcoming fresh concepts in Java (Figure 4).

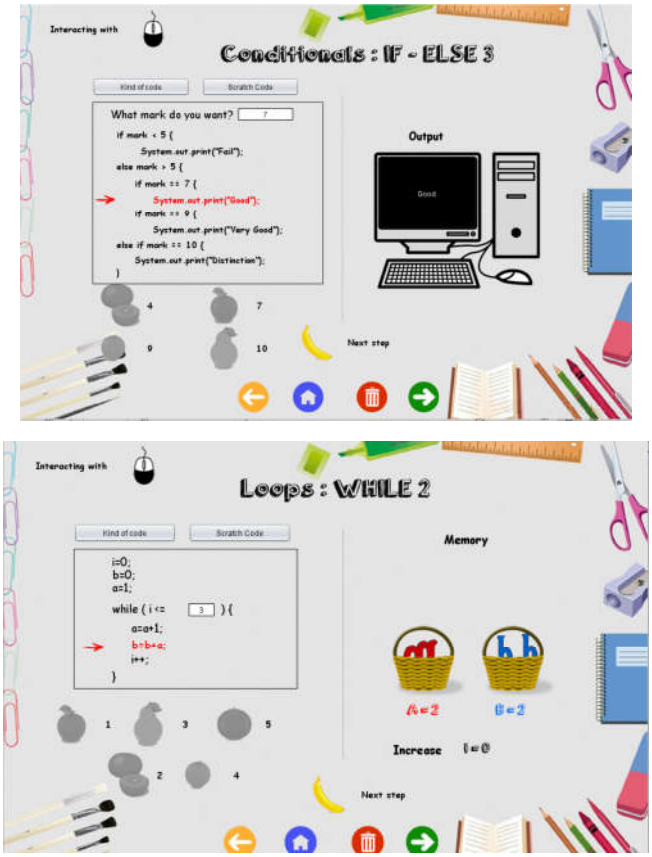

Figure 3. The TPACK PrimaryCode VEE with Java script execution scenarios for loops and conditionals on the left and right, respectively [46].

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 August 2023 doi:10.20944/preprints202308.1390.v1

Figure 4. The TPACK PrimaryCode VEE shows interactive Java script execution samples for every topic (1–7) on the left also its interactivity right-hand side [46].

3.2. Research Participants

Computer science CS1 undergraduates in two separate groups are participating in the research. At the Atlantic Technological University of Galway in Ireland, two cohorts, one for the experimental group with 11 students and another one for the control group, with 10 students. Similar to those in Madrid's Universidad Rey Juan Carlos, Spain, with an arrangement of 22 students within the test group, and 20 among the control group. All 63 students in both universities were registered for an "Introduction to Programming" in Java course, the ones in Spain in the autumn of the academic year 2021-22, and the ones in Ireland during the second semester of the academic year 2022-23. The age range of the students in this cohort was 18 to 21, and they were all citizens of Ireland and Spain, respectively.

3.3. Instrument for Measuring

Pre- and post-tests using the same questionnaire on Java programming topics were utilized in this study to assess the effects of the semester's worth of instruction on students' programming abilities. A standardized methodology was utilized in both the preliminary and final evaluations. The exact same methodology, which included 28 multiple-choice questions about programming topics, was used to evaluate each group. Table 2 deals explicitly with these ideas. Each test had a total score of 10 points. Questions from the initial and subsequent tests included were written and edited by professionals with many years of programming expertise. When a question was answered properly, the scoring rubric automatically awarded 1 point, and when it was answered erroneously 0 points. Additionally, Links exist to the Java tests in English (https://tinyurl.com/primarycodeTest) and Spanish (https://tinyurl.com/t8ecaf6x).

Table 2. The quantity of the preliminary and final multiple-choice questions and concepts covered.

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

3.4. Validity and Reliability

IBM SPSS Statistics Version 28 was used to complete the entire statistical analysis. Cronbach's alpha is 0.819, which is a good value, to gauge the internal consistency of the responses to the preliminary and final questionnaires as well as the questions posed to assess the programming ideas. This value does not rise as items are deleted.

4. Results

This study first focuses on the overall findings, where it is investigated whether the use of the JVEE affects how well students acquire programming skills. Second how specifically was the learning of the different programming concepts namely (input and output, conditionals, loops, arrays, files, functions, and recursion).

4.1. Overall Results

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 August 2023 doi:10.20944/preprints202308.1390.v1

The variations in comparison to the preliminary and final evaluations are investigated in order to generally measure this potential improvement. If there are variations in either, they are measured.

Table 3 displays the average, median, and standard deviation values as well as the primary statistics for each of their centralization, position, and dispersion. These values are shown for both the study group and the unaltered group.

Table 3. Descriptive analysis in Pre and Post-test for Control and Experimental group.

	Pre-test		Post-test	
	Control	**Experimental**	**Control**	**Experimental**
Mean	5.10	4.61	6.91	7.48
Median	5.00	4.64	7.67	7.86
SD	1.29	1.15	1.92	1.50

As stated in Table 3, the mean and median values for both in terms of control and experimental groups are higher in the post-test than what was observed in the pre-test. It is also observed that the increase in the value of the mean and median is greater in the experimental group (from 4.61 to 7.48 for the mean and from 4.64 to 7.86 for the median) than in control group (from 5.10 to 6.91 for the mean and from 5.00 to 7.67 for the median). The standard deviation increases in the post-test for both groups, being much bigger in control group. This table together with its interpretation can be seen complemented with the Figure 5 where the box-plots for both are shown.

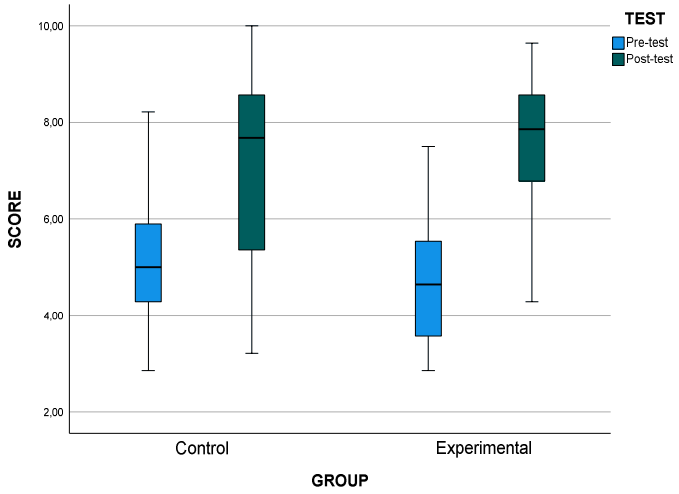

Figure 5. Box-plot for Control and Experimental groups in Pre and Post-test.

Normality of the data allows verifying this improvement with a t test of paired samples. As reflected in the Table 4 , the improvements, both in the control and in the experimental group, are statistically significant. This improvement is accounted for using d-Cohen with a value of 1.851 for the control group and 1.898 for the experimental group, both very large.

Table 4. A paired t-test and effect size for Pre vs Post-test Scores in Control and Experimental group.

	Pre-Post Control	Pre-Post Experimental
Mean	-1.84	-2.85
Deviation	1.857	1.898
df	30	31
t	-5.521	-8.513
p-value	<0.001	<0.001
d Cohen	1.851	1.898

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 August 2023 doi:10.20944/preprints202308.1390.v1

If the interest is in knowing the difference between both methods, that is, directly comparing the control group with the experimental group, the t-student test will be used considering unrelated instances (see Table 5).

Table 5. t- test for independent samples for Control and Experimental group in Pre-test and Post-test Scores.

	t	df	p-value
Pre-test	1.604	61	0.057
Post-test	-1.196	61	0.119

Table 5 shows the difference, first of all, comparing the experimental group pre-test results to those of the control group. As can be seen, both pre-tests are homogeneous (p-value 0.057). For this reason, both groups' follow-up test results can be directly compared, since they start from the same conditions. Again, both post-tests, although they have different mean and median values, this difference is not statistically significant (p-value=0.119) therefore, there is no distinction in learning between the two methods (control and experimental groups).

To take into account all these particularities that have been presented separately, a more advanced mathematical model is presented, ANCOVA. In this model, pre-test scores are included as a covariate. Factors such as place of origin (Spain or Irland), group (control and experimental). To do this, first of all, it has been verified that the conditions to be able to apply this model are verified, namely, normality of the data and homocedasticity are verified.

In Table 6 it can be seen that the pre-test grade does not influence the model, although the influence of the location of the students is statistically significant (in Spain there are better scores, on average, than in Ireland), as well as the group to which they belong (Post-test results show that the experimental group performed better than the control group). The interaction of location and group is not statistically significant. Through the partial values of Eta-Squared we can measure the effect of each significant factor, being 0.266 for the Location, and 0.577 for the Group factor, both corresponding to a large effect, though the value is double for Group (experimental or control).

Table 6. ANCOVA model for Post-test Scores.

	df	Quadratic means	F	p-value	Partial Eta squared
Pre-score	1	3.920	3.051	0.086	0.050
Location	1	94.930	73.892	<0.001	0.260
Group	1	6.236	4.854	0.032	0.577
Location*Group	1	2.634	2.050	0.158	0.034
Error	58	1.285			

4.1. By Means of Programming Topics

Now, the research concentrates on observing what occurs when we take each dimension separately, being: input and output, loops, conditionals, functions, arrays, recursion and files.

The descriptive values for these variables in both the control group and the experimental group, the pre- and post-test, which are depicted in Figure 8, are shown in Table 7. For each dimension, mean, median and standard deviation, in that order, are shown.

Table 7. Topic-specific descriptive analysis of the sample.

	Pre-test		Post-test	
	Control	Experimental	Control	Experimental
Input & Output	8.62	8.04	9.16	9.24
	10	10	10	10
	2.02	2.13	1.21	1.32
	4.76	3.71	6.89	7.50

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 August 2023 doi:10.20944/preprints202308.1390.v1

Loops	4.44	3.33	7.78	7.77
	2.62	2.02	2.68	2.11
Conditionals	6.77	7.96	7.16	8.93
	5.00	10	10	10
	3.54	3.07	3.39	2.07
Functions	5.00	5.15	6.50	7.34
	5.00	5.00	7.50	7.5
	2.23	2.61	1.80	1.86
Arrays	3.97	3.02	6.55	6.86
	3.33	3.33	6.67	6.66
	2.77	2.72	3.21	3.11
Recursion	2.90	2.18	5,11	5.25
	3.33	3.33	3.33	6.66
	2.82	2.17	3.47	3.43
File	3.76	3.75	6.11	7.07
	3.33	3.64	6.67	6.67
	2.23	4.34	3.16	3.43

As can be seen in Table 7, in which the main descriptors are shown, all the dimensions increase, or maintain, their mean value from the pre-test to post-test scores, both in the control group and in the experimental group. The information given in the table is complemented with Figure 6 where the greatest rise in the mean values for the experimental group can be observed.

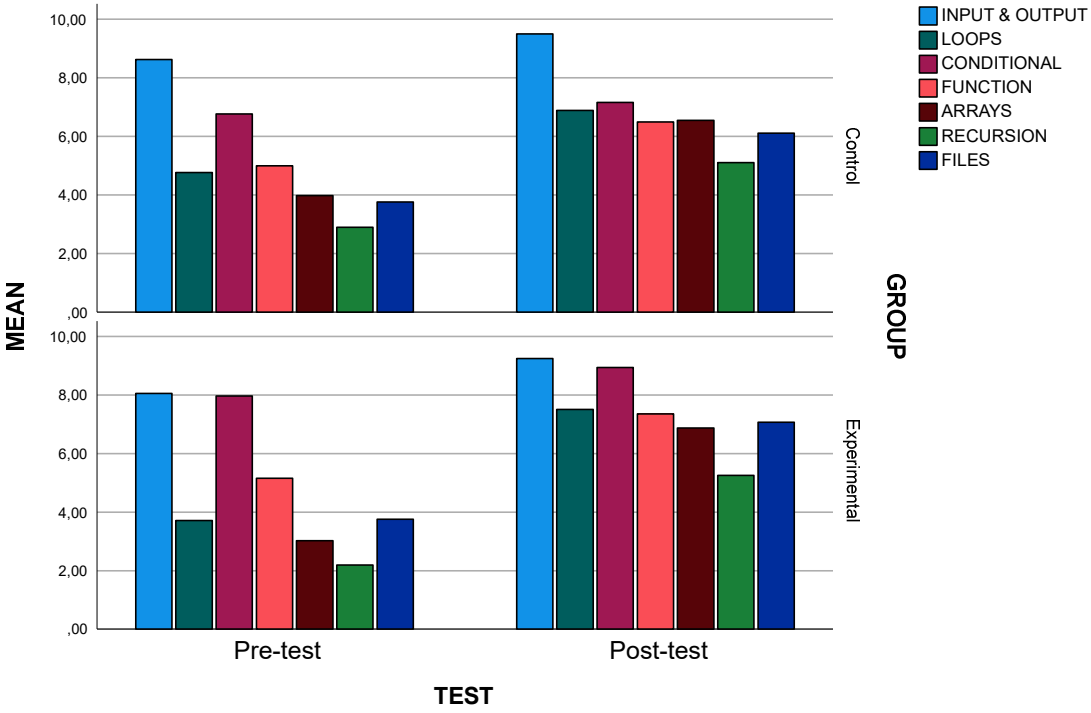

Figure 6. Bar- chat of means for different topics variables in pre-test and post-test.

Table 8 reveals what one of these topic differences between the pre- and post-test are statistically significant. This is tested using an t-Student for paired sample design. For the control group, statistically significant advancement has been made between pre and post-test for all but conditional concepts (p-value>0.05). The same thing occurs in the experimental group. Conditionals is the only concept in which the improvement is not statistically significant.

Table 8. A paired t- test and effect size for Pre vs Post-test Scores by dimensions in the Control and Experimental group.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 August 2023 doi:10.20944/preprints202308.1390.v1

	Group			Experimental		
	t	p-value	d	t	p-value	d
Input/Output	-2.079	0.023	2.37	-2.335	0.013	2.83
Loops	-4.239	<0.001	2.82	-7.408	<0.001	2.88
Conditionals	-1.000	0.163	2.69	-1.438	0.080	3.68
Function	-3.574	<0.001	2.38	-4.625	<0.001	2.67
Arrays	-4.167	<0.001	3.59	-5.947	<0.001	3.56
Recursion	-2.808	0.004	4.26	-3.816	<0.001	4.63
Files	-4.383	<0.001	3.00	-4.209	<0.001	4.48

5. Discussion and Conclusion

The contribution of this work is a rigorous investigation of how to teach fundamental CS1 level coding topics and how this affects students' educational gains in response to such a proposition. In this research, a Java Visual Executing Environment (JVEE) is evaluated as a teaching, learning, and practice tool for computer programming that was created for CS1 students. There are two research questions that aim to address this.

RQ1: Can this cohort of CS1 students benefit from a Java Visual Execution Environ-ment to enhance their understanding of programming concepts?

As reflected in previous section, both improvements, in the control and in the experimental group, are statistically significant for both the test group and the observation group, being both very large.

Since the question is whether use of the JVEE had an impact on the students' learning, it is therefore necessary to directly compare the control group (no use of JVEE) with the experimental group (use of the JVEE). As it was seen, both pre-tests were homogeneous. For this reason, the post-test of both groups could be directly compared, since they started from the same conditions. Both post-tests, although they have different mean and median values, presented this difference as being not statistically significant so it can be said that there is no difference in learning by the two methods (control and experimental groups).

To take into account all these particularities that had been presented separately, a more advanced mathematical model was presented, ANCOVA. In this model, Pre-test results were a covariate that was included. Factors such as place of origin (Spain or Ireland), group (control and experimental) were taken into account. To do this, first of all, it has been verified that the conditions to be able to apply this model were verified, namely, normality of the data and homoscedasticity.

Therefore it has been seen that the pre-test grade does not influence the model, although the influence of the location of the students is statistically significant (in Spain there are better grades, on average, than in Ireland), as well as the group to which they belong to (the experimental group's post-test scores are better compared with those of the control group in all locations). The interaction of location and group is not statistically significant. Through the partial values of Eta-Squared it could be measured the effect of each significant factor, the Location (Spain or Ireland), and the Group factor (experimental or control group), both correspond to a large effect, though the value is double for Group.

RQ2. Which programming principles are typically easier to understand and which are more challenging?

When the study concentrates on observing what occurs when we take each dimension separately, being: input and output, loops, conditionals, functions, arrays, recursion and files. As has been seen in previous section, all the dimensions increase, or maintain, their mean value from the pre- and post-test results, both in the experimental group and the control group. The information given is complemented where the greatest rise in the mean values for the experimental group can be observed.

We examined these concept variations between the pre- and post-test to find out if they were statistically significant. This is tested using a t-Student for paired sample design. For the control

group, there was a statistically significant improvement between pre- and post-test for all the concepts (input and output, loops, functions, arrays, recursion and files) except the conditional concept (p-value>0.05). The same thing occurs in the experimental group, where the mean values increase even more (in this order) for: loops, recursion, files, arrays and functions; again, conditionals is the only concept in which the improvement is not statistically significant.

Author Contributions: For research articles with several authors, a short paragraph specifying their individual contributions must be provided. The following statements should be used "Conceptualization, R.H.N., C.P, J.F., P.P.B, and M.D..; methodology R.H.N., C.P., J.F., P.P.B, and M.D.; software, R.H.N.; validation, R.H.N., C.P, J.F., P.P.B, and M.D.; formal analysis, C.P.; investigation, R.H.N., C.P, J.F., P.P.B, and M.D.; resources, R.H.N., C.P, J.F., P.P.B, and M.D.; data curation, R.H.N.; writing—original draft preparation, R.H.N., C.P, and J.F.; writing—review and editing, R.H.N., C.P, and J.F.; visualization, R.H.N., and C.P.;. All authors have read and agreed to the published version of the manuscript." Please turn to the CRediT taxonomy for the term explanation. Authorship must be limited to those who have contributed substantially to the work reported.

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.

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, https://doi.org/10.1016/j.compedu.2011.01.002.
2. Hijón-Neira, R.; Connolly, C.; Palacios-Alonso, D.; Borrás-Gené, O. A Guided Scratch Visual Execution Environment to Introduce Programming Concepts to CS1 Students. Information 2021, 12, 378. https://doi.org/10.3390/info12090378.
3. 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.
4. 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, https://doi.org/10.1016/j.sbspro.2015.04.224.
5. Lahtinen, E.; Ala-Mutka, K.; Järvinen, H.-M. A study of the difficulties of novice programmers. *ACM SIGCSE Bull.* **2005**, *37*, 14–18, https://doi.org/10.1145/1151954.1067453.
6. Ginat, D. On Novice Loop Boundaries and Range Conceptions. *Comput. Sci. Educ.* **2004**, *14*, 165–181, https://doi.org/10.1080/0899340042000302709.
7. Seppälä, O.; Malmi, L.; Korhonen, A. Observations on student misconceptions—A case study of the Build—Heap Algorithm. *Comput. Sci. Educ.* **2006**, *16*, 241–255, https://doi.org/10.1080/08993400600913523.
8. Barker, L.J.; McDowell, C.; Kalahar, K. Exploring factors that influence computer science introductory course students to persist in the major. *ACM SIGCSE Bull.* **2009**, *41*, 153–157, https://doi.org/10.1145/1539024.1508923.
9. Coull, N.J.; Duncan, I. Emergent Requirements for Supporting Introductory Programming. *Innov. Teach. Learn. Inf. Comput. Sci.* **2011**, *10*, 78–85, https://doi.org/10.11120/ital.2011.10010078.
10. Papert, S. *Mindstorms: Children, Computers, and Powerful Ideas*; Basic Books: New York, NY, USA, 1980.
11. Astin, W.A. *College Retention Rates are Often Misleading*; Chronicle of Higher Education: Washington, DC, USA, 1993.
12. Stuart, V.B. Math Course or Math Anxiety? *Natl. Counc. Teach. Math.* **2000**, *6*, 330.
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, https://doi.org/10.1076/csed.12.3.197.8618.
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. Jenkins, T. The motivation of students of programming. *ACM SIGCSE Bull.* **2001**, *33*, 53–56, https://doi.org/10.1145/377435.377472.
22. Kurland, D.M.; Pea, R.D.; Lement, C.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, https://doi.org/10.2190/bkml-b1qv-kdn4-8ulh.
23. *Alshaigy, Bedoor, and Linda Ott. "Novice programming students: Common difficulties and misconceptions." In Proceedings of the 43rd ACM technical symposium on Computer Science Education, 2012.*
24. *Simon, Beth, et al. "The influence of student aptitude on performance in an introductory computer science course." In Proceedings of the 2013 ITiCSE on Working Group Reports, 2013.*
25. *Robins, Anthony, et al. "Learning and teaching programming: A review and discussion." Computer Science Education, vol. 13, no. 2, 2003, pp. 137-172.*
26. Brooks, F.P. No Silver Bullet: Essence and Accidents of Software Engineering. In Proceedings of the Tenth World Computing Conference, Dublin, Ireland, 1–5 September 1986; pp. 1069–1076.
27. Mishra, D.; Ostrovska, S.; Hacaloglu, T. Exploring and expanding students' success in software testing. *Inf. Technol. People* **2017**, *30*, 927–945, https://doi.org/10.1108/itp-06-2016-0129.
28. *Hanks, Brian, et al. "Keeping students engaged in computer science." In Proceedings of the 45th ACM technical symposium on Computer science education, 2014.*
29. Clancy, M.J.; Linn, M.C. Case studies in the classroom. *ACM SIGCSE Bull.* **1992**, *24*, 220–224.
30. .Vihavainen, Arto, et al. "Multi-faceted support for learning computer programming." In Proceedings of the 42nd ACM technical symposium on Computer Science Education, 2011.
31. R7.Luxton-Reilly, Andrew, et al. "How do students solve intro programming tasks?" In Proceedings of the 13th annual SIGCSE conference on Innovation and technology in computer science education, 2008.
32. R8.Ragonis, Noa, and Uri Leron. "Factors explaining success in an introductory computer science course." ACM Transactions on Computing Education (TOCE), vol. 19, no. 1, 2018, pp. 1-21.
33. Chandramouli, M.; Zahraee, M.; Winer, C. A fun-learning approach to programming: An adaptive Virtual Reality (VR) platform to teach programming to engineering students. In Proceedings of the IEEE International Conference on Electro/Information Technology, Milwaukee, WI, USA, 5–7 July 2014.
34. Silapachote, P.; Srisuphab, A. Teaching and learning computational thinking through solving problems in Artificial Intelligence: On designing introductory engineering and computing courses. In Proceedings of the 2016 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE), Bangkok, Thailand, 7–9 December 2016.
35. Liu, C.-C.; Cheng, Y.-B.; Huang, C.-W. The effect of simulation games on the learning of computational problem solving. *Comput. Educ.* **2011**, *57*, 1907–1918, https://doi.org/10.1016/j.compedu.2011.04.002.
36. Kazimoglu, C.; Kiernan, M.; Bacon, L.; Mackinnon, L. A Serious Game for Developing Computational Thinking and Learning Introductory Computer Programming. *Procedia-Soc. Behav. Sci.* **2012**, *47*, 1991–1999, https://doi.org/10.1016/j.sbspro.2012.06.938.
37. Kazimoglu, C.; Kiernan, M.; Bacon, L.; MacKinnon, L. Learning Programming at the Computational Thinking Level via Digital Game-Play. *Procedia Comput. Sci.* **2012**, *9*, 522–531, https://doi.org/10.1016/j.procs.2012.04.056.
38. Saad, A.; Shuff, T.; Loewen, G.; Burton, K. Supporting undergraduate computer science education using educational robots. In Proceedings of the ACMSE 2018 Conference, Tuscaloosa, AL, USA, 29–31 March 2012.
39. Weintrop, W.; Wilensky, U. Comparing Block-Basedand Text-Based Programming in High School Computer Science Classrooms. *ACM Trans. Comput. Educ.* **2017**, *18*, 1.
40. 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.

41. Murphy, Laurie, et al. "Improving novices' debugging strategies with epistemic scaffolding." In Proceedings of the 2011 annual conference extended abstracts on Human factors in computing systems, 2011.
42. Patitsas, Elizabeth, et al. "Identifying and supporting struggling students in introductory computer science courses." In Proceedings of the 2014 conference on Innovation & technology in computer science education, 2014
43. 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.
44. Mishra, P.; Koehler, M. Technological Pedagogical Content Knowledge: A Framework for Teacher Knowledge. Teach. Coll. Rec. 2006, 108, 1017–1054, https://doi.org/10.1111/j.1467-9620.2006.00684.x.
45. Mishra, P.; Koehler, M.J. Introducing Technological Pedagogical Content Knowledge; American Educational Research Association: Vancouver, BC, Canada, 2008; pp. 1–16.
46. 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