Received: 18 May 2020 | Revised: 25 August 2020 | Accepted: 28 August 2020
DOI: 10.1111/jcal.12495

ARTICLE

Journal of Computer Assisted Learning WILEY

Comparison of object-oriented and robot programming activities: The effects of programming modality on student achievement, abstraction, problem solving, and motivation

Murat Çınar[1] | Hakan Tüzün[2]

[1]Borsa Istanbul Vocational and Technical Anatolian High School, Republic of Turkey Ministry of National Education, Adana, Turkey
[2]Department of Computer Education and Instructional Technology, Faculty of Education, Hacettepe University, Ankara, Turkey

Correspondence
Murat Çınar, Borsa Istanbul Vocational and Technical Anatolian High School, Republic of Turkey Ministry of National Education, Adana, Turkey.
Email: muratcinar@hacettepe.edu.tr, murat_cinar@rocketmail.com

Abstract
This study compares the effects of object-oriented and robot programming activities on programming achievement, abstraction, problem solving, and motivation. In the study, two consecutive experimental cases were conducted to examine the consistency of findings. The research sample comprises 81 tenth-grade students undergoing vocational secondary education. A total of 41 students participated in the first case that spanned 12 weeks, and 40 students participated in the second that spanned 8 weeks. After computational activities, the programming achievement scores significantly increased in all test groups. However, the achievement scores did not differ significantly between the groups. For the comparison groups, there was no statistically significant difference in the pre- and posttest scores of the formal and descriptive abstraction in both cases. However, a statistically significant increase was found in the formal (only Case 2) and descriptive abstraction scores of the students in the experimental groups. The abstraction results revealed a significant difference in the descriptive abstraction scores of Case 2 in favour of the experimental group. No statistically significant difference was found in the problem-solving scores within or between groups. In both cases, the motivation scores of the experimental groups were found to be statistically higher than those of the comparison groups.

KEYWORDS
abstraction, achievement, computational thinking, educational robotics, motivation, problem-solving

1 | INTRODUCTION

Seymour Papert, one of the first researchers who opened the debate on how computers can affect human thinking and learning, underlined the distinction between how computers change information access and human thought. Papert (1980) stated that computers, even without their physical presence can contribute conceptually to our thinking process by influencing how we think. Despite being innovative, the idea of developing students' intellectual thinking skills through computer science concepts did not spread widely owing to the traditional school culture prevalent in those years. However, by the mid-2000s, this idea gained immense popularity under the name of computational thinking (CT) concept through Wing's (2006, 2008) studies.

CT is a problem-solving approach based on computer science concepts, techniques, and approaches. In fact, CT means to think like a programmer and use this thinking skill to solve problems in other areas. Similarly, Bork (1981) stated that computer programming can be used as an educational tool or activity and that students can apply

Required ethical permissions were obtained from the academic ethics commission of Hacettepe University.

the analytical thinking skills developed through programming to a wide range of problems. Algorithms, integral components of CT, are one of the main thinking tools across all disciplines (Doleck, Bazelais, Lemay, Saxena, & Basnet, 2017; Wilkerson-Jerde, 2014). For example, consider the current worldwide situation that involves the novel coronavirus (2019-nCoV) pandemic management which concerns public health. This situation is being closely monitored worldwide. Health officials and researchers are constantly striving to develop algorithms (e.g., contact-tracing, real-time spatial distribution, rapid recognition, patient management, and population-flow-based risk assessment) related to the spread and screening of the 2019-nCoV. This demonstrates their efforts to address complex and multidimensional processes in a scalable manner.

Learning to think computationally has long been considered an important theme in the field of computer science (Pellas & Peroutseas, 2016). Experienced computer scientists can analyse and solve computational problems at a specific level of abstraction, independent of programming language. Another important aspect regarding algorithms is to predict their output for a given input. In other words, programmers need to develop a mental model concerning their program and predict its behaviour (Robins, Rountree, & Rountree, 2003). This is closely related to the abstraction process that decides which details to consider or ignore. While creating representative systems for real-world facts/problems, individuals can choose which aspects of the real-world should be modelled, based on the problem they intend to understand in-depth and solve (Basawapatna, 2016; Irgens et al., 2020).

Abstraction is the cornerstone of CT and the most important hallmark that distinguishes it from other types of thinking. It is essentially a model creation process that enables solving complex problems by separating logical and physical perspectives. Therefore, abstraction is a key to tackle complexity. Moreover, it is actually a skill that is routinely employed in daily life. Furthermore, several fields are based on the concepts attained through various levels of abstraction. The concept of abstraction is used formally or informally in a wide range of disciplines, such as mathematics, cognitive science, artificial intelligence, art, philosophy, complex systems, and computer science. For example, in a geography lesson, concepts such as parallel, meridian, and ecliptic plane use abstractions as their cornerstone. These abstract representations are the basis of navigational calculations (e.g., celestial navigation), especially in the maritime field. Abstraction, which is omnipresent in almost every discipline is defined in a different manner in different disciplines. Therefore, it is difficult to provide a general definition for it. In visual arts, abstraction is defined as reshaping reality through imagination and expressing it with symbols, whereas in mathematics, it is defined to distinguish the properties of objects and/or relationships between the objects and organize them to develop a mathematical structure.

In this study, abstraction is considered as "elimination of redundant details to simplify and acquire generalizations that unveil the core ideas" as suggested by Hill, Houle, Merritt, and Stix (2008). Based on this definition related to computer science, it can be seen that abstraction is a structure comprising two mutually complementary dimensions. Some generic abstract tests are available online. However, most of them are mainly for formal operations that focus on logical, spatial, and inductive reasoning. These tests are commonly suitable for measuring non-verbal skills. Noting that abstraction does not have a single aspect; Hill et al. (2008) suggested that abstraction task types must include not only formal problem-solving operations by noticing the irrelevances in a given problem situation but also descriptive items that are based on the extraction of core meanings from verbal expressions to produce more common generalizations. Although abstraction is highly dependent on developmental stages, it is a skill that can be developed through instructional processes. Computer science has rich references in terms of activities that can be characterized as an abstraction (Colburn & Shute, 2007).

Following a broad consensus on the importance of CT and the necessity of its integration into educational environments, more practical questions regarding how to support and evaluate CT development have recently received attention. Programming is not only a key skill in CT but also a prominent activity that enables the demonstration of CT competences. Programming activities play a key role in the advancement of CT, which requires several different skills. Programming promotes a unique way of thinking that can be considered cardinal to solve problems that entail a synthesis of cognitive efforts and information processing capacity.

Problem solving comprises a series of mental, emotional, and behavioural efforts that an individual exhibits in order to find an effective way to cope with the problematic and stressful situations confronted in professional and everyday life (Heppner & Krauskopf, 1987). The development of learners' problem-solving competences is one of the most desired educational outcomes for life. However, real-life problems are rather dynamic, complex, and versatile compared the ones predefined in structured formal learning contexts (Heppner, Hibel, Neal, Weinstein, & Rabinowitz, 1982). Problem-solving skills require the construction of mental models regarding the problematic situation (Jonassen, 2000). This also provides a basis for the use of computational approaches in problem-solving processes. In present study, programming is considered as an approach—a type of problem-solving form, to reveal the CT process (Lye & Koh, 2014). It is aimed to enable students deal with complexity by using abstractions and develop their problem-solving skills. However, learning to program and transferring programming skills into real-world scenarios require a series of challenging mental efforts. Moreover, the acquisition of programming concepts, even at a basic level of programming, is a challenging task for novices (Durak, 2020). This difficulty mostly stems from the endeavours to concurrently tackle the semantic and syntactic aspects of programming. The representation of ideas in a very structured form becomes especially a chaotic engagement in some instances, where there are not enough orientations and feedbacks to make programming intuitions functional. This situation, which is likely to decrease student motivation, also causes programming to be perceived as a difficult lesson just like mathematics (Alturki, 2016). Motivation is the tendency to maintain one's efforts toward a predetermined purpose. Based on this definition, learning motivation can be considered as a continuity of learning efforts (Ngan & Law, 2015). Students are unlikely to be successful in any learning setting that does not trigger their motivation (Tüzün, Barab, & Thomas, 2019). When students hold inner motivation for tasks that require conceptual understanding, they are more likely to exhibit better learning performance (Carlborg, Tyrén, Heath, & Eriksson, 2019). Therefore, one of the most frequently addressed topics in CT studies is student motivation.

Concerning how best to help students acquire computational concepts, practices, and perspectives, various tools, environments, and pedagogies are being used. In this study, educational robots are proposed as both educational innovation and an alternative to traditional programming education commonly provided through object-oriented languages for computational problem-solving and programming activities. The object-oriented programming is basically conducted in a programming modality in which data and functions are encapsulated within modifiable graphical objects in a visual or textual interface. In contrast, concrete robotic programming is a programming modality in which the function of programming codes created in visual or textual programming environments can be observed on physical objects in real-world settings. The effects of this alternative programming activity on programming achievement, abstraction, problem solving, and motivation were examined.

Despite the increasing research trend in computer science education, especially CT, there is a lack of studies investigating abstraction, which is one of the main components of CT in literature. Moreover, it is seen that most of the studies on CT are based on a single cross-sectional study shaped by relatively short-term instructional activities and narrow-scoped computational concepts. This study is important in terms of contributing to the current understanding of CT and the nature of computational problem-solving processes, especially abstraction. Furthermore, two consecutive case studies offer us the opportunity to examine the consistency of the findings and describe the contextualities from a broad perspective.

1.1 | Research questions

- What is the effect of object-oriented and robot programming education on students' achievement?
    - Does the type of programming activities create a significant impact on programming competencies?
- What is the effect of object-oriented and robot programming education on students' abstraction skills?
    - Does the type of programming activities create any significant differences in abstraction scores?
- What is the effect of object-oriented and robot programming education on students' problem-solving perception?
    - Does the type of programming activities create any significant differences in problem-solving perception?
- Do different programming modalities make a significant difference in students' motivation?

2 | METHOD

2.1 | Research design

True experimental models including the pretest-posttest and posttest control group experimental designs were adopted in this study (Fraenkel & Wallen, 2006). Two consecutive and independent experimental interventions were conducted to examine the consistency of results. Hereinafter, these experimental studies are referred to as Case 1 and Case 2. Case 1 and Case 2 were conducted in 2016 and 2017, respectively, with a time interval of one school year. In each case, students were divided into experimental and comparison groups formed according to their instructional intervention types. The conditions in both cases were designed in a similar way in order to assess to what extent the results obtained from the cases mutually support or exclude each other.

2.2 | Participants

The sample comprises 81 tenth-grade students studying at the information technology department at a vocational high school in Turkey. Only tenth-grade students were included in the study because they had not opted for a computer programming course beforehand in the context of the Information Technology program. In both cases, participants were randomly assigned to the experimental and comparison groups via lottery method. Following the assignments, the participants distributed to experimental and comparison groups in a balanced manner in terms of academic achievement level and gender (Table 1). During the implementation periods, the transition rate of students to 4-year higher education programs was below 1% at the school where the study was conducted. Therefore, the selected participants represent the population with low academic performance to a large extent.

The number of participants in the experimental and comparison groups was 20 and 21 in Case 1, respectively, and 21 and 19 in Case 2, respectively. In both case studies, the number of participants was distributed evenly among the test groups. However, the number of females in both experimental and comparison groups was lower than that of males. The number of female participants in the experimental and comparison groups was 5 (25%) and 4 (19%) in Case 1, respectively, and 4 (19%) and 3 (16%) in Case 2, respectively. This can be regarded as a result of the study involving students undergoing vocational secondary education.

2.3 | Data collection tools

In this study, data were collected through programming achievement and abstraction tests and problem-solving and motivation scales, respectively. Additionally, a personal information form was used to obtain the qualitative data of the participants.

TABLE 1 Distribution of participants' gender and ninth-grade point average (GPA) according to test groups

			Gender		GPA	
Case	Implementation date	Group	Male (f)	Female (f)	Mean	Std. dev.
Case 1	2016-spring	Comparison	17	4	54.22	2.15
		Experimental	15	5	53.57	1.86
Case 2	2017-spring	Comparison	16	3	65.06	1.93
		Experimental	17	4	64.14	2.24

2.3.1 | Programming achievement test

The programming achievement test was developed by the researchers to assess basic programming concepts and the acquisition of programming logic. In the preparation phase of the test items, a programming test comprising 18 items developed by Saygıner (2017) for the Pascal programming language was used. The questions were prepared to be free of syntactic rules so that the achievement test is independent of programming language or mode. Initial format of the achievement test includes a total of 25 questions: five open-ended and 20 multiple choice items. The opinions from four experts with over 5 years of programming teaching experiences were obtained for the content validity of the test. The achievement test that was revised using expert opinions was applied to 206 vocational high school students who had taken programming lessons. The sub-upper group technique was used to perform item analysis. Students were first ranked according to their scores, and both 27% of lower and upper groups were selected. Subsequently, the item analyses were performed. After item analysis, two items with a discrimination index of less than 0.25 were excluded from the test. This was the final format of the achievement test. The average difficulty and discrimination indices of the test are 0.476 and 0.473, respectively, whereas the KR-20 internal consistency coefficient was found to be 0.878. An English version of sample test items is presented in Appendix. The achievement test is especially appropriate for students at a high school level and above. However, since test items are free of syntactic rules of any particular programming language, some items can also be applied to middle-school students to evaluate their acquisition of CT concepts.

2.3.2 | Abstraction test

To evaluate students' abstraction skills, the abstraction skill inventory developed by Hill et al. (2008) was used. Inventory addresses abstraction skills in the frame of three discrete cognitive dimensions comprising conceptual (five-point Likert-type three items), formal (six multiple-choice test items), and descriptive (three multiple-choice test items) abstraction. To bring the research on a more rational ground, only formal and descriptive abstraction tests that are non-self-reporting instruments were used in the study.

2.3.3 | Problem-solving scale

The problem-solving scale originally developed by Heppner (1988) and Heppner and Petersen (1982) and then adapted by Şahin, Şahin, and Heppner (1993) to the local language, in which the study was conducted, was used to evaluate the effects of CT activities on students' problem-solving approaches in the face of problematic situations in daily life. The high score of the scale shows that individuals perceive themselves incapable in problem-solving. The Cronbach alpha reliability coefficients of the original and adapted forms of the scale are high values such as 0.90 and 0.88, respectively. The reliability coefficients of the scale applied in the study were 0.76 and 0.88 for the pretests in Case 1 and Case 2, respectively.

2.3.4 | Instructional materials motivation scale

The instructional materials motivation scale, originally developed by Keller, and adapted by Kutu and Sözbilir (2011) to the local language in which the study was conducted, was used to evaluate the effect of different programming modalities and tools used on student motivation. The reliability coefficients of the original and adapted forms of the scale were 0.93 (Keller, 2010) and 0.83 (Kutu & Sözbilir, 2011), respectively, and the reliability values were calculated as 0.84 and 0.87 for the cases in the scope of the study. High scores obtained from the scale indicate a high level of motivation.

2.3.5 | Personal information form

The personal information form lists the items with questions regarding age, gender, personal computer (PC) ownership, daily average computer usage time, and programming experience.

2.4 | Data collection

Data were collected within the scope of the programming foundations course. This introductory-level programming course includes computational rules that require the understanding of basic computational concepts and sharing with others and employment of CT skills to propose a solution in the problem-solving settings. The course that is followed by theoretical and practical sessions, was allotted 3–4 hours per week. In both case studies, learning activities in the test groups were implemented under the guidance of the same instructor. This helped to minimize the possible differences of the effects of the instructors on the groups. The weekly course contents and activity plans prepared by the instructor of the course—the first author of the current study, were reviewed by a specialist in the field of instructional technology. Before implementations, the students were divided into two groups, and instructional activities related to CT were performed in one group through object-oriented visual programming tools (comparison group) and in the other through educational robot kits (Lego Mindstorms NXT 2.0) and software tools (experimental group) (Figure 1). C language-based programming tools (C# and Robot C) were used in both groups. This ensured similar conditions in terms of syntactic difficulty in programming. The learners in the experimental groups used Lego blocks to build the robots during the initial phase of the implementations. They were then expected to functionally manipulate the robots to improve their programming knowledge and skills. In fact, educational robot programming activities, which work as a tool to communicate with the external world, are considered as an alternative to traditional programming conditions for demonstrating CT skills. In the theoretical sessions of the course, short presentations and examples regarding basic information were presented for students to become familiar with basic programming concepts such as variables, operators, arithmetical and logical

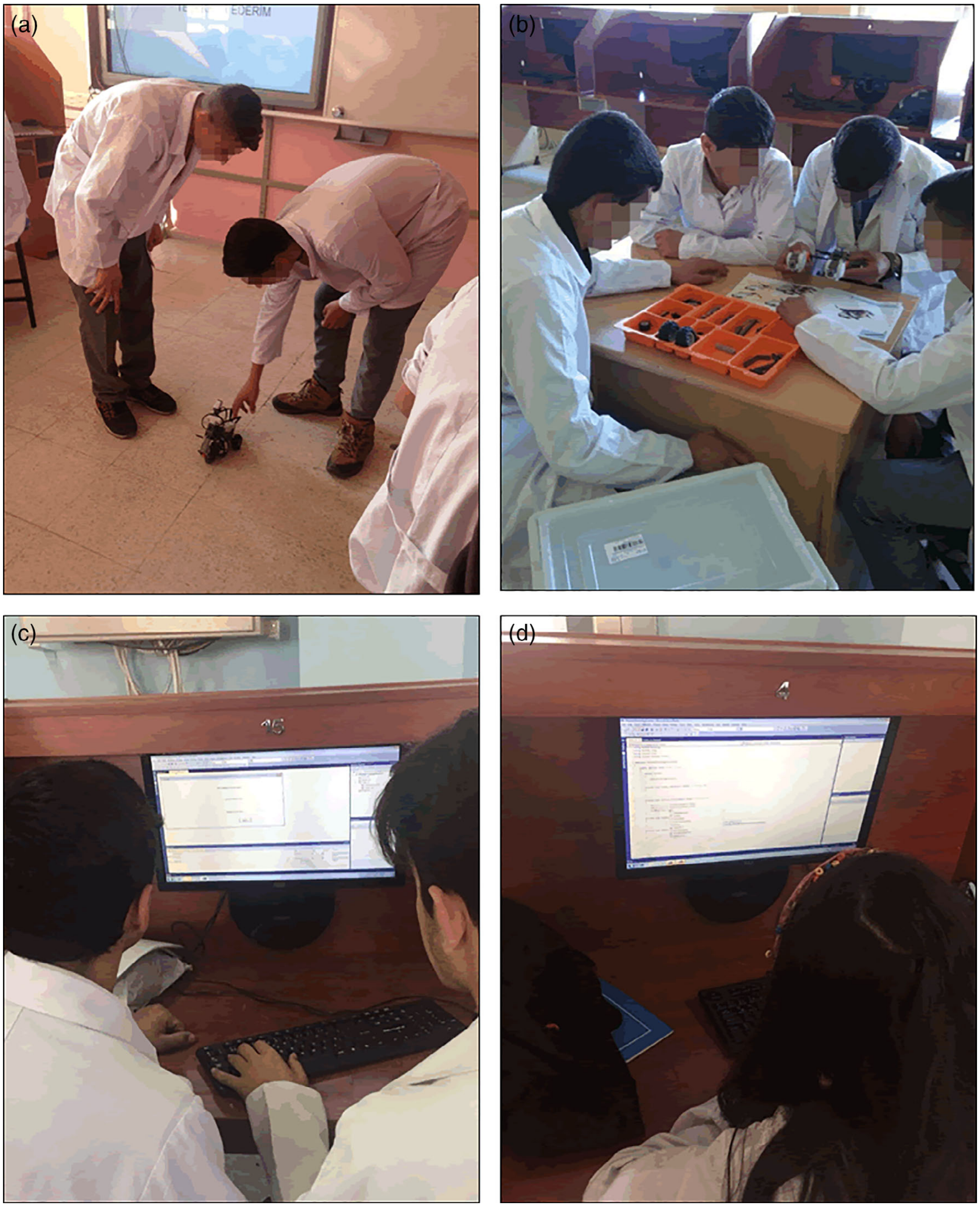

FIGURE 1 Images of experimental (a, b) and comparison (c, d) groups engaged in computational problem-solving activities through programming [Colour figure can be viewed at wileyonlinelibrary.com]

operations, conditional statements, and loops. During the practical session of the course, students tried to find solutions to the computational problems, and thereby had the opportunity to deepen their understanding of programming structures and processes. At this stage, problem-solving forms with the guideline questions suggested by Polya (1957, 1973) were provided to students in both groups to scaffold the problem-solving activities. The implementation period of Case 1 spanned 12 weeks; this period was reduced to 8 weeks in Case 2 by lowering the subject density. This aimed to avoid the problems of cognitive overload for novice programmers and prevent any problems (e.g., loss of subjects) that could be encountered during the long intervention phase.

Introduction of programming concepts to the test groups was performed in parallel in a temporal sense. Although it required the use of similar concepts, the structure and presentation of CT problems were differentiated according to the groups. The number of problems in the first weeks of practical sessions was gradually reduced in the following weeks as their complexity increased. Prerequisite conditions (robot components, physical settings, problem scenarios, etc.) were created for the experimental groups to perform robotic manipulations according to the CT problems (Figure 2). Students in the experimental groups worked in teams throughout the practical sessions. In contrast, students in the comparison groups were also given flexibility in working individually or as a team.

2.5 | Data analysis

Prior to data analysis, suitability of the dataset to normal distribution was examined using Q-Q and P–P plots, Shapiro–Wilk test, and Z-scores calculated regarding the skewness and kurtosis statistical measures. Pretest scores were compared between groups using the independent samples *t*-test or Mann–Whitney *U* test according to normality. In-group score differences pre- and post-implementations were investigated with the paired sample *t*-test or its nonparametric equivalent Wilcoxon signed-rank test. In the study, the relationships between variables were tested by correlation analysis. Posttest scores between groups were examined using mean difference analysis or covariance analysis depending on whether there is a significant relationship between pre- and posttest scores. In the analysis of the programming achievement scores, students' grade point average (GPA) was also considered as covariate. Statistical significance (p) threshold value was accepted to be .05.

2.6 | Limitations

In this study, there is a difference in terms of sample size between the two discrete case studies. Despite how carefully or rigorously the participant selection is performed for the interventional studies, especially with long implementation periods, it is possible that some participants may leave the study for a variety of reasons. Among the threats to internal validity (Fraenkel & Wallen, 2006), participant loss is probably the most difficult to control. Participant loss is a limitation of this study. A total of seven (14%) and six participants (13%) in Case 1 and Case 2, respectively were excluded from the study for various reasons (transfer, high absentee rate, special education requirement, etc.). Data obtained from these participants were removed from the dataset. Moreover, underrepresentation of female participants is another limitation of this study.

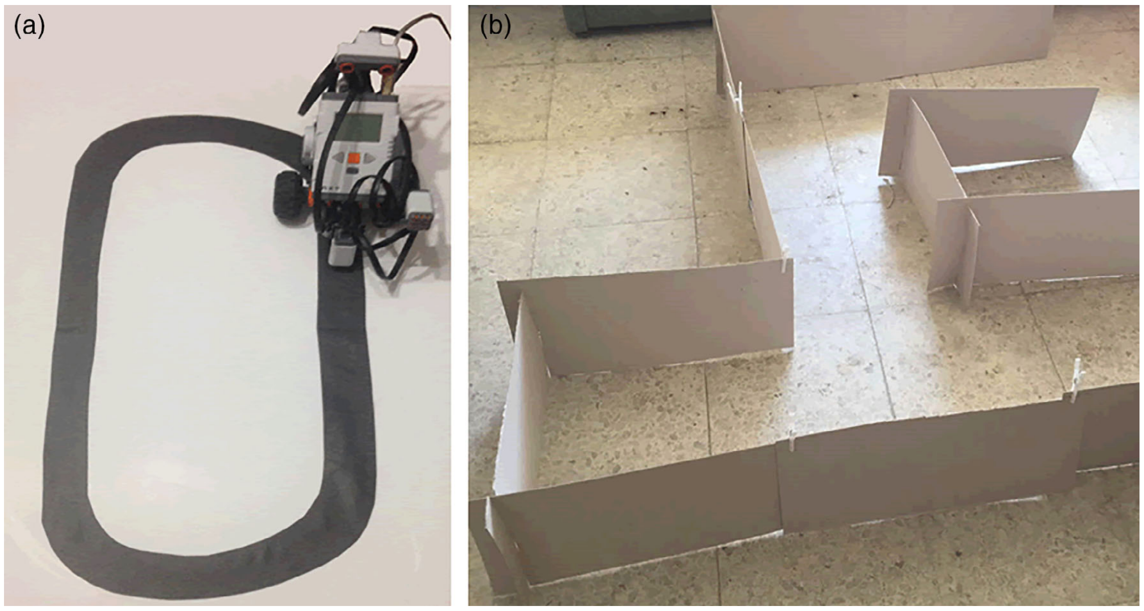

FIGURE 2 Examples of tracks prepared for robotic activities. Circular route for path tracking (a); and a maze created through removable cardboard blocks for maze solving (b) [Colour figure can be viewed at wileyonlinelibrary.com]

3 | RESULTS

The demographic characteristics of the participants are presented in Table 2. The age of the participants was generally between 15–16 years, and most of them had 4–6 years or 7–9 years of computer experience. The GPAs of students who participated in Case 2 are relatively higher than those in Case 1. Participants with a relatively high rate of PC ownership spent an average of 2–3 hours a day at the computer. However, the programming familiarity of the participants was considerably low as expected.

Difference between the students' prior knowledge about programming was evaluated by the pretest measurements. The programming pretest results showed that the participants did not differ significantly in terms of programming pre-knowledge level (Case 1: $t_{39} = .350$, $p = .728 > .05$; Case 2: $t_{38} = -.052$, $p = .958 > .05$). Following the CT activities, there was a significant increase in the programming test scores in all groups ($p < .001$). In both groups, the programming test scores of the students increased two times approximately, and they approached a medium level from the lower one after the implementations. In the comparison group, the programming mean score of the learners elevated from 5.952 to 9.904 ($t_{20} = 6{,}921$, $p < .001$) and from 6.579 to 10.579 ($t_{18} = 6.040$, $p < .001$) in Case 1 and Case 2, respectively. Similarly, programming test scores of the experimental group increased from 5.650 to 10.250 ($t_{19} = 8.027$, $p < .001$) and from 6.619 to 11.524 ($t_{20} = 7.147$, $p < .001$) in Case 1 and Case 2, respectively (Table 3 and Figure 3).

Cohen's d (>0.8) and η^2 (>0.14) values indicate that CT activities have a large effect on the increase in programming achievement in both experimental and comparison groups. Values in Table 3 clearly indicate that the posttest scores of the students in the robotic programming group are higher compared to those of the students in the object-oriented visual programming group in both case studies. While

TABLE 2 Students' descriptive statistics of age, GPA, PC ownership and experience, daily usage of PC, and familiarity of programming

		Case 1		Case 2	
Demographics	**Intervals**	**Frequency (*f*)**	**Percentage (%)**	**Frequency (*f*)**	**Percentage (%)**
Age (years)	15–16	34	83%	31	77.5%
	17–18	7	17%	9	22.5%
	Total	41	100%	40	100%
GPA (scores)	40–49	16	39%	2	5%
	50–59	12	29.3%	11	27.5%
	60–69	12	29.3%	16	40%
	70–79	1	2.4%	9	22.5%
	80–89	—	—	2	5%
	Total	41	100%	40	100%
Computer usage experience (years)	None	4	10%	3	7.5%
	0–1	2	5%	2	5%
	2–3	5	12%	5	12.5%
	4–6	19	46%	13	32.5%
	7–9	9	22%	13	32.5%
	10 and above	2	5%	4	10%
	Total	41	100%	40	100%
Home PC ownership	Yes	27	66%	28	70%
	No	14	34%	12	30%
	Total	41	100%	40	100%
Daily PC usage (hours)	None	8	19.5%	9	22.5%
	0–1	12	29.3%	10	25%
	2–3	13	31.7%	14	35%
	4–6	8	19.5%	4	10%
	7–9	—	—	2	5%
	10 and above	—	—	1	2.5%
	Total	41	100%	40	100%
Prior enrolment in a programming course	Yes	2	5%	5	12.5%
	No	39	95%	35	87.5%
	Total	41	100%	40	100%

examining the effect of programming modality for CT on students' achievement, the ninth-GPA and programming pretest results were included as covariates in the analysis. Pearson correlation coefficients (r) between the ninth-GPA and programming posttest scores are 0.692 and 0.433 for Case 1 and Case 2, respectively ($p < .01$). Similarly, the correlation between programming pretest and posttest results was found to be 0.749 and 0.475 for Case 1 and Case 2, respectively ($p < .01$). Analysis of the adjusted posttest results based on the pretest and GPA scores revealed that the experimental groups (X = 10.25 for Case 1 and X = 11.52 for Case 2) obtained relatively high posttest scores compared to those of the comparison groups (X = 9.90 for Case 1 and X = 10.58 for Case 2). However, the results obtained from covariance analysis in both cases showed that there was no statistically significant difference between the programming achievement posttest scores in terms of programming modality ($F_{1\text{-}37} = .884$, $p > .05$ for Case 1; $F_{1\text{-}36} = 1.422$, $p > .05$ for Case 2). The eta-squared statistic indicates a small effect size for both cases ($\eta^2 = .023$ for Case 1; $\eta^2 = .038$ for Case 2).

Another research question is how CT activities performed via programming affect the abstraction (descriptive and formal abstraction) skills of the learners. Programming is a computational activity that enables different levels of data and procedural abstraction. Formal abstraction deals with the ability to reason about symbolic situations and structures to comprehend the underlying ideas and achieve a more direct or simplified perspective (Hill et al., 2008). This process involves removing redundant details to concentrate on the main ideas and achieve simplification. Therefore, the test questions have both precise and less precise answers. The formal and descriptive abstraction test scores were analysed using nonparametric tests. Wilcoxon signed-ranks and Mann–Whitney *U* tests were employed for the comparisons within and among groups, respectively. Additionally, the covariance rank analysis proposed by Quade (1967) that is considered as the non-parametric equivalent of covariance analysis was used in the comparison of the posttest scores.

There was no significant difference between the groups in terms of formal abstraction skills before the implementations (U = 196.5, $p > .05$ for Case 1; U = 194.5, $p > .05$ for Case 2). Following the CT activities, the number of students with increased formal abstraction scores was relatively higher than those with decreased scores (except for the experimental group in Case 1). However, a significant difference was found

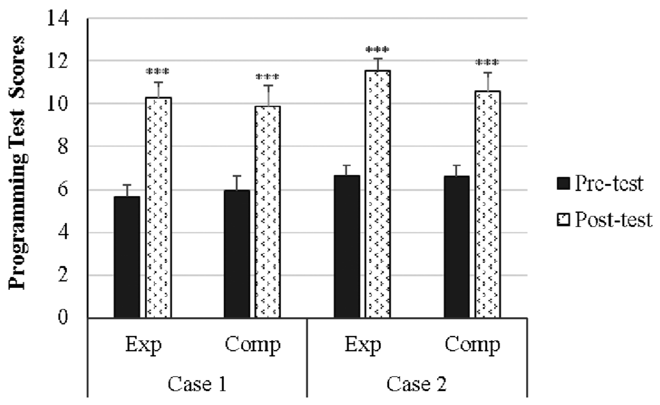

FIGURE 3 Change in students' programming achievement scores following the implementations. All comparisons were made between pretest and posttest results of the relevant test group. Error bars represent standard error of the means. Triple asterisk denotes $p < .001$

between the pretest-posttest scores of the experimental group in only Case 2 (Z = 2.00, $p < .05$) (Table 4 and Figure 4).

On the contrary, the statistical power value (power = 0.58 < 0.80) calculated for the test, leads to a limitation owing to the small sample size ($n = 21$). Based on the effect size (d = 0.4294) of the comparison between the experimental group pretest-posttest scores in Case 2, the sample size must be 37 to obtain a statistical power of 0.80. Although it is irrelevant in cases where the rank orders are not statistically significant when the effect size statistics related to other groups are examined, it is observed that the effect sizes are considerably low. In other words, instructional interventions have a considerably low effect on the development of students' formal abstraction scores. Lastly, the formal abstraction posttest scores of the experimental and comparison groups were compared following the CT activities. Before posttest comparison, the relationships between academic achievement and formal abstraction pre- and posttest scores were examined by Spearman's rank differences correlation analysis. In both cases, GPA scores did not have a significant relationship with formal abstraction posttest scores ($r_s = .239$, $p = .133 > .05$ for Case 1; $r_s = .074$, $p = .649 > .05$ for Case 2). Therefore, using the GPA variable as a covariate in the analysis of formal abstraction test scores was not considered necessary. However, the formal abstraction pretest scores

TABLE 3 Change of students' programming achievement scores following the implementations

	Group	Tests	N	$\bar{X}$	*SD*	*df*	t	*p*	Cohen's d	η^2
Case 1	Object-oriented visual prog.	Pretest	21	5.952	3.074	20	6.921	.000	1.510	.705
		Posttest	21	9.904	4.380					
	Educational robotic prog.	Pretest	20	5.650	2.390	19	8.027	.000	1.795	.772
		Posttest	20	10.250	3.385					
Case 2	Object-oriented visual prog.	Pretest	19	6.579	2.411	18	6.040	.000	1.386	.669
		Posttest	19	10.579	3.906					
	Educational robotic prog.	Pretest	21	6.619	2.418	20	7.147	.000	1.559	.718
		Posttest	21	11.524	2.676					

are included in nonparametric analyses as a covariate. According to the results of Quade's rank analysis of covariance, there is no significant difference between the test groups in terms of formal abstraction posttest scores (Quade F = 0.661, *p* = .421 > .05 for Case 1; Quade F = 0.255, *p* = .616 > .05 for Case 2).

Descriptive abstraction corresponds to the generalization process to define a common basis or essence. It is the ability to recognize the most substantial characteristics and infer meaningful but shortened inclusive models based on these characteristics. This involves identifying key points, focusing on clear and direct analysis, obtaining to the crux of the topic, and comprehending the main principles. Formal abstraction is more related to problem solving, whereas descriptive abstraction deals with language manipulation. In both cases, there was no significant difference according to the descriptive abstraction levels between the groups before the implementations (U = 210.00, *p* > .05 for Case 1; U = 187.50, *p* > .05 for Case 2). Regarding descriptive abstraction pretest scores in both cases, the number of students from the experimental groups with significantly increased scores was higher than those with decreased scores (Z = 2.070, *p* < .05 for Case 1; Z = 2.530, *p* < .05 for Case 2) (Table 5 and Figure 5). On the contrary, in both case studies, there was no significant difference between the pre- and posttest scores of the comparison group students in favour of the positive scores (Z = 1.355, *p* > .05 for Case 1; Z = –.447, *p* > .05 for Case 2).

The effect size coefficient, r, to be used to evaluate the magnitude of the pretest-posttest score differences of the experimental group was calculated as 0.327 and 0.390 in Case 1 and Case 2, respectively. This indicates that instructional intervention in the experimental group had a moderate effect on the pretest-posttest score differences. Power analysis showed that the power of statistical analysis for the comparison of the pretest-posttest scores of the experimental group was 0.68 and 0.90, for Case 1 and Case 2, respectively. These results reveal that CT and problem-solving activities fulfilled through educational robot programming can be used to support students' descriptive abstraction skills.

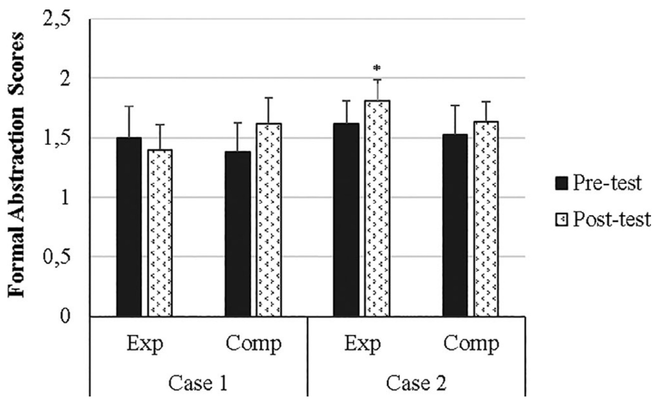

FIGURE 4 Comparison of students' formal abstraction test scores following the interventions. Error bars represent standard error of the means. Single asterisk denotes *p* < .05

In both case studies, no significant relationship of GPAs with neither pretest (r_s = –.162, *p* = .310 > .05 for Case 1; r_s = –.137, *p* = .398 > .05 for Case 2) nor posttest (r_s = .178, *p* = .267 > .05 for Case 1; r_s = –.149, *p* = .360 > .05 for Case 2) results of the descriptive abstraction was determined. Therefore, using the GPA variable as a covariate in the analysis of descriptive abstraction scores was not required. However, descriptive abstraction pretest scores were included in nonparametric analyses as a covariate. The posttest scores revealed no statistically significant difference between the test groups according to the descriptive abstraction in Case 1 (Quade F = 0.337, *p* = .565 > .05). On the contrary, the rank order differences between the groups were significant in Case 2 (Quade F = 6.401, *p* = .016 < .05).

A problem-solving scale was applied to examine how CT activities affect students' reactions or approaches to problems in their daily lives. No significant difference was found between the test groups in terms of problem-solving scores before pre-implementation phases (t (39) = 1.313, *p* > .05 for Case 1; t(30,179) = –.166, *p* > .05 for Case 2). This can be interpreted as—the test groups have similar approaches to

TABLE 4 Analysis of change in students' formal abstraction test scores following the interventions

	Group	Posttest-pretest	n	Mean rank	Sum of ranks	Z	p	r
Case 1	Object-oriented visual programming	Negative ranks	7	8.00	56	.688	.491	.106
		Positive ranks	9	8.89	80			
		Ties	5					
	Educational robotic programming	Negative ranks	7	7.93	55.50	–.193	.847	–.030
		Positive ranks	7	7.07	49.50			
		Ties	6					
Case 2	Object-oriented visual programming	Negative ranks	3	2.50	7.50	.649	.516	.105
		Positive ranks	3	4.50	13.50			
		Ties	13					
	Educational robotic programming	Negative ranks	0	.00	.00	2.000	.046*	.309
		Positive ranks	4	2.50	10.00			
		Ties	17					

Note: Single asterisk denotes *p* < .05.

TABLE 5 Analysis of students' descriptive abstraction test scores post interventions

	Group	Posttest-pretest	n	Mean rank	Sum of ranks	Z	p	r
Case 1	Object-oriented visual programming	Negative ranks	5	6.50	32.50	1.355	.175	.209
		Positive ranks	9	8.06	8.06			
		Ties	7					
	Educational robotic programming	Negative ranks	2	4.00	8.00	2.070*	.038	.327
		Positive ranks	8	5.88	47.00			
		Ties	10					
Case 2	Object-oriented visual programming	Negative ranks	3	3.00	9.00	−.447	.655	−.073
		Positive ranks	2	3.00	6.00			
		Ties	14					
	Educational robotic programming	Negative ranks	0	0.00	0.00	2.530*	.011	.390
		Positive ranks	7	4.00	28.00			
		Ties	14					

Note: Single asterisk denotes $p < .05$.

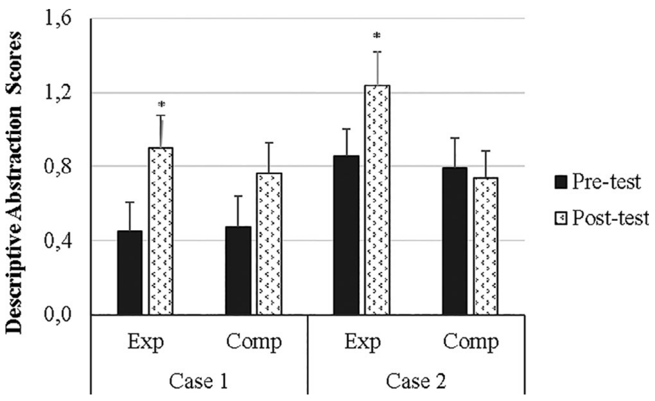

FIGURE 5 Comparison of students' descriptive abstraction scores following the interventions. Error bars represent standard error of the means. Single asterisk denotes $p < .05$

daily problems before implementations. The students overall had a moderate level of competency perception in problem-solving. After implementations, no significant change was found in the problem-solving skills perceptions of the students in both experimental and comparison groups in both cases ($p > .05$) (Table 6 and Figure 6).

Results of from the covariance analysis, wherein the problem-solving pretest results were considered as a covariate, showed that the programming modality did not cause a significant difference in terms of students' problem-solving skills. ($F_{1\text{-}38} = 1.805$, $p > .05$ for Case 1; $F_{1\text{-}37} = .770$, $p > .05$ for Case 2).

In this study, the motivation scale was applied at the post-intervention phase to reduce the novelty effect of the programming tools and environments used for computational activities on student motivation. In both cases, the motivation scores of the learners showed a significant difference in favour of the experimental group ($t_{39} = -2.295$, $p < .05$ for Case 1; $t_{38} = -2.211$, $p < .05$ for Case 2) (Table 7 and Figure 7). Cohen's d values between 0.7 and 0.8 indicate a moderately high effect. Moreover, it can be said that approximately 11–12% of the variance observed in the motivation scores occurred depending on the instructional technology.

In this study, relationships between programming achievement, problem-solving perception, abstraction ability (formal and descriptive abstraction), and motivation were examined based on the posttest results (Table 8). The data were analysed collectively without classification according to test groups.

In Case 1, which lasted for 12 weeks, moderate positive and significant relationships were found between (a) students' programming achievement and formal abstraction scores ($r_s = .384$, $p < .05$), (b) programming achievement and descriptive abstraction scores ($r_s = .353$, $p < .05$), and (c) formal and descriptive abstraction scores ($r_s = .352$, $p < .05$). However, these relationships observed between the three variables in Case 1 were not significant in Case 2, for which, the implementation process lasted for 8 weeks ($p > .05$). In both case studies, a moderate negative relationship was determined to exist between the problem-solving and motivation scores ($r_s = -.475$, $p < .01$ for Case 1; $r_s = -.542$, $p < .01$ for Case 2). Lower scores obtained from the problem-solving scale indicate a higher level of problem-solving perception. Therefore, as students' perceptions of problem-solving skills increased, their motivation also increased in both cases. Problem-solving and motivation scales are both based on self-reporting data. It is thought that collecting data from both instruments based on student perceptions contributes to the relationships between these two variables.

4 | DISCUSSION

This study comparatively examines the effects of object-oriented programming and robot programming activities on achievement, abstraction, problem solving, and motivation. The results of the study are summarized in Table 9.

After CT and problem-solving activities through programming, there was a significant increase (approximately two times) in the

TABLE 6 Analysis of learners' problem-solving perceptions according to test groups

	Group	Tests	N	$\bar{X}$	SD	df	t	p
Case 1	Object-oriented visual programming	Pretest	21	100.857	15.58	20	.564	.579
		Posttest	21	103.381	19.02			
	Educational robotic programming	Pretest	20	93.700	19.22	19	−.455	.654
		Posttest	20	92.050	19.97			
Case 2	Object-oriented visual programming	Pretest	19	88.6316	12.18	18	1.882	.076
		Posttest	19	94.1579	17.02			
	Educational robotic programming	Pretest	21	89.619	24.15	20	.903	.377
		Posttest	21	91.7619	23.48			

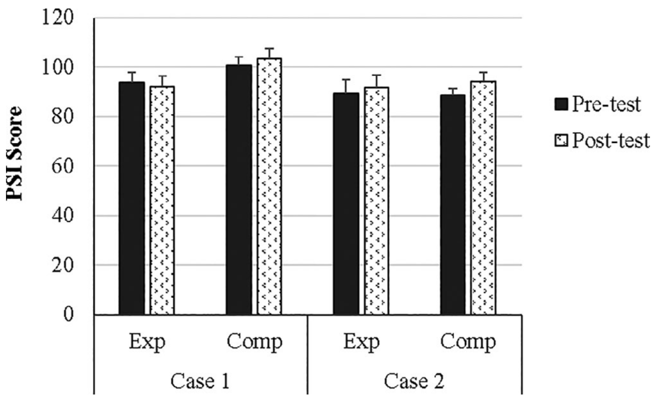

FIGURE 6 Change in learners' problem-solving perceptions after the implementations according to test groups. Error bars represent standard error of the means

programming achievement scores of both experimental and comparison groups. Furthermore, instructional interventions showed a high impact on students' programming success. In both case studies, the gain scores of the experimental group were higher than those of the comparison group. However, this difference in posttest scores between groups was not significant for both case studies.

In this study, all learning conditions except programming modality were designed to be as similar as possible to each other for the experimental and control groups. Moreover, in both groups, programming tools based on the C language were used to support students' CT skills. Both Visual C# and Robot C have the technical and syntactical complexities of text-based programming languages. Therefore, these findings regarding achievement scores may be the result of learning activities implemented in considerably similar conditions, especially code-writing procedures across the groups.

Several studies report the positive effects of educational robotic kits and software tools on supporting students' programming skills. On the contrary, there is a dearth of studies with comparison of educational robot programming and traditional programming activities. Garcia and De la Rosa (2016) reported that the middle school students' acquisition of programming and algorithm concepts increased significantly (91%) as a result of the robotic activities performed in a virtual environment through a web-based visual-block programming application. They additionally compared the effectiveness of the web-based virtual robotics application using Scratch, a popular block-programming tool. At the end of the five-hour experimental intervention, both tools were found to have a similar effect on student learning. The study conducted by Scott et al. (2015) to determine the impact of web programming and robotic applications on the programming process in the introductory programming courses reported that undergraduate students engaged in robotic activities allocated more time for programming and produced higher-quality programs in terms of functionality and sophistication. In that study, it has been stated that the time allocated for coding practices has a significant effect on the functionality of the program. Gender did not produce a significant difference in the quality of the created programs. On the other hand, the instructional activities organized for the experimental and comparison groups in the study were performed sequentially (1-semester gap) within the scope of different courses, and not simultaneously. Moreover, student assessment techniques (e.g., midterm exams, weekly tasks) differed according to the test groups. Overall, apart from the manipulations regarding the variables examined in the study, the instructional conditions arranged for the test groups significantly

TABLE 7 Analysis of motivation scores between test groups with *t*-test after implementations

	Test	Group	N	$\bar{X}$	SD	df	t	p*	Cohen's d	η^2
Case 1	Motivation scale	Object-oriented visual programming	21	85.52	12.53	39	−2.295	.027	0.717	0.114
		Educational robotic programming	20	93.65	9.92					
Case 2	Motivation scale	Object-oriented visual programming	19	80.79	15.50	38	−2.211	.033	0.700	0.113
		Educational robotic programming	21	90.05	10.78					

Note: *Single asterisk denotes p < .05.

differed from each other. Findings of the current and other studies in literature indicate that educational robots are as effective as traditional programming tools in supporting programming skills. However, more comparative studies are needed in this area.

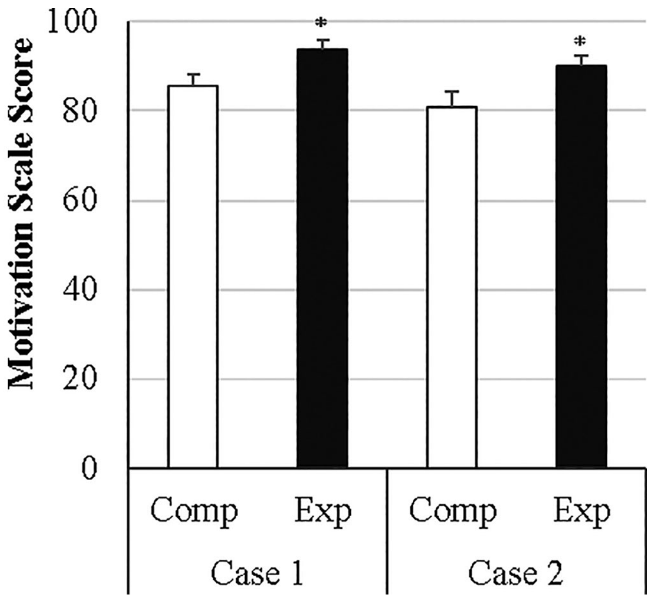

FIGURE 7 Comparison of motivation scores between test groups according to case studies. Error bars represent standard error of the means. Single asterisk denotes *p* < .05

The formal abstraction test results demonstrate no significant improvement in the formal abstraction skills of both experimental and comparison groups in Case 1 after CT activities. However, in Case 2, a significant increase was seen in the formal abstraction scores of students in the educational robotics group only. In both cases, no significant difference was observed between the experimental and comparison groups in terms of formal abstraction posttest scores. The descriptive abstraction results indicate a significant increase in the experimental groups following the implementation in both cases; however, a similar result was not seen for the comparison group in any case. No significant difference was observed between the test groups in terms of descriptive abstraction posttest scores in Case 1; however, a significant difference in favour of the experimental group in Case 2 was present. These results show that educational robot programming practices support the development of students' abstraction skills more than traditional object-oriented programming activities. It can be concluded that the descriptive abstraction skills of the experimental group students showed steady development in both cases compared to those in formal abstraction. This may be owing to the fact that the problem statements presented to the experimental group students in the practical sessions of the course are longer and more complex, thereby requiring higher-level descriptive abstractions.

Abstraction is a conceptual process, in which inclusive inferences based on basic characteristics are acquired by ignoring redundant or irrelevant details. Therefore, abstraction skill or creating an abstraction requires filtering out less important features while highlighting important aspects of the ideas to conceptualize and

TABLE 8 Relationship between programming achievement; problem solving; formal and descriptive abstraction; and motivation scores according to case studies

			Case 1				
			Programming achievement	Formal abstraction	Descriptive abstraction	Problem-solving	Motivation
Case 2	Programming achievement	r	—	.384[b]	.353[b]	−.208[a]	.258[b]
		p		.013*	.024*	.193	.104
		N		41	41	41	41
	Formal abstraction	r	.064[b]	—	.352[b]	.065[b]	.080[b]
		p	.693		.024*	.684	.620
		N	40		41	41	41
	Descriptive abstraction	r	.208[b]	−.191[b]	—	.169[b]	.181[b]
		p	.198	.238		.292	.258
		N	40	40		41	41
	Problem-solving	r	−.180[a]	.255[b]	−.219[b]	—	−.475[b]
		p	.266	.162	.175		.002**
		N	40	40	40		41
	Motivation	r	.152[b]	−.035[b]	.252[b]	−.542[b]	—
		p	.351	.829	.117	.000**	
		N	40	40	40	40	

Note: *$p < .05$, **$p < .01$.
[a]Pearson's correlation coefficient.
[b]Spearman's rank correlation coefficient.

TABLE 9 Examination of the differences in scores of the test groups according to dependent variables

		Differences within group		
Variables	**Case**	**Object-oriented visual programming**	**Educational robotic programming**	**Differences between groups**
Achievement	Case 1	Yes[a]	Yes[a]	No
	Case 2	Yes[a]	Yes[a]	No
Formal abstraction	Case 1	No	No	No
	Case 2	No	Yes[a]	No
Descriptive abstraction	Case 1	No	Yes[a]	No
	Case 2	No	Yes[a]	Yes[b]
Problem solving	Case 1	No	No	No
	Case 2	No	No	No
Motivation	Case 1	N/A	N/A	Yes[b]
	Case 2	N/A	N/A	Yes[b]

[a]In favour of posttest.
[b]In favour of experimental group.

represent the core ideas or processes with more general definitions (Weintrop et al., 2016). Computational activities performed through programming can be used to develop students' intellectual thinking and problem-solving skills (Feurzeig, Papert, & Lawler, 2011; Papert, 1993). However, similar to other disciplines, the prerequisite for reasoning in the subject areas of computer science is to create appropriate mental models or representations with essential abstractions in essence. CT inherently comprises multi-level abstractions. Programming (Voogt, Fisser, Good, Mishra, & Yadav, 2015), one of the basic forms of CT, has rich references in terms of components that can promote abstraction, particularly data and procedural abstraction. For example, the expression x = x + 1 is a procedure for assigning a value that requires a considerably different abstraction from a math equation. According to Papert (1980), through programming, students can gain an intuitive sense of quantity without their physical existence, related to some qualitative features, and work on abstract elements. When students program the computer to accomplish their desired goals, they think about what they can do on their own and develop an inner reflection of how they think. In summary, programming environments that encourage how to think rather than what to think, have the potential to be effective instructional tools for cognitive processes. On the contrary, learning to program is a considerably challenging task. Programming activities, especially for the novice, involve a wide variety of requirements and challenges. Programming courses are therefore considered difficult, and they often have high dropout rates (Alturki, 2016; Robins et al., 2003). The large number of difficulties encountered in computer programming instruction stems from the structure of the traditional tasks in computer science education and their presentation forms (Ben-Ari, 2001; diSessa, 2001; Guzdial & Forte, 2005). The traditional approach of teaching programming is to ask students to memorize commands specific to a particular programming language and expect them to follow syntactic rules for that language to create a series of programs starting from easy to difficult levels of complexity (Chang, 2014). In higher education and vocational/technical secondary education institutions, professional software development tools such as C# or Visual Basic are generally used in introductory programming courses. This gives rise to several difficulties, especially for novice programmers. The complex interface of professional programming environments, abundance of object or code blocks, and highly procedural code-writing rules introduce difficulties in establishing the semantic and syntactic relationship of programming, thereby limiting the demonstration of computational skills. Moreover, this hinders students' programming performance, participation and learning engagement, motivation and career choices related to computer science (Felleisen, Findler, Flatt, & Krishnamurthi, 2004). In summary, traditional programming-teaching approaches neglect students' cognitive role; therefore, they fall short of supporting the development of students' innovative and creative thinking skills (Fang, Chen, Cai, Cui, & Harn, 2017).

Educational robot programming is proposed as an alternative programming activity in this study. Educational robots embody programming, which is a highly conceptual process, within real-world settings. This facilitates the establishment of semantic and syntactic relationship of program codes. There is strong consensus in literature that abstraction is a developable skill. In this study, a wide variety of activities, particularly problem solving and programming, were held to exhibit and support abstraction skills. Additionally, through problem-solving forms, students were encouraged to determine the limit or scope of the problem, build models for further examination of the problem, and express the solution ideas or suggestions in a specific format (such as algorithm, flow charts, and situation scenarios) with the aim to make them functional, that is, to convert them into program codes more easily. These representations are thought to contribute to abstraction practices. Algorithms and flow charts not only enable expressing and reasoning of mental processes but also allow modelling of the proposed solutions. Such models serve as a scaffold for the functionalization of existing information and understanding by ensuring that mental representations are expressed in relation to each other. Recent studies have shown that paper-pencil-based CT activities such as algorithms and flow charts can be as effective as programming tools (Kim, Kim, & Kim, 2013).

In the current study, although students in both experimental and comparison groups studied the same subjects in the field of computer science, they had different experiences regarding abstraction and decomposition, which are the main components of CT. In addition to the computational concepts introduced in each activity throughout the curriculum, the experimental group had the opportunity to (a) examine complex processes in the physical world through concrete robotics, (b) observe and control robot movements, and thereby (c) regulate the relationships between the robot and the physical world.

The physical world is more intuitive and meaningful, especially for novices (Papert, 1980). The key benefit of visualization of the program outputs is that it enables learners to follow the work of their programmed objects, observe the relationships between all program elements simultaneously, handle the program results as a whole, and thereby find an opportunity to further develop the programming logic (Città et al., 2019). Robotics also enables the creation of highly functional feedback points for the written programs. This enables students to gain important insights into the process of modifying or rebuilding the program (Berland & Wilensky, 2015; Soleimani, Herro, & Green, 2019). In this study, the students in the experimental group developed the programs they initially wrote, in iterative steps based on their observations and insights. Moreover, thanks to concrete robotics, they became aware of the work of other teams, which lead them to establish norms to evaluate their own development.

Robotic activities are highly suitable for supporting student autonomy and exhibiting creative, strategic, and metacognitive thinking skills (Noh & Lee, 2020). Robotic technologies encourage students to think about abstract structures because students envision robot movements in their minds before programming. In this respect, robotic technologies based on the active manipulation of authentic materials allow abstract concepts to be connected to something that can be observed and imagined (Burleson et al., 2018).

In this study, the Lego educational robotic kits were chosen for the robotic applications. The Lego Mindstorms robotic kit not only offers learners a highly manipulative learning activity thanks to its several physical components (e.g., building blocks, servo motors, multi-type sensors) but also enables the functionalization of these building blocks with a variety of simple-to-use programming tools (Flórez et al., 2017). Lego robotics is an exploratory activity that supports "learning by tinkering" (Bers, Flannery, Kazakoff, & Sullivan, 2014). Tinkering-oriented learning describes having or requiring plans in the process of writing or modifying a computer program without relying on any theory. Tinkering includes a combination of activities such as trial-error, making small changes, and unveiling and using feedback mechanisms (such as tests). The iterative and variable nature of tinkering can help novices learn to program. The bottom-up approaches or guess-check processes (Burke & Kafai, 2010; Resnick et al., 2009) involved in "learning by tinkering" support the discovery of key components in the programming language or environment (Perkins, Hancock, Hobbs, Martin, & Simmons, 1986). Several researchers state that tinkering is a useful activity for novice programmers (Wu, Hu, Ruis, & Wang, 2019). Tinkering is an enjoyable activity that includes exploring and just-in-time planning. It also encourages questioning the causes of an action that produces the desired result and generates feedback from the environment (Berland, Martin, Benton, Smith, & Davis, 2013). In this respect, robot programming gains significance as a learning activity that offers highly convenient conditions for tinkering.

Despite promoting different learning trajectories, in the literature, physical and virtual robotic education have been reported to provide relatively similar student outcomes in terms of the acquisition of CT concepts and programming skills (Berland & Wilensky, 2015; Zhong, Zheng, & Zhan, 2020). It is quite reasonable to benefit from the pedagogical values of virtual robotics, especially regarding engineering thinking and complex problem-solving. Virtual robotics is also recommended for improving attitudes toward programming and to reduce cognitive load in programming education (Zhong et al., 2020). Two or three-dimensional virtual robot simulations offered by some programming tools such as Robot C virtual robot and Robot Virtual Worlds can be used in educational settings in which purchase costs and storage facilities are limited. Virtual robot applications eliminate technical problems and malfunctions that are sometimes encountered in physical robotics such as problems with batteries and sensors. In addition, simulation tools can allow robotic activities to be easily integrated into distance education beyond face-to-face education.

CT activities did not result in significant advancements in students' problem-solving approaches in this study. In addition, no difference was observed between the test groups in terms of problem-solving skills in both cases. Some studies in literature revealed similar findings on CT activities and problem-solving approaches (Çiftci & Bildiren, 2019; Kalelioğlu & Gülbahar, 2014). Kalelioğlu and Gülbahar (2014) found that programming activities with Scratch did not cause a significant difference in primary school students' problem-solving skills. In another study conducted by Kalelioğlu (2015) involving primary school fourth-grade students, online coding activities (code.org) did not cause a significant difference to students' reflective thinking skills toward problem-solving.

In the current study, it was observed that the conducted problem-solving and CT activities were not sufficiently effective on students' approaches to real-life problems. To offer programming activities or conditions to students is not solely sufficient for effective learning; however, the transfer of the learned matter to different disciplines and real-world problems is additionally required. Therefore, selection of problem scenarios in CT activities that are more related to daily life may facilitate the transfer of computational skills to real-world problems. Furthermore, in addition to CT skills, practices that encourage metacognitive thinking in the problem-solving process can provide important opportunities for students to develop problem-solving approaches.

In both cases, learner motivation significantly differentiated in favour of the educational robot programming group. Study data were collected following the implementations to eliminate any possible differences owing to the novelty effect of educational technologies on motivation scores. Therefore, it is considered that the difference in motivation scores is not innovation-induced. In several studies on the construction of computational artefacts, student participation and motivation were highlighted rather than learning about the subject matter (Wilkerson-Jerde, 2014). Saez-Lopez and Sevillano-Garcia (2017) determined in their study on CT integration in art education that robotic technologies and programming activities increase student interest, motivation, and participation in addition to computational competencies.

5 | CONCLUSION AND RECOMMENDATIONS

The past half-century has seen the generation of a comprehensive body of literature regarding computer science and programming education. However, these studies were mainly conducted in the context of undergraduate classes. Recently, increasing initiatives to integrate computer science into the K12 curriculum, including CT and programming activities are being conducted worldwide. However, studies on how CT processes should be guided, what learning outcomes would be obtained at the end of the process, and how these outcomes would be evaluated are notably limited (Grover & Pea, 2013; Lye & Koh, 2014). In the current study, the effects of CT experiences on students' programming achievement, abstraction skills, problem-solving approaches, and motivation were examined through the programming activities, in which they try to find solutions to computational problems. To demonstrate CT skills more effectively, students are encouraged to perform the problem-solving process under the guidance of the problem-solving steps proposed by Polya (1957, 1973). Their problem-solving process has been scaffolded in this way. The study was conducted with vocational secondary education students—a group of participants that is relatively underrepresented in the literature.

The results revealed that educational robot programming is as effective as traditional programming activities in enhancing learners' programming competencies. Additionally, more positive outcomes regarding abstraction were obtained in the group engaged in robotic activities. It was determined that the students in this group were more successful in simplifying the problem statements and in revealing their outlines. However, there was no difference between the test groups in the reasoning related to symbolic structures. Based on these results, instead of developing the abstraction skill as a whole, it is recommended to focus on instructional interventions and problem-solving processes that will be conducted according to the types of abstraction. Employment of instructional design elements that are customized according to different types of abstraction may yield more effective results.

In this study, CT and problem-solving activities did not significantly affect students' approach to real-world problems and their perception of problem-solving skills. Therefore, if problems more relevant to daily life are chosen to develop students' CT skills, it becomes easier for them to transfer their computational skills to real-world problems. Furthermore, it is considered that the effect of robot programming activities on student motivation may contribute positively to students' computer science and STEM career preferences.

The participants in the current study majorly represent the population with low academic performance. Future studies involving participants with a relatively higher level of academic achievement or from different educational levels can provide new insights to the current understanding of the relationship between CT and educational robot programming activities. The current research is based on two cross-sectional experimental interventions. Additional cross-sectional studies or collection of longitudinal data can reveal different perspectives on the variables under investigation.

ACKNOWLEDGEMENTS

This work is derived from the first author's doctoral dissertation under the supervision of the second author. The authors would like to thank Prof. Arif Altun, Prof. Selçuk Özdemir, and anonymous reviewers for their constructive feedback. The authors would also like to thank Hacettepe Technopolis Technology Transfer Centre for advanced language editing service. There has been no financial support for this work that might have influenced its outcome.

DISCLOSURE STATEMENT

The authors declare that there are no conflicts of interest associated with this publication. This manuscript is the original work of authors and all authors mutually agree for its submission.

PEER REVIEW

The peer review history for this article is available at https://publons.com/publon/10.1111/jcal.12495.

DATA AVAILABILITY STATEMENT

The data are available from the corresponding author upon reasonable request.

ORCID

Murat Çınar https://orcid.org/0000-0003-4012-4174
Hakan Tüzün https://orcid.org/0000-0003-1153-5556

REFERENCES

Alturki, R. A. (2016). Measuring and improving student performance in an introductory programming course. *Informatics in Education*, *15*(2), 183–204. https://doi.org/10.15388/infedu.2016.10

Basawapatna, A. (2016). Alexander meets Michotte: A simulation tool based on pattern programming and phenomenology. *Educational Technology & Society*, *19*(1), 277–291.

Ben-Ari, M. (2001). Constructivism in computer science education. *Journal of Computers in Mathematics and Science Teaching*, *20*(1), 45–73.

Berland, M., Martin, T., Benton, T., Smith, C. P., & Davis, D. (2013). Using learning analytics to understand the learning pathways of novice programmers. *The Journal of the Learning Sciences*, *22*(4), 564–599.

Berland, M., & Wilensky, U. (2015). Comparing virtual and physical robotics environments for supporting complex systems and computational thinking. *Journal of Science Education and Technology*, *24*(5), 628–647.

Bers, M. U., Flannery, L., Kazakoff, E. R., & Sullivan, A. (2014). Computational thinking and tinkering: Exploration of an early childhood robotics curriculum. *Computers & Education*, *72*, 145–157.

Bork, A. (1981). *Learning with computers*. Bedford, MA: Digital Press.

Burke, Q., & Kafai, Y. B. (2010). Programming & storytelling: Opportunities for learning about coding & composition. Paper presented at: 9th International Conference on Interaction Design and Children (pp. 348–351). Barcelona, Spain: ACM.

Burleson, W. S., Harlow, D. B., Nilsen, K. J., Perlin, K., Freed, N., Jensen, C. N., … Muldner, K. (2018). Active learning environments with robotic tangibles: Children's physical and virtual spatial programming experiences. *IEEE Transactions on Learning Technologies*, *11*(1), 96–106.

Carlborg, N., Tyrén, M., Heath, C., & Eriksson, E. (2019). The scope of autonomy when teaching computational thinking in primary school. *International Journal of Child-Computer Interaction*, *21*, 130–139. https://doi.org/10.1016/j.ijcci.2019.06.005

Chang, C. K. (2014). Effects of using Alice and Scratch in an introductory programming course for corrective instruction. *Journal of Educational Computing Research*, *51*(2), 185–204.

Città, G., Gentile, M., Allegra, M., Arrigo, M., Conti, D., Ottaviano, S., … Sciortino, M. (2019). The effects of mental rotation on computational thinking. *Computers & Education*, *141*, 103613. https://doi.org/10.1016/j.compedu.2019.103613

Colburn, T., & Shute, G. (2007). Abstraction in computer science. *Minds and Machines*, *17*(2), 169–184.

Çiftci, S., & Bildiren, A. (2019). The effect of coding courses on the cognitive abilities and problem-solving skills of preschool children. *Computer Science Education*, *30*(1), 3–21. https://doi.org/10.1080/08993408.2019.1696169

diSessa, A. A. (2001). *Changing minds: Computers, learning, and literacy* (1st ed.). Cambridge, MA: MIT Press.

Doleck, T., Bazelais, P., Lemay, D. J., Saxena, A., & Basnet, R. (2017). Algorithmic thinking, cooperativity, creativity, critical thinking, and problem solving: Exploring the relationship between computational thinking skills and academic performance. *Journal of Computers in Education*, *4*(4), 355–369. https://doi.org/10.1007/s40692-017-0090-9

Durak, H. Y. (2020). Modeling different variables in learning basic concepts of programming in flipped classrooms. *Journal of Educational Computing Research*, *58*(1), 160–199. https://doi.org/10.1177/0735633119827956

Fang, A. D., Chen, G. L., Cai, Z. R., Cui, L., & Harn, L. (2017). Research on blending learning flipped class model in colleges and universities based on computational thinking – "database principles" for example. *Eurasia Journal of Mathematics Science and Technology Education*, *13*(8), 5747–5755.

Felleisen, M., Findler, R. B., Flatt, M., & Krishnamurthi, S. (2004). The TeachScheme! project: Computing and programming for every student. *Computer Science Education*, *14*(1), 55–77.

Feurzeig, W., Papert, S. A., & Lawler, B. (2011). Programming-languages as a conceptual framework for teaching mathematics. *Interactive Learning Environments*, *19*(5), 487–501.

Flórez, B. F., Casallas, R., Hernández, M., Reyes, A., Restrepo, S., & Danies, G. (2017). Changing a generation's way of thinking: Teaching computational thinking through programming. *Review of Educational Research*, *87*(4), 834–860.

Fraenkel, J. R., & Wallen, N. E. (2006). *How to design and evaluate research in education* (6th ed.). New York, NY: McGraw-Hill.

Garcia, P. G. F., & De la Rosa, F. (2016). RoBlock–web app for programming learning. *International Journal of Emerging Technologies in Learning*, *11*(12), 45–53.

Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational Researcher*, *42*(1), 38–43.

Guzdial, M., & Forte, A. (2005). Design process for a non-majors computing course. *ACM SIGCSE Bulletin*, *37*(1), 361–365.

Heppner, P. P. (1988). *The problem solving inventory (PSI): Research manual*. Palo Alto, CA: Consulting Psychologists Press.

Heppner, P. P., Hibel, J., Neal, G. W., Weinstein, C. L., & Rabinowitz, F. E. (1982). Personal problem solving: A descriptive study of individual differences. *Journal of Counseling Psychology*, *29*(6), 580–590. https://doi.org/10.1037/0022-0167.29.6.580

Heppner, P. P., & Krauskopf, C. J. (1987). An information-processing approach to personal problem solving. *The Counseling Psychologist*, *15*(3), 371–447. https://doi.org/10.1177/0011000087153001

Heppner, P. P., & Petersen, C. H. (1982). The development and implications of a personal problem-solving inventory. *Journal of Counseling Psychology*, *29*(1), 66–75.

Hill, J. H., Houle, B. J., Merritt, S. M., & Stix, A. (2008). Applying abstraction to master complexity: The comparison of abstraction ability in computer science majors with students in other disciplines. Paper presented at: Proceedings of the 2nd International Workshop on The Role of Abstraction in Software Engineering (pp. 15–21). Leipzig, Germany: ACM.

Irgens, G. A., Dabholkar, S., Bain, C., Woods, P., Hall, K., Swanson, H., … Wilensky, U. (2020). Modeling and measuring high school students' computational thinking practices in science. *Journal of Science Education and Technology*, *29*(1), 137–161. https://doi.org/10.1007/s10956-020-09811-1

Jonassen, D. H. (2000). Toward a design theory of problem solving. *Educational Technology Research and Development*, *48*(4), 63–85. https://doi.org/10.1007/bf02300500

Kalelioğlu, F. (2015). A new way of teaching programming skills to K-12 students: Code.org. *Computers in Human Behavior*, *52*, 200–210.

Kalelioğlu, F., & Gülbahar, Y. (2014). The effects of teaching programming via scratch on problem solving skills: A discussion from learners' perspective. *Informatics in Education*, *13*(1), 33–50.

Keller, J. M. (2010). *Motivational design for learning and performance: The ARCS model approach*. New York, NY: Springer US.

Kim, B., Kim, T., & Kim, J. (2013). Paper-and-pencil programming strategy toward computational thinking for non-majors: Design your solution. *Journal of Educational Computing Research*, *49*(4), 437–459.

Kutu, H., & Sözbilir, M. (2011). Öğretim Materyalleri Motivasyon Anketinin Türkçeye Uyarlanması: Güvenirlik ve Geçerlik Çalışması. *Necatibey Eğitim Fakültesi Elektronik Fen Ve Matematik Eğitimi Dergisi*, *5*(1), 292–312.

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior*, *41*, 51–61.

Ngan, S.-C., & Law, K. M. Y. (2015). Exploratory network analysis of learning motivation factors in e-learning facilitated computer programming courses. *The Asia-Pacific Education Researcher*, *24*(4), 705–717. https://doi.org/10.1007/s40299-014-0223-0

Noh, J., & Lee, J. (2020). Effects of robotics programming on the computational thinking and creativity of elementary school students. *Educational Technology Research and Development*, *68*(1), 463–484.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York, NY: Basic Books.

Papert, S. (1993). *The children's machine: Rethinking school in the age of the computer*. New York, NY: Basic Books.

Pellas, N., & Peroutseas, E. (2016). Gaming in second life via Scratch4SL: Engaging high school students in programming courses. *Journal of Educational Computing Research*, *54*(1), 108–143.

Perkins, D. N., Hancock, C., Hobbs, R., Martin, F., & Simmons, R. (1986). Conditions of learning in novice programmers. *Journal of Educational Computing Research*, *2*(1), 37–55.

Polya, G. (1957). *How to solve it?* (2nd ed.). Princeton, NJ: Princeton University Press.

Polya, G. (1973). *How to solve it: A new aspect of mathematical method* (2nd ed.). Princeton, NJ: Princeton University Press.

Quade, D. (1967). Rank analysis of covariance. *Journal of the American Statistical Association*, *62*(320), 1187–1200.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., … Silverman, B. (2009). Scratch: Programming for all. *Communications of the ACM*, *52*(11), 60–67.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, *13*(2), 137–172.

Saez-Lopez, J. M., & Sevillano-Garcia, M. L. (2017). Sensors, programming and devices in art education sessions. One case in the context of primary education. *Cultura Y Educacion*, *29*(2), 350–384.

Saygıner, Ş. (2017). *Effects of block-based visual and text-based programming instruction on achievement, logical thinking and motivation*. Ankara, Turkey: Hacettepe University.

Scott, M. J., Counsell, S., Lauria, S., Swift, S., Tucker, A., Shepperd, M., & Ghinea, G. (2015). Enhancing practice and achievement in introductory programming with a robot Olympics. *IEEE Transactions on Education*, *58*(4), 249–254.

Soleimani, A., Herro, D., & Green, K. E. (2019). CyberPLAYce—A tangible, interactive learning tool fostering children's computational thinking through storytelling. *International Journal of Child-Computer Interaction*, *20*, 9–23.

Şahin, N., Şahin, N. H., & Heppner, P. P. (1993). Psychometric properties of the problem solving inventory in a group of Turkish university students. *Cognitive Therapy and Research*, *17*(4), 379–396.

Tüzün, H., Barab, S. A., & Thomas, M. K. (2019). Reconsidering the motivation of learners in educational computer game contexts. *Turkish Journal of Education*, *8*(2), 129–159. https://doi.org/10.19128/turje.546283

Voogt, J., Fisser, P., Good, J., Mishra, P., & Yadav, A. (2015). Computational thinking in compulsory education: Towards an agenda for research and practice. *Education and Information Technologies*, *20*(4), 715–728.

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology*, *25*(1), 127–147.

Wilkerson-Jerde, M. H. (2014). Construction, categorization, and consensus: Student generated computational artifacts as a context for disciplinary reflection. *Etr&D-Educational Technology Research and Development*, *62*(1), 99–121.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–35.

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *366*(1881), 3717–3725.

Wu, B., Hu, Y. L., Ruis, A. R., & Wang, M. H. (2019). Analysing computational thinking in collaborative programming: A quantitative ethnography approach. *Journal of Computer Assisted Learning*, *35*(3), 421–434.

Zhong, B., Zheng, J., & Zhan, Z. (2020). An exploration of combining virtual and physical robots in robotics education. *Interactive Learning Environments*. https://doi.org/10.1080/10494820.2020.1786409

How to cite this article: Çınar M, Tüzün H. Comparison of object-oriented and robot programming activities: The effects of programming modality on student achievement, abstraction, problem solving, and motivation. *J Comput Assist Learn*. 2021;37:370–386. https://doi.org/10.1111/jcal.12495

APPENDIX: SAMPLE ITEMS IN PROGRAMMING ACHIEVEMENT TEST

```
STEP 1: START
    STEP 2: counter = 0
    STEP 3: IF counter <10 THEN REPEAT {counter = counter+1; GO STEP 3;}
    STEP 4: END.
```

Which of the following is wrong for the statements above?

a. REPEAT is a loop process.
b. counter = 0 indicates the initial value of the loop.
c. counter = counter +1 indicates that the counter will increase by more than 2 times each turn.
d. counter <10 indicates the condition for the loop.
e. The operations in the "REPEAT" block are repeated 10 times.

Which of the expressions given below cause an infinite loop?

a. STEP 1: counter = 0;
   STEP 2: IF counter <100 THEN REPEAT {counter = counter+1; GO STEP 2}
b. STEP 1: counter = 0;
   STEP 2: IF counter<100 THEN REPEAT {counter = counter-1; GO STEP 2}
c. STEP 1: counter = 100;
   STEP 2: IF counter >0 THEN REPEAT {counter = counter-1; GO STEP 2}
d. STEP 1: counter = 0;
   STEP 2: IF counter <= 100 THEN REPEAT {counter = counter +5; GO STEP 2}
e. STEP 1: counter = 10;
   STEP 2: IF counter <0 THEN REPEAT {counter = counter-5; GO STEP 2}.

What is seen on the screen when the program given below is run?

```
a = 10;
b = 20;
a = a + b * 2 + 5;
WRITE a;
```

a. 35 b. 45 c. 55 d. 65 e. 150