Attending to structural programming features predicts differences in learning and motivation in a virtual robotics programming curriculum

Eben B. Witherspooon[1*], Christian D. Schunn[1], Ross M. Higashi[1], Robin Shoop[2]

[1]University of Pittsburgh, [2]The Robotics Institute, Carnegie Mellon University

Abstract

Educational robotics programs offer an engaging opportunity to potentially teach core computer science concepts and practices in K-12 classrooms. Here we test the effects of units with different programming content within a virtual robotics context on both learning gains and motivational changes in middle school (6th-8th grade) robotics classrooms. Significant learning gains were found overall, particularly for groups introduced to content involving program flow, the structural logic of program execution. Relative gains for these groups were particularly high on items that require the transfer of knowledge to dissimilar contexts. Reaching units that included program flow content was also associated with greater maintenance of programming interest when compared with other units. Therefore, our results suggest that explicit instruction in the structural logic of programming may develop deeper transferrable programming knowledge, and prevent declines in some motivational factors.

Keywords: robotics, programming, learning, motivation, computational thinking

Attending to structural programming features predicts differences in learning and motivation in a virtual robotics programming curriculum

Introduction

Computer science (CS) is quickly becoming an essential part of core K-12 STEM curricula, as schools attempt to prepare students for an expanding range of careers that require substantial CS knowledge. Despite a decline in participation in the early 2000s, enrollment in Advanced Placement CS classes are again on the rise, with 15% to 25% year-over-year increases in students taking the AP CS A exam every year from 2011 to 2016 (The College Board AP Data, 2016; Ericson & Guzdial, 2014). Policy initiatives like *CS for All* highlight the importance of preparing all students to apply computer science skills within a wide variety of careers (Smith, 2016). Therefore, research on K-12 CS education should examine features of learning environments that enable students to apply a conceptual understanding of CS to a variety of contexts, and grow STEM interest, identity, and engagement for a wider range of students.

Educational robotics can provide engaging CS experiences to diverse students (Rusk, Resnick, Berg, & Pezalla-Granlund, 2008). These experiences also support learning abstract computer programming by using concrete external representations (Papert & Harel, 1991). Out-of-school robotics activities like summer camps and club teams can also expand interest in STEM careers (Hendricks, Alemdar, & Ogletree, 2012; Petre & Price, 2004). Overall, introducing robotics curriculum into general education classrooms may help disseminate programming to a broader population beyond those who self-select into robotics electives and clubs.

However, little is known about whether programming knowledge gained from these activities is carried beyond the context of robotics. Further, relatively few empirical studies examine whether educational robotics experiences can produce gains in both motivation and programming knowledge (i.e., be fun and rigorous). In this study, we investigate what aspects of a robotics programming curriculum may lead to transferrable knowledge that will prepare students for a range of future CS-relevant careers. Further, we are interested in determining if there is a relationship between curricular features and shifts in motivational factors, which may also be relevant to persisting in CS learning experiences; namely, the development of higher levels of students' programming interest, programming identity, and their beliefs in their ability to be successful in CS.

Teaching generalizable programming skills

Computational thinking, a term that has gained a great deal of attention in K-12 CS education over the past decade, is broadly defined as "an approach to solving problems in a way that can be solved by a computer…a problem solving methodology that can be transferred and applied across subjects" (Barr & Stephenson, 2011). Consensus has not yet been reached over the specific concepts that make up computational thinking. However, there is consensus that it should involve very general programming practices such as algorithmic thinking and design processes, which, as high-level practices, could be generalizable across contexts (Grover & Pea, 2013; Wing, 2006). Various organizations have suggested that more specific concepts and skills of computer science such as iteration and task decomposition are fundamental in the development of computational thinking; in this paper, we focus on particular fundamental programming concepts and skills that are generally endorsed as important (see *AP Computer*

Science Principles, 2016, *Interim CSTA K-12 Computer Science Standards*, 2016; Bienkowski, Snow, Rutstein, & Grover, 2015).

The K-12 Computer Science Framework (2016) suggests that by 8th grade, CS students should be learning to develop modular, generalizable algorithms that can produce a range of outputs based on different inputs, and incorporate more complex control structures (i.e., conditions nested within loops). These build upon the basic concepts developed in elementary school such as identifying "everyday" algorithms (i.e., steps for making a sandwich) and prepare students for more advanced CS content in high school (i.e., recursion, arrays). In addition to specific grade-level concepts, the Frameworks also emphasize core computational practices across grade bands, such as decomposition of complex problems into smaller sub-goals, using abstraction to functionalize their solutions, and troubleshooting programs to identify errors in logic.

Aspects of general programming can be taught within a wide range of learning environments, including robotics (Lye & Koh, 2014). Since the 1980s, cognitive scientists have theorized that learning computer programming could develop students' general problem solving skills, yet few studies have been able to demonstrate empirical support for these claims (Dalbey & Linn, 1985; Kurland, Pea, Clement, & Mawby, 1986; Pea & Kurland, 1984). Transfer of knowledge and skills to other contexts is believed to be more likely if students understand the underlying structural features of programming logic (i.e., which commands in a loop will be repeated, when/if a conditional statement will execute), rather than more superficial rote repetition of programming language features and syntax (i.e., when a semi-colon is needed to indicate the end of a line; where to place brackets around loops; Palumbo, 1990; Salomon & Perkins, 1987). For example, early studies involving the BASIC programming language showed that students were able to make more progress towards developing problem solving skills that could be applied to learning other programming languages when their instructor focused explicitly on program design (e.g., task decomposition, debugging processes), rather than language-specific features (Linn & Dalbey, 1985). Explicit attention to the structural logic of a program (i.e., control structures and program flow) could organize programming knowledge into schemas, which make it easier to recognize similarities in novel tasks and select an appropriate solution.

Recent advances in educational technologies have revitalized research on the potential generalizability of programming knowledge and skills. Visual programming languages like Scratch are thought to reduce the cognitive load for novice programmers by reducing superficial syntactic errors, freeing learners to focus on the structural logic of code (Kelleher & Pausch, 2005; Robins, Rountree, & Rountree, 2010). For example, rather than requiring specific punctuation to denote a looping structure, visual languages embed these features within graphical blocks, using colors and other cues to easily represent these functions (see Figure 1). Particularly important within the K-12 context, these scaffolds could make programming more approachable for students with relatively little prior programming experience (Repenning, Webb, & Ioannidou, 2010).

(Figure 1 here)

To develop these transferable knowledge structures, some research suggests that providing a variety of examples could help novices to induce more general rules that develops an adaptive rather than routine expertise (Barnett & Koslowski, 2002; Hatano & Inagaki, 1986).

Multiple examples may be particularly effective for transfer when there are variations in the superficial features of a problem, but no change in the underlying structural features that are functionally related to the task outcome (Gick & Holyoak, 1983). Given the high level of perceptual features in robotics contexts (i.e., narrative task features, mechanical robotics parts) that might limit transfer, programming transfer from robotics might be especially dependent upon dynamic programming tasks that change superficial features of the problem (i.e., layout of a maze, arrangement of blocks to be moved) before each attempt, in order to develop students' ability to recognize the underlying solution requirements.

In sum, advances in educational technology and an understanding of its pedagogical implications for programming could allow curriculum designers to develop lessons that better scaffold problem solving expertise to support the generation of transferrable knowledge. In this study, we examine students' progress through a virtual robotics curriculum that introduces dynamic problem solving tasks using a visual programming language, allowing them to focus on the logic of programming structures that must be responsive to dynamically shifting task features.

Motivating interest and maintaining competency beliefs

While developing a conceptual understanding of CS that can be applied within a variety of contexts is important, students' continued participation in CS-related careers is also likely to require the development of motivational factors such as programming interest, identity as a programmer, and positive beliefs about programming abilities. Robotics programs have been heavily studied in out-of-school learning environments, which generally focus on extending students' existing STEM interests. By contrast, in-school robotics courses are less likely to have self-selected populations of students with high STEM interest, and therefore may struggle to keep all students equally engaged in programming. Studies at the middle, high school, and undergraduate level suggest that students' interest in a topic can predict how they choose to spend their free time, what courses they select, and what major they pursue (Harackiewicz & Hulleman, 2010). However, early CS experiences that do not seem relevant to prior interests, or reinforce negative stereotypes about CS careers, could deter students from choosing a CS major in the future; these negative effects may be particularly strong for women (Beyer, 2014; Carter, 2006). For example, a study of middle and high school robotics teams showed that girls' declining participation in programming activities was explained by a concurrent decline in interest in programming (Witherspoon, Schunn, Higashi, & Baehr, 2016). To encourage long-term participation in CS, K-12 general education robotics classes should aim to trigger situational interest in novice programming students, while also deepening individual interest by developing confidence in programming ability.

Providing opportunities for students to develop their identity as a programmer may also be an important factor in encouraging continued participation in CS. Activities that provide opportunities for "productive disciplinary engagement", where students work on authentic problem-solving tasks that require them to use the practices of a field, could develop students' identity in ways that predict continued participation (Collins, 2006; Engle, 2006; Engle & Conant, 2002). In a longitudinal study of persistence in science, high school students who had opportunities to participate in communities of practice (Collins, 2006; Lave & Wenger, 1991) were more likely to continue to consider a career in science (Aschbacher, Li, & Roth, 2010).

Levels of engagement and persistence in CS may also be closely associated with students' perceptions of both the difficulty of programming as a discipline, and their own abilities (Bandura, 1989). Further, beliefs about ability in STEM domains have been shown to be more predictive of performance than both prior experience or outcome expectations; gifted women may be particularly prone to under-confidence in those traditionally male-dominated fields (Pajares, 1996; Zeldin & Pajares, 2000). However, exposing students to advanced content can sometimes have the unfortunate side-effect of reducing student confidence levels because they come to learn what competence in the domain actually involves; ironic tradeoffs between actual ability development and perceived abilities can exist. Allowing students to experience "small wins" at each step of the programming process could cause the perception of each overall problem solving task to be less daunting, appear less demanding, and raise students' perceived ability level (Weick, 1984).

Overall, this suggests that interest and other motivational factors may play a significant role in K-12 students' persistence in CS activities. In addition to learning transferrable CS skills, motivating all students in K-12 to continue to engage with a variety of CS learning opportunities will be important for preparing students for the growing variety of CS-relevant careers. Therefore, to determine the effectiveness of this curriculum in achieving these goals, in this study we also measure participants' development of programming interest, identity, and competency beliefs.

Research questions

Our main research questions are twofold: First, is reaching more conceptually rich units associated with larger overall learning gains in programming? Further, are learning gains found within contextually distant transfer items? If so, this will help strengthen our claims that it is the particular character of the tasks in later units that are driving knowledge transfer, and not simply the repetition of more foundational concepts. Second, we aim to determine if participation in the curriculum is associated with any significant changes (positive or negative) in motivational characteristics, particularly interest, identity, and competency beliefs. Effects on motivation is also an important feature of a K-12 CS curriculum that aims to be effective in non-elective learning environments (i.e., broaden participation at later levels).

Methods

Sample

The sample for this study consisted of *N*=136 6th and 8th grade students within seven different robotics classes taught by three teachers across two schools in southwestern Pennsylvania. The sample was predominantly White (78%) and relatively equally divided by sex (Male = 52%, Female = 48%). Each participating teacher taught 2-3 sections of robotics using the virtual curriculum and elected to be part of the study. Teachers were relatively comparable in terms of level of training and teaching experience, all having received at least a four year Bachelor's of Science degree in Technology Education, as well as attending two professional development sessions about the curriculum prior to implementation. All human subjects research received Institutional Review Board (IRB) approval prior to the commencement of the study.

Participants in the study completed different levels of the curriculum; some finished only the Basic Movement unit (n=39, M_{age}=11.42, SD=.68), others also completed the Sensors unit (n=40, M_{age}=11.26, SD=.44), and some completed all three units; Basic Movement, Sensors and the Program Flow unit (n=57, M_{age}=13.11, SD=.37, see further description of these units in the Materials section). On average, students who completed the Program Flow unit were shown to be significantly older than both the Basic Movement, $t(92)$=15.51, p<.001, d=3.26) and the Sensors groups, $t(89)$=21.61, p<.001, d=4.66). However, on a single item measuring robotics experience ("This is my first robotics class"), proportion of first time robotics students in the Program Flow group (12%) and the Basic Movement group (21%) showed no significant differences, *t*(94)=1.1, *p*=.28, *d*=0.23, with only the Sensors group (70%) showing significantly more first-timers than both the Basic Movement group, *t*(77)=5.02, *p*<.001, *d*=1.13, and Program Flow group, *t*(95)=7.15, *p*<.001, *d*=1.48.

Materials

Robotics programming curriculum. The virtual robotics curriculum used here, developed by Carnegie Mellon University and Robomatter, involves a sequence of lessons in robotics programming utilizing a visual programming language, ROBOTC Graphical. Earlier versions of a similar programming curriculum have been reported on in previous studies (see Authors et al., in press); however, for this study, each unit was shortened by removing sections that did not contain conceptual programming content, with the aim of allowing more students to reach the rich programming content in later units. Here we provide a brief overview of this revised curriculum, emphasizing the elements which were created to support efficient learning and transfer: procedural scaffolds (worked examples, guided videos), dynamic mini-challenges, visual programming language, and Robot Virtual Worlds (RVW), a virtual programming environment[1].

The design of these curricular materials reflects a constructionist approach to instruction in which learners' use worked examples, scaffolding and reflection to build increasingly complex programmed solutions and construct an understanding of the requisite programming principles (Papert, 1980). To provide a shared context, students are provided with a short introductory video to frame the activity. These videos are learner-paced and present visual

[1] Supplemental resources with additional details and examples of this curriculum are available in the online version of this article.

support together with a conversational narrative around the key concepts, to reduce extraneous processing and foster generative processing (Mayer, 2008). Partial scaffolding (Puntambekar & Hubscher, 2005) is introduced by way of questions to check students understanding, step-by-step instruction on a conceptually related robotics programming activity, and a brief post activity quiz to assess understanding, followed by the open-ended application of these skills within a game-like challenge in the virtual programming environment, allowing students to apply their knowledge more independently.

The curriculum consisted of three instructional units: Basic Movement, Sensors, and Program Flow. For each unit, students engaged in the sequence of guided videos and mini-lessons introducing the key concepts, and a final open-ended challenge. Each challenge required a programmed solution that would vary superficially, but structurally require the inclusion of the key concepts targeted in each unit. Important to note is that early tasks were more likely to be *static*; that is, the task did not vary and could be solved using a relatively rote set of common programmed commands. However, as students progressed, they encountered tasks which required the use of robotics sensors and programming logic in ways that were *dynamic*; that is, surface level aspects of the task would change in ways that required superficial adjustments to the code, while the structural and conceptual features of the task remained the same (see Table 1).

(Table 1 here)

Students interacted with the curriculum through Robot Virtual Worlds, a simulated 3D game-like virtual environment (see Figure 2) designed to emphasize the programming aspects of robotics, while maintaining student interest and engagement. Students can iteratively test modular programmed solutions with simulated VEX IQ robots in a three-dimensional virtual platform. It is important to note here that while the curriculum can be completed entirely within the virtual environment, it is designed to replicate existing physical robotics hardware, and therefore the capability to download and test programs on physical VEX IQ robots was available to teachers with access to them. While some teachers may have taken advantage of this capability, we are confident from observations and discussions with teachers that these did not compose a majority of instruction.

Finally, these solutions are “remixed and reused” (Brennan & Resnick, 2012) to complete more complex virtual challenges, in which learners must apply their previous programming knowledge to problem solving tasks that foreground computational thinking principles like abstraction, decomposition, and systems thinking.

(Figure 2 here)

To solve these challenges, students used a programming language called ROBOTC Graphical to develop programmed solutions. ROBOTC Graphical has a visual programming language interface, intended to allow students to focus on the broader logic of programming while deemphasizing the particular syntactic requirements of more traditional programming languages (see Figure 3).

(Figure 3 here)

By representing robotics challenges in a virtual environment, this curriculum offers affordances over physical robotics programs by reducing the potential frustration and distractors of mechanical error, enabling students to focus on higher-level computational principles of programming. While physical robots may have some advantages, an earlier study by Liu, Newsom, Schunn, and Shoop (2013) found that students using an earlier version of this technology achieved learning gains in programming content equivalent to students using physical robots, but in significantly less time. Simulating robot movement reflects an authentic engineering practice (see Michel, 2004), and virtual robots are also less expensive than physical ones, allowing the benefits of the curriculum to reach a broader population where the costs of physical robotics curricula can be prohibitive.

Programming assessment. Students completed a pre-and post-assessment of robotics programming and computational thinking, consisting of programming items ranging from concepts within the relatively narrow context of the virtual robotics programming language, to more general language-agnostic programming questions, as well as generalizable computational thinking items in a non-robotics context (see Appendix A for sample items). These items were developed to target three core programming concepts common across a range of accepted frameworks of programming and computational thinking; *sequences*, *conditions* and *iteration* (see *AP Computer Science Principles*, 2016, *Interim CSTA K-12 Computer Science Standards*, 2016; Bienkowski et al., 2015).

Items were created to assess each of these concepts, at varying levels of functional distance from the learning context (Barnett & Ceci, 2002). Functional distance was manipulated in order to create three varying levels of transfer within the assessment; Robot Programming, General Programming and Computational Thinking (see Table 2).These items assess both knowledge of programming concepts (i.e., sequences, conditions) as well as require the learner to engage in computational practices (i.e., testing and refining computational artifacts; see *K–12 Computer Science Framework*, 2016). Armor's θ[2] for the assessment overall was 0.84. By section, the robot programming language questions had a θ of 0.64; general programming items had a θ of 0.68, and the computational thinking items had a θ of 0.65. Such theta values are common for relatively short assessments that are intended to cover a range of concepts.

(Table 2 here)

Attitudes towards programming. Additionally, students also completed a short survey prior to the pre and post-test exams, with 12 items that asked students about their interest, competency beliefs, and development of identity in computer programming (see Appendix B for a complete list of these survey items). Interest was gauged through four items (e.g., "I wonder about how computer programs work", Cronbach's α=.79), rated along a four point Likert scale (e.g. "Never" to "Every Day"). Competency beliefs were gauged through four items (e.g., "I am sure I could do advanced work in programming", α=.83) rated along a six point Likert scale (e.g. "Strongly Disagree" to "Strongly Agree"). Finally, four items gauged level of identity as a programmer (e.g., "My family thinks of me as a programming person", α=.85), rated along a four point Likert scale (e.g. NO! to YES!).

Measure characteristics. Overall, the three subcategories of skills were moderately correlated with each other (see Table 3). Further, post correlations along skills categories

[2] Armor's θ is similar to Cronbach's α, but is more appropriate for binary data (item correct vs. item incorrect).

followed the predicted pattern by level of transfer: robot programming and general programming were more highly correlated (*r* =.51) than robot programming and computational thinking (*r* =.43). Attitudinal measures were strongly correlated with each other, but not so high as to be redundant measures. Skill measures were relatively independent of attitude measures, but computational thinking items were significantly correlated with interest items at pre (*r* =.24), and with competency beliefs at post (*r* =.31).

(Table 3 here)

Procedure and analysis

Students within each classroom were randomly assigned one of two analogous forms of the assessment (Form A or Form B) as a pre-test prior to starting the robotics curriculum, and later took the alternate form of the assessment as a post-test, in order to mitigate test-retest effects. Pre-test scores showed no significant differences between Form A (*n*=64, *M*=10.3, *SD*=4.5) and Form B (*n*=66, *M*=9.64, *SD*=4.0), *t*(128)=0.88, *p*=.38, *d*=0.15. That is, students from a relatively similar population preformed equally well on either form of the assessment prior to instruction, suggesting these two forms are relatively analogous measures. Therefore, results from both forms were collapsed into pre-test and post-test scores in the reported analyses. The motivational survey was identical pre and post, since test-retest effects are not typically a concern for such surveys.

Assessment results were modeled using ANCOVA to determine if there were significant differences in post-test scores between the Basic Movement, Sensors and Program Flow student groups, while controlling for pre-test scores. Normality and homogeneity of variances for the three groups were tested and all assumptions for the analyses were met. Levene's tests for homogeneity of variance across the three groups showed no significant differences across the three groups on pre-test scores for any of the sub-categories of programming assessment items or motivational items. Next, a second series of ANCOVAs were conducted to test for differences between these groups, but this time separately for each section of the assessment (robotics programming, general programming, and computational thinking). This analysis was performed to determine if students' experience with particular units was associated with different levels of gains in certain categories of programming assessment items.

Finally, survey scales were analyzed to determine if participation in each unit was significantly associated with changes in the level of student reported programming interest, identity formation as a programmer, and competency beliefs in their ability to program. First, gains scores were generated as the mean differences between the pre- and post-survey responses for each section of the survey. Next, ANCOVA were conducted using these gain scores to determine if there were significant differences in gains in interest, identity and competency beliefs from pre- to post-survey between the groups of students who reached Basic Movement, Sensors, and Program Flow.

Results

Assessment Results

Overall results showed on average, significant increases of about 8 percentage points, $t(101)=5.44$, $p<.001$, $d=0.48$, from pre- to post for all groups on assessment items, with larger mean gains found in the Sensors and Program Flow groups. For the initial analysis of the programming assessment items, an ANCOVA showed that overall differences between students in the Basic Movement, Sensors, and Program Flow groups were significant when controlling for pre-test score, $F(2,98)=10.14$, $p<.001$, $\eta_p^2=0.17$. Post-hoc analyses showed that while gains for the Basic Movement group were significantly lower than both the Sensors and Program Flow groups at the $p<.001$ level, no significant overall differences were found between the Sensors and Program Flow groups, $p=.73$ (see Figure 4). Therefore, the predicted pattern was found, with higher gains for students who progressed beyond the Basic Movement unit. Unexpectedly, however, students who reached the Sensors unit also demonstrated overall gains on the programming assessment that were comparable to the gains found for students who reached Program Flow.

(Figure 4 here)

We next conducted a series of ANCOVA to see if there were significant differences between each group in post-test scores within each section of the assessment, controlling for the pre-test scores of those sections (see Figure 5).

Robot programming. The robot programming items showed significant differences between the three different student groups, $F(2,96)=8.24$, $p<.001$, $\eta_p^2=0.15$. Post-hoc analyses showed significant differences between the Basic Movement group and both the Sensors ($p<.001$) and Program Flow ($p<.001$) groups, but no significant differences between the Sensors and Program Flow groups ($p=.53$). Students in the Basic Movement group, $t(31)=0.37$, $p=.71$, $d=0.10$, and the Program Flow group, $t(43)=1.32$, $p=.19$, $d=0.20$, showed no significant pre- to post-test gains on the robotics programming items, while significant gains were found for students in the Sensors group, $t(23)=2.61$, $p<.05$, $d=0.50$. In sum, while only students who reached Sensors showed significant pre- to post-test gains on robotics programming items, students who reached either Sensors or Program flow showed more growth on programming items within a robotics context than students who only reached Basic Movement[3].

General programming. The general programming items showed significant differences between the three student groups, $F(2,96)=11.62$, $p<.001$, $\eta_p^2=0.19$. Differences were significant between the Basic Movement group and both the Sensors ($p<.001$) and Program Flow ($p<.001$) group, but there were no significant differences found between the Sensors and Program Flow groups ($p=.85$). Results from the General Programming section of the assessment also show no significant pre-post gains for the Basic Movement group $t(31)=0.00$, $p=1.0$, $d=0.00$ but significant gains for both the Sensors, $t(23)=3.76$, $p<.01$, $d=0.86$, and Program Flow $t(43)=3.78$, $p<.001$, $d=0.59$, groups. In sum, students who reached either Sensors or Program Flow showed gains on items in a general programming context relative to students who only reached Basic Movement, and these gains were slightly larger than the gains showed with the robotics programming items, suggesting relatively robust transfer at this level.

Computational thinking. Computational thinking items showed significant differences between the three student groups, $F(2,94)=3.82$, $p<.05$, $\eta_p^2=0.08$. The differences were not significant between the Basic Movement group and the Sensors group ($p=.16$), but significant differences were found between the Basic Movement and the Program Flow group ($p<.01$). No significant differences were found between the Sensors and Program Flow groups ($p=.27$). Important to note here is that while on the other sections of the assessment the Sensors group showed a significantly higher gain over the Basic Movement group, in the Computation Thinking section of the assessment, these differences over Basic Movement only continue to be

[3] Effect sizes are of d=.20 are considered “small”, d=.50 “medium” and d=.80 “large”.

significant for the group that completed the Program Flow section. Results from the Computational Thinking section of the assessment also show no significant gains for the Basic Movement group $t(23)=1.50$, $p=.14$, $d=0.40$, or the Sensors, $t(23)=1.22$, $p=.24$, $d=0.31$, groups, but significant gains were found for the Program Flow group, $t(41)=3.00$, $p<.01$, $d=0.45$. Therefore, the results suggest that particularly on items in the most distant computational thinking context, there is a unique advantage for students who reach the Program Flow unit.

(Figure 5 here)

Motivation Change Results

To determine if different motivational effects were associated with each of the classroom conditions, simple ANOVAs were conducted using the pre and post scores in Interest, Identity, or Competency Beliefs (see Figure 6). All three conditions showed pre-post declines, but to different degrees.

Interest. For Interest, significant differences were found in the level of interest changes between the three groups, $F(2,99)=6.06$, $p <.01$, $\eta_p^2=0.11$. Between group differences were significant for the Program Flow and Basic Movement groups ($p<.001$) and for the Sensors and Basic Movement groups ($p<.05$), but there were no significant differences in Interest changes found between the Sensors and Program Flow groups. Therefore, results would suggest that in addition to the content gains that were shown above, reaching both the Sensors, and particularly the Program Flow unit, is also associated with relatively smaller declines in Interest in programming. A significant decline in Interest from pre- to post-survey was found for students in the Basic Movement group, $t(31)=4.56$, $p<.001$, $d=0.61$. Small, marginally statistical declines in

Interest were found for the Sensors group, *t*(24)=1.91, *p*<.1, *d*=0.36 and there was no significant decline in Interest for the Program Flow group, *t*(44)=1.57, *p*=.12, *d*=0.13.

Identity. Identity showed no significant differences between the three student groups, *F*(2,99)=0.99, *p* =.37, η_p^2=0.01. These results suggest that there was little impact on Identity for the six to nine-week course, despite the different levels of content that were reached. Of the three groups, only the Sensors groups showed small but marginally significant pre-post differences in Identity, *t*(24)=1.79, *p*<.1, *d*=0.31, with no significant changes found in either the Basic Movement group, *t*(31)=0.51, *p*=.61, *d*=0.08, or the Program Flow group, *t*(44)=0.00, *p*=1.0, *d*=0.00.

Competency Beliefs. Finally, for Competency Beliefs, there were no significant differences between the Basic Movement and Program Flow groups; however post-hoc analyses showed that significant differences were found between Sensors and Basic Movement (*p*<.05). Significant declines in Competency Beliefs were found for both the Basic Movement group, *t*(31)=2.81, *p*<.01, *d*=0.40, and for the Program Flow group, *t*(44)=2.46, *p*<.05, *d*=0.27. There were no significant pre-post differences in Competency Beliefs for the Sensors group, *t*(24)=0.0, *p*=1.0, *d*=0.00. In sum, the Competency Belief declines were largest in the Basic Movement group, smaller yet significant in the Program Flow group, and non-existent in the Sensors group.

(Figure 6 here)

Differences by Gender

Because the literature suggests that motivational factors can be strong predictors of performance for women, particularly in traditionally male-dominated fields like CS (Pajares, 1996; Zeldin & Pajares, 2000), we conducted a follow-up ANOVA to determine if there were significant differences by gender on the knowledge and motivational assessments, across all units. For motivational factors, we found no significant differences between boys and girls at post, controlling for pre, on interest, $F(1,99)=0.69$, $p=.41$, $\eta_p^2=0.01$, identity, $F(1,99)=1.10$, $p=.30$, $\eta_p^2=0.01$, or competency beliefs, $F(1,99)=0.57$, $p=.45$, $\eta_p^2=0.01$. However, when looking at the three components of the knowledge assessment, girls showed significantly higher gains on each section of the assessment, with particularly large advantages for girls in Robotics Programming, $F(1,97)=15.28$, $p<.001$, $\eta_p^2=0.15$. Smaller significant differences were also found between boys and girls on General Programming, $F(1,97)=6.65$, $p<.05$, $\eta_p^2=0.06$, and in Computational Thinking, $F(1,97)=5.45$, $p<.05$, $\eta_p^2=0.05$. This translates to girls performing about a 14 percentage points higher than males on the Robotics Programming items, about 12 percentage points higher on the General programming items, and about 9 percentage points higher on the Computational Thinking items, when controlling for their pre-test scores (see Figure 7).

(Figure 7 here)

Therefore, while girls and boys showed no significant differences in their motivations about programming, girls showed larger gains in their programming knowledge than boys who participated in the curriculum; possible explanations of these and other findings are taken up in the next section.

General Discussion

With this study, we contribute to the literature on development of computational thinking and of knowledge transfer in a programming context by evaluating whether students' involvement in a virtual robotics curriculum is associated with an increase in their generalizable programming knowledge and computational thinking. Thus, the very concrete context of robotics can be used to successfully teach relatively abstract computational concepts and skills.

Further, we found that students' experiences with more conceptually rich units, which provide multiple opportunities to engage in structurally similar tasks, are associated with larger gains on programming assessment items presented in increasingly dissimilar contexts. This finding supports the initial design decision to shorten the curriculum, in order to allow more students to reach these later units. Importantly, when looking at specific groups of assessment items, groups reaching those units performed better on more generalizable assessment items on relatively dissimilar tasks, suggesting that the particular features of those units better facilitate transfer for programming knowledge.

Another primary aim of this study was to determine if participation in a virtual robotics curriculum was associated with changes in certain motivational characteristics that might predict continued participation in computer science. Specifically, we examined whether students' participation in each curriculum was associated with differential gains in programming interest, identity as a programmer, and beliefs in their programming abilities. Overall, results showed declines in these motivation characteristics; however, more nuanced patterns emerged when observing these characteristics by group. Most importantly, programming interest did not decline for those students who had completed the Program Flow unit.

Finally, our study shows that despite showing no pre-test differences on both knowledge and motivational factors, girls who participated in these programs showed relatively higher performance on the post-test than boys. This supports prior work suggesting that girls typically outperform boys on a variety of subject throughout middle school, despite not necessarily being any more confident in their abilities (Pomerantz, Altermatt, & Saxon, 2002).

Limitations

One potential concern regarding the current study is the lack of random assignment to conditions. As a result, the inherently correlation design does not directly address causality, and we cannot be 100% certain that it was in fact differential exposure to the curriculum that caused the observed changes. For example, it is possible that other unobserved factors, such the particular instructional approaches taken by the teacher or the use of physical robotics to supplement the virtual curriculum, may have varied. However, the teachers were equivalent in training, and the classrooms making less progress were given a curriculum which have many more basic activities, providing a plausible explanation for the differential progress.

Second, there is the potential age confound with our three groups. The Program Flow group was found somewhat older than the other two; therefore, it is possible that latent, unmeasured factors (e.g. mathematics learning) that occur during these grades could also be different for this group, accounting for some of the differences in gains found. However, we were able to show that students were comparable in their robotics experience and on their pre-test scores across all groups, and age differences would also not account for the differences found between the Basic Movement and Sensors group, which did not significantly differ in age. Therefore, while age may explain some of the overall effect on assessment score, it is unlikely to account for the differential patterns across the range of dissimilar assessment items.

Implications

Results found here support earlier evaluation of a similar curriculum (see Authors, in press), showing that groups of students who were able to reach more conceptually rich units of the curriculum (i.e., Sensors, Program Flow) were more likely to show gains across all assessment items. Additionally, however, our current results show that groups reaching these units demonstrated particularly large gains specifically on the assessment items that were relatively dissimilar from the original learning context. This is consistent with other studies in the transfer literature that suggest students might acquire more generalizable knowledge by engaging in tasks that vary superficial features while maintaining structural features, as was most characteristic of tasks found in the Sensors and Program Flow units.

An interesting finding from this study is that on the most dissimilar computational thinking items, only groups reaching the Program Flow unit showed significant gains above those found in the Basic Movement group. One interpretation could be that the particular content taught during Program Flow may provide students with additional opportunities to engage with the logical structure of programming above and beyond Sensors. For example, the key programming concept of iteration that is first introduced in this unit requires students to understand how a looping (or "repeat") structure can disrupt the typically sequential flow of a program execution, and that the "repeat" operates on all commands within that loop. In the visual programming language used with this curriculum, these loop structures are visually supported through the use of color-coded virtual "brackets" which allow students to drag and drop the commands to be repeated inside this loop structure. Therefore, through attending to the underlying logic of the program execution in this unit, and using the features of the programming language to emphasize program structure, students who engage in this unit may develop more schematized programming knowledge that is more likely to be instantiated when solving dissimilar problems (Reed, 1993).

Overall, results from the motivational analyses show declines across each of our motivational measures. As we mentioned before, one of the known challenges in implementing robotics courses in K-12 environments, unlike in informal robotics programs, is that students have not self-selected into these learning environments. Prior motivational studies of broad cross-sections of students (i.e., not those specifically focused on STEM) have shown significant declines in valuation of STEM subjects in middle school (Wigfield & Eccles, 2000). It is possible, therefore, that our negative results merely show that our relatively short intervention was unable to reverse this trend. Studies showing growth in interest development and STEM identity development often find significant effects only after years of intervention (Aschbacher et al., 2010; Hidi & Renninger, 2006). However, it is encouraging that while the overall direction of these gains remained negative, in some groups we were able to observe a significant effect that appears to maintain interest in programming even after a short amount of time. Theories of interest development suggest that if maintained over time, a triggered situational interest can evolve into a self-generated individual interest that leads to seeking additional opportunities to participate in similar tasks (Hidi & Renninger, 2006). Therefore, future studies should continue to investigate the potential for interventions that both prevent the typical decline in STEM during middle school, as well as develop more powerful interventions that may reverse the effect.

The observed patterns in competency beliefs can be interpreted in relation to the relative unfamiliarity of the requirements of programming for middle schoolers. Particularly in middle school, students who are relatively unfamiliar with programming may experience a Dunning-Kruger effect, where their assessment of their abilities are overly high, in part because their lack of conceptual understanding of what programming entails causes their metacognitive self-assessments to be inaccurate prior to engaging in the curriculum (Kruger & Dunning, 1999). Further, superficial levels of understanding can help these inaccurate self-assessments to persist, particularly if the learning environment prioritizes task completion rather than conceptual understanding. For example, in simpler units of a robotics programming curriculum, student may be able to complete the robot tasks through guess-and-check methods, without much understanding of the steps that were critical to accomplishing that task. If the curriculum allows students to continually reconfirm their high self-assessment through automated feedback from the virtual environment that affirms their progress, this mismatch between competency beliefs and actual ability could grow, creating more drastic declines self-assessed competency when later more difficult tasks are no longer solvable using the same novice methods. Therefore, to avoid this effect in measurement, it would be interesting in future studies to implement a retrospective pre-test method for measuring competency beliefs, where students would reflect at post-test about the amount of knowledge they thought they had beginning, compared to the amount that they had after completing the curriculum. Nonetheless, even if the effect is attributable to an improved understanding of what it means to be competent, future work should seek to leave students feeling more confident in their abilities, perhaps through a growth mindset intervention (Blackwell, Trzesniewski, & Dweck, 2007; Miele & Molden, 2010).

Conclusion

Educational robotics may provide an opportunity to develop generalizable programming knowledge and skills, as well as maintaining interest in programming through middle school. This is particularly important as the number of careers related to computer science continues to grow, and as computer science knowledge and skills become more broadly applicable in a variety of non-computing fields. Teaching CS to all students is likely to require the inclusion of CS instruction as a non-elective, general education class; however, less is known about potential difficulties in implementing educational robotics in these more traditional K-12 classrooms. This study suggests that a virtual robotics curriculum that utilizes a visual programming language, and offers dynamic programming tasks that vary superficial task features while keeping structural features constant, could help develop knowledge structures that can be more readily applied in relatively dissimilar contexts. Future studies utilizing a more rigorous quasi-experimental design to control for a variety of potential differences between the conditions, as well as complementary qualitative analyses, could provide additional insight into how particular instructional decisions may interact with the curriculum to prevent or enhance the effects on student learning and motivation.

References

AP Computer Science Principles. (2016). New York, NY.

Aschbacher, P. R., Li, E., & Roth, E. J. (2010). Is science me? High school students' identities, participation and aspirations in science, engineering, and medicine. *Journal of Research in Science Teaching*, *47*(5), 564–582. http://doi.org/10.1002/tea.20353

Bandura, A. (1989). Social Cognitive Theory. In R. Vasta (Ed.), *Annals of child development* (Vol. 6, pp. 1–60). Greenwich, CT: JAI Press. http://doi.org/10.1146/annurev.psych.52.1.1

Barnett, S. M., & Ceci, S. J. (2002). When and where do we apply what we learn? A taxonomy for far transfer. *Psychological Bulletin*, *128*(4), 612–637. http://doi.org/10.1037/0033-2909.128.4.612

Barnett, S. M., & Koslowski, B. (2002). *Adaptive expertise: Effects of type of experience and the level of theoretical understanding it generates*. *Thinking & Reasoning* (Vol. 8). http://doi.org/10.1080/13546780244000088

Barr, V., & Stephenson, C. (2011). Bringing Computational Thinking to K-12: What is Involved and What is the Role of the Computer Science Education Community ? *ACM Inroads*, *2*(1), 48–54. http://doi.org/10.1145/1929887.1929905

Beyer, S. (2014). Why are women underrepresented in Computer Science? Gender differences in stereotypes, self-efficacy, values, and interests and predictors of future CS course-taking and grades. *Computer Science Education*, *24*(2–3), 153–192. http://doi.org/10.1080/08993408.2014.963363

Bienkowski, M., Snow, E., Rutstein, D., & Grover, S. (2015). *Assessment design patterns for computational thinking practices in secondary computer science : A first look (SRI* *technincal report)*. Menlo Park, CA.

Blackwell, K. L., Trzesniewski, K. H., & Dweck, C. S. (2007). Implicit Theories of Intelligence Predict Achievement Across an AdolescentTransition: A Longitudinal Study and an Intervention in Child. *Child Development*, *78*(1), 246–263. http://doi.org/10.1111/j.1467-8624.2007.00995.x

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. *Annual American Educational Research Association Meeting, Vancouver, BC, Canada*, 1–25. Retrieved from http://web.media.mit.edu/~kbrennan/files/Brennan_Resnick_AERA2012_CT.pdf

Carter, L. (2006). Why students with an apparent aptitude for computer science don't choose to major in computer science. *ACM SIGCSE Bulletin*, *38*(1), 27. http://doi.org/10.1145/1124706.1121352

Collins, A. (2006). Cognitive Apprenticeship. In R. K. Sawyer (Ed.), *The Cambridge Handbook of the Learning Sciences* (pp. 47–60). Cambridge, UK: Cambridge University Press.

Dalbey, J., & Linn, M. C. (1985). The demands and requirements of computer programming: A review of the literature. *Journal of Educational Computing Research*, *1*(3), 253–274. http://doi.org/10.2190/BC76-8479-YM0X-7FUA

Engle, R. A. (2006). Framing Interactions to Foster Generative Learning: A Situative Explanation of Transfer in a Community of Learners Classroom. *The Journal of the Learning Sciences*, *15*(4), 451–498.

Engle, R. A., & Conant, F. R. (2002). Guiding Principles for Fostering Productive Disciplinary Engagement: Explaining an Emergent Argument in a Community of Learners Classroom. *Cognition and Instruction*, *20*(4), 399–483. http://doi.org/10.1207/S1532690XCI2004

Ericson, B., & Guzdial, M. (2014). Measuring demographics and performance in computer science education at a nationwide scale using AP CS data. In *Proceedings of the 45th ACM technical symposium on Computer science education* (pp. 217–222). http://doi.org/10.1145/2538862.2538918

Gick, M. L., & Holyoak, K. J. (1983). Schema induction and analogical transfer. *Cognitive Psychology*, *15*(1), 1–38. http://doi.org/10.1016/0010-0285(83)90002-6

Grover, S., & Pea, R. (2013). Computational Thinking in K-12: A Review of the State of the Field. *Educational Researcher*, *42*(1), 38–43. http://doi.org/10.3102/0013189X12463051

Harackiewicz, J. M., & Hulleman, C. S. (2010). The Importance of Interest: The Role of Achievement Goals and Task Values in Promoting the Development of Interest. *Social and Personality Psychology Compass*, *4*(1), 42–52. http://doi.org/10.1111/j.1751-9004.2009.00207.x

Hatano, G., & Inagaki, K. (1986). Two Courses of Expertise.

Hendricks, P. C. C., Alemdar, D. M., & Ogletree, D. T. W. (2012). Ac 2012-2994 : the Impact of Participation in Vex Robotics Competition on Middle and High School Students ' Inter- Est in Pursuing Stem Studies and Stem-Related Careers. *2012 ASEE Annual Conference*.

Hidi, S., & Renninger, K. A. (2006). The Four-Phase Model of Interest Development. *Educational Psychologist*, *41*(2), 111–127. http://doi.org/10.1207/s15326985ep4102

Interim CSTA K-12 Computer Science Standards. (2016). New York, NY.

K–12 Computer Science Framework. (2016). Retrieved from http://www.k12cs.org

Kelleher, C., & Pausch, R. (2005). Lowering the Barriers to Programming : a survey of programming environments and languages for novice programmers. *Science*, *37*(2), 83–137. http://doi.org/10.1145/1089733.1089734

Kruger, J., & Dunning, D. (1999). Unskilled and unaware of it: how difficulties in recognizing one's own incompetence lead to inflated self-assessments. *Journal of Personality and Social Psychology*, *77*(6), 1121–34. http://doi.org/10.1037/0022-3514.77.6.1121

Kurland, D. M., Pea, R. D., Clement, C., & Mawby, R. (1986). A Study of the development of programming ability and thinking skills in high school students. *Journal of Educational Computing Research*, *2*(4), 429–458. http://doi.org/10.2190/BKML-B1QV-KDN4-8ULH

Lave, J., & Wenger, E. (1991). *Situated Learning: Legitimate Peripheral Participation - Jean Lave, Etienne Wenger - Google Books*. *Cambridge University Press*. Cambridge, UK: Cambridge University Press. Retrieved from https://books.google.co.uk/books?id=ZVogAwAAQBAJ&printsec=frontcover&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false

Linn, M. C., & Dalbey, J. (1985). Cognitive consequences of programming instruction: instruction, access, and ability. *Educational Psychologist*, *20*(4), 191–206.

Liu, a, Newsom, J., Schunn, C., & Shoop, R. (2013). Students Learn Programming Faster through Robotic Simulation. *Tech Directions*, *72*(march), 16–19. Retrieved from http://www.education.rec.ri.cmu.edu/content/educators/research/files/p16-19 Shoop et al.pdf

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. http://doi.org/10.1016/j.chb.2014.09.012

Mayer, R. E. (2008). Applying the science of learning: evidence-based principles for the design of multimedia instruction. *The American Psychologist*, *63*(8), 760–769. http://doi.org/10.1037/0003-066X.63.8.760

Michel, O. (2004). Webots TM : Professional Mobile Robot Simulation. *International Journal of Advanced Robotic Systems*, *1*(1), 39–42. http://doi.org/10.1.1.86.1278

Miele, D. B., & Molden, D. C. (2010). Naive theories of intelligence and the role of processing fluency in perceived comprehension. *Journal of Experimental Psychology: General*, *139*(3), 535–557. http://doi.org/10.1037/a0019745

Pajares, F. (1996). Self-Efficacy Beliefs in Academic Settings. *Review of Educational Research*, *66*(4), 543–578. http://doi.org/10.3102/00346543066004543

Palumbo, D. B. (1990). Programming Language/Problem-Solving Research: A Review of Relevant Issues. *Review of Educational Research*, *60*(1), 65–89. http://doi.org/10.3102/00346543060001065

Papert, S. (1980). *Mindstorms*. New York, NY: Basic Books, Inc. Retrieved from http://dl.acm.org/citation.cfm?id=1095592

Papert, S., & Harel, I. (1991). SItuating Constructionism.

Pea, R. D., & Kurland, D. M. (1984). On the cognitive effects of learning computer programming. *New Ideas in Psychology*, *2*(2), 137–168. http://doi.org/10.1016/0732-118X(84)90018-7

Petre, M., & Price, B. (2004). Using robotics to motivate "back door" learning, 147–158. http://doi.org/10.1023/B:EAIT.0000027927.78380.60

Pomerantz, E. M., Altermatt, E. R., & Saxon, J. L. (2002). Making the grade but feeling distressed: Gender differences in academic performance and internal distress. *Journal of Educational Psychology*, *94*(2), 396–404. http://doi.org/10.1037/0022-0663.94.2.396

Puntambekar, S., & Hubscher, R. (2005). Tools for scaffolding students in a complex learning environment: What have we gained and what have we missed? *Educational Psychologist*, *40*(1), 1–12. http://doi.org/10.1207/s15326985ep4001

Reed, S. K. (1993). A schema-based theory of transfer. *Transfer on Trial: Intelligence, Cognition, and Instruction*, *1*, 39–67.

Repenning, A., Webb, D., & Ioannidou, A. (2010). Scalable game design and the development of a checklist for getting computational thinking into public schools. *SIGCSE 10 Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, 265–269. http://doi.org/10.1145/1734263.1734357

Robins, A., Rountree, J., & Rountree, N. (2010). Learning and Teaching Programming : A Review and Discussion. *Computer Science Education*, *13*(2), 137–172. http://doi.org/10.1076/csed.13.2.137.14200

Rusk, N., Resnick, M., Berg, R., & Pezalla-Granlund, M. (2008). New pathways into robotics: Strategies for broadening participation. *Journal of Science Education and Technology*, *17*(1), 59–69. http://doi.org/10.1007/s10956-007-9082-2

Salomon, G., & Perkins, D. N. (1987). Transfer of cognitive skills from programming: When and how? *Journal of Educational Computing Research*, *3*(2), 149–169.

Weick, K. E. (1984). Small wins: Redefining the scale of social problems. *American Psychologist*, *39*(1), 40–49. http://doi.org/10.1037/0003-066X.39.1.40

Wigfield, A., & Eccles, J. S. (2000). Expectancy–Value Theory of Achievement Motivation. *Contemporary Educational Psychology*, *25*(1), 68–81. http://doi.org/10.1006/ceps.1999.1015

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33. http://doi.org/10.1145/1118178.1118215

Witherspoon, E. B., Schunn, C. D., Higashi, R. M., & Baehr, E. C. (2016). Gender, interest, and prior experience shape opportunities to learn programming in robotics competitions. *International Journal of STEM Education*, *3*(1), 18. http://doi.org/10.1186/s40594-016-0052-1

Zeldin, A. L., & Pajares, F. (2000). Against the Odds: Self-Efficacy Beliefs of Women in Mathematical, Scientific, and Technological Careers. *American Educational Research Journal*, *37*(1), 215–246. http://doi.org/10.3102/00028312037001215

Supporting Information

Supplemental resources containing additional details and examples of this curriculum are available **here.**

The linked website provides limited access to a wireframe mockup example of one of the lessons from the Program Flow unit utilized in the study, including a contextual introduction, scaffolded mini-lessons, worked example code review, and an open-ended dynamic challenge, as well as all accompanying videos. Available for navigation in the side menu are all materials and video within the pages titled: Introduction with Container Transport, Looped Decision, Code Review and Strawberry Sorter Challenge (other menu items are restricted in this sample version).

Tables

Table 1. Analysis of challenge tasks and programming concepts for each curricular unit

Unit	Chapter	Challenge Task	Programming Concepts
Basic Movement	Moving Forward	Static	Sequences
	Turning	Static	Sequences
Sensors	Forward Until Near	Dynamic	Sequences, Conditions
	Turn for Angle	Static	Sequences, Conditions
	Color Sensor	Dynamic	Sequences, Conditions
Program Flow	Loops	Dynamic	Sequences, Conditions, Iteration
	If-Else	Dynamic	Sequences, Conditions, Iteration
	Repeated Decisions	Dynamic	Sequences, Conditions, Iteration

Table 2. An overview of the three functional dimensions and the three programming concepts of the Programming Assessment.

Dimensions	**Items**	**Concepts**	**Items**	**Example content**
Robot Programming	6	Sequences	*2*	*What sequence of movements will get the robot to the end of the maze?*
		Conditions	*2*	*At what distance sensor value will the robot stop moving?*
		Loops	*2*	*Which actions will the robot repeat if the bumper sensor is pushed in?*
General Programming	7	Sequences	*2*	*In what order will a program run if additional lines are added to it?*
		Conditions	*3*	*Can a condition always be evaluated as either true or false?*
		Loops	*2*	*Can program be written that will make a loop repeat in reverse?*
Computational Thinking	12	Sequences	*3*	*Will the removal of this line of the program change the display on a heart monitor?*
		Conditions	*5*	*At what combination of blood pressure readings will this heart monitor emit an alarm?*
		Loops	*4*	*Which of these two programs will identify the correct blood pressure in the least number of iterations?*

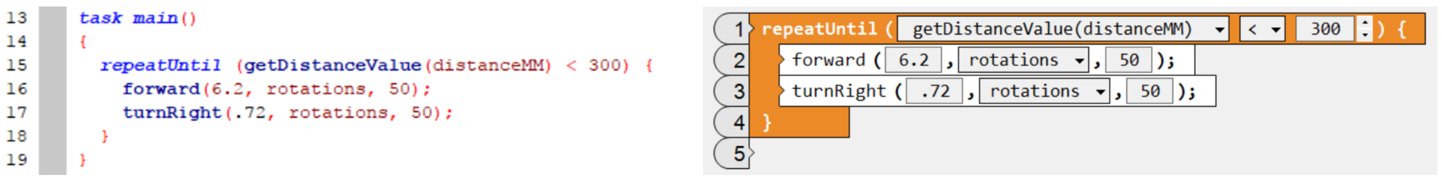
Table 3. Maximum values, pre-post grand means (and SD), and measure intercorrelations, with pre-post correlations on the diagonal in square brackets, post-post correlations above the diagonal, and pre-pre correlations below the diagonal.

	Pre		***Post***								
	N	***M (SD)***	***N***	***M (SD)***	***Max***	**1**	**2**	**3**	**4**	**5**	**6**
1. RP[1]	128	2.6 (1.3)	107	3.0 (1.3)	6	[.49]**	.51**	.43**	.15	.05	.23
2. GP[2]	128	3.1 (1.6)	107	4.0 (1.8)	7	.54**	[.37]**	.52**	.12	-.05	.12
3. CT[3]	127	4.5 (2.2)	107	5.3 (2.4)	12	.40**	.31**	[.41]**	.18	.08	.31*
4. Int.[4]	130	2.4 (.66)	107	2.2 (.75)	4	.13	.15	.24*	[.73]**	.68**	.64**
5. ID[5]	130	1.9 (.69)	107	1.8 (.75)	4	.09	.14	.17	.72**	[.70]**	.60**
6. CB[6]	130	4.3 (1.0)	107	4.0 (1.3)	6	.17	.13	.17	.64**	.62**	[.70]**

Note. * p <.01 ** p <.00[1]

[1] Robot Programming [2]General Programming [3]Computational Thinking [4] Interest [5] Identity [6] Competency Beliefs

Figure Legend



```
task main()
{
  repeatUntil (getDistanceValue(distanceMM) < 300) {
    forward(6.2, rotations, 50);
    turnRight(.72, rotations, 50);
  }
}
```

```
repeatUntil ( getDistanceValue(distanceMM) < 300 ) {
  forward ( 6.2 , rotations , 50 );
  turnRight ( .72 , rotations , 50 );
}
```

Figure 1. An example of a text-based programming language, and a visual programming language using graphical features.

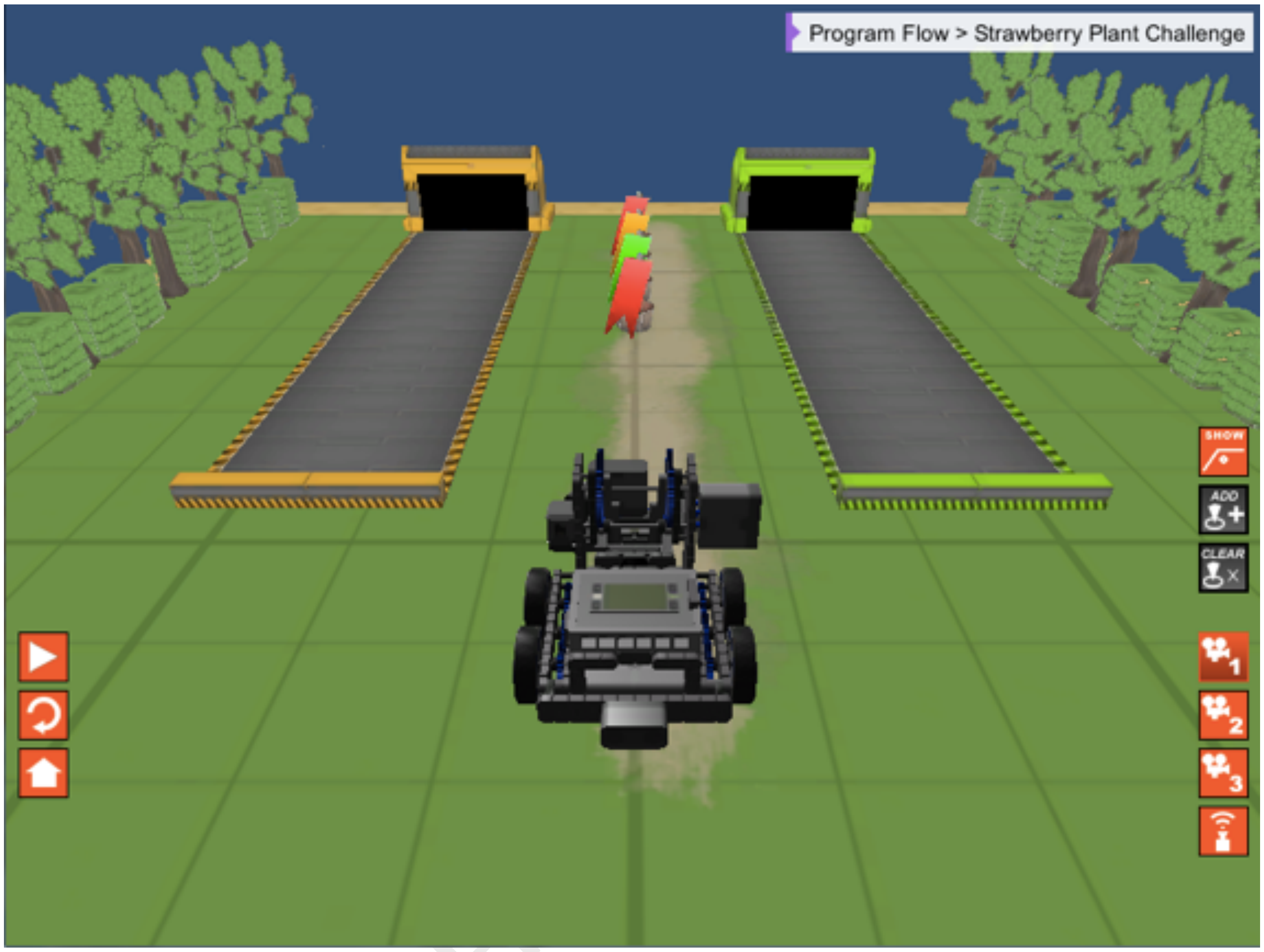
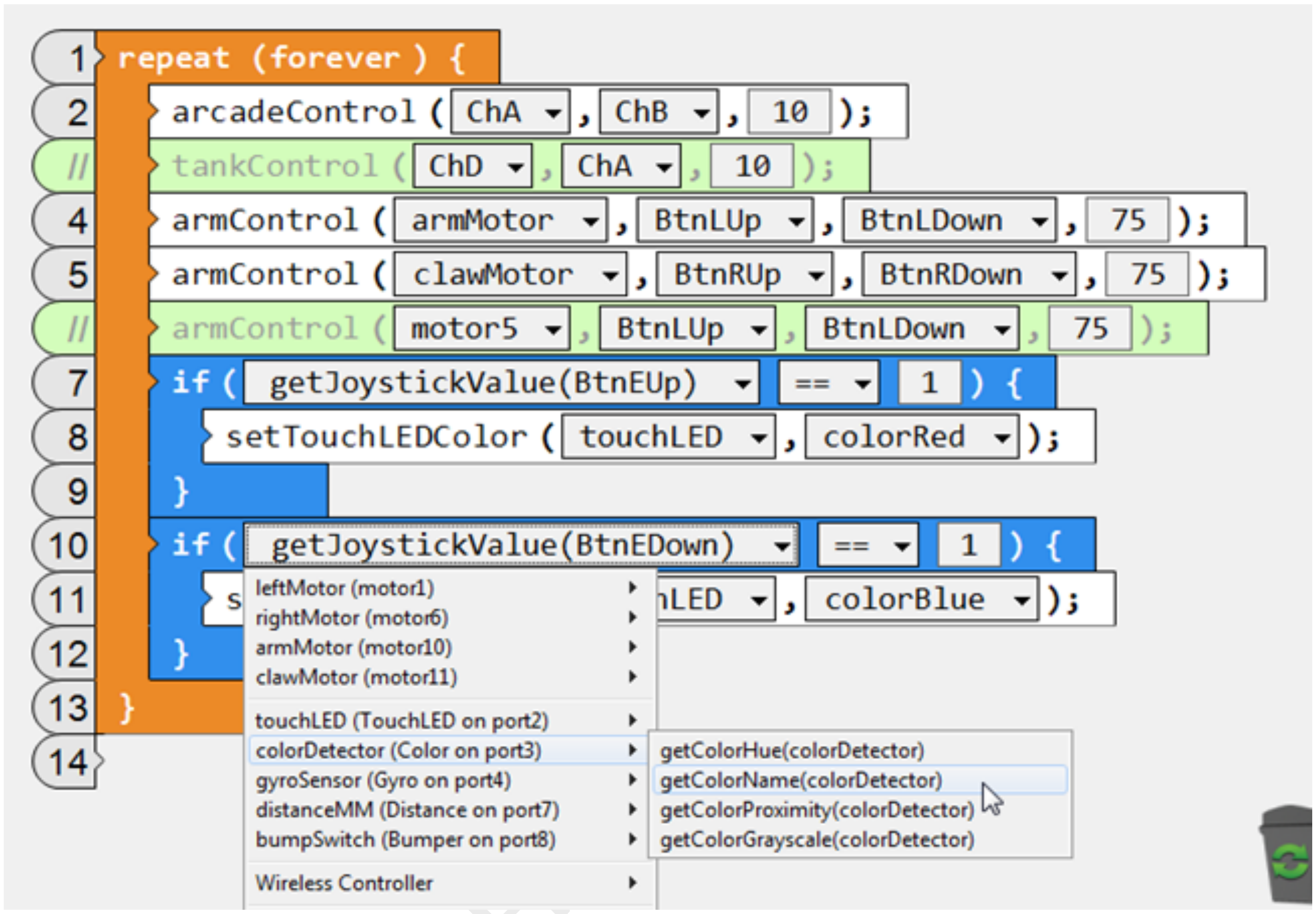


Figure 2. An example of a dynamic programming task. Using if/else statements, loops, and sensors, students program the robot to sort flags onto the left or right conveyor belt based on the color, which is dynamically assigned.



```
repeat (forever ) {
    arcadeControl ( ChA , ChB , 10 );
//  tankControl ( ChD , ChA , 10 );
    armControl ( armMotor , BtnLUp , BtnLDown , 75 );
    armControl ( clawMotor , BtnRUp , BtnRDown , 75 );
//  armControl ( motor5 , BtnLUp , BtnLDown , 75 );
    if ( getJoystickValue(BtnEUp) == 1 ) {
        setTouchLEDColor ( touchLED , colorRed );
    }
    if ( getJoystickValue(BtnEDown) == 1 ) {
        s nLED , colorBlue );
    }
}
```

leftMotor (motor1)
rightMotor (motor6)
armMotor (motor10)
clawMotor (motor11)
touchLED (TouchLED on port2)
colorDetector (Color on port3)
gyroSensor (Gyro on port4)
distanceMM (Distance on port7)
bumpSwitch (Bumper on port8)
Wireless Controller
getColorHue(colorDetector)
getColorName(colorDetector)
getColorProximity(colorDetector)
getColorGrayscale(colorDetector)

Figure 3. Examples of the graphical programming language used.

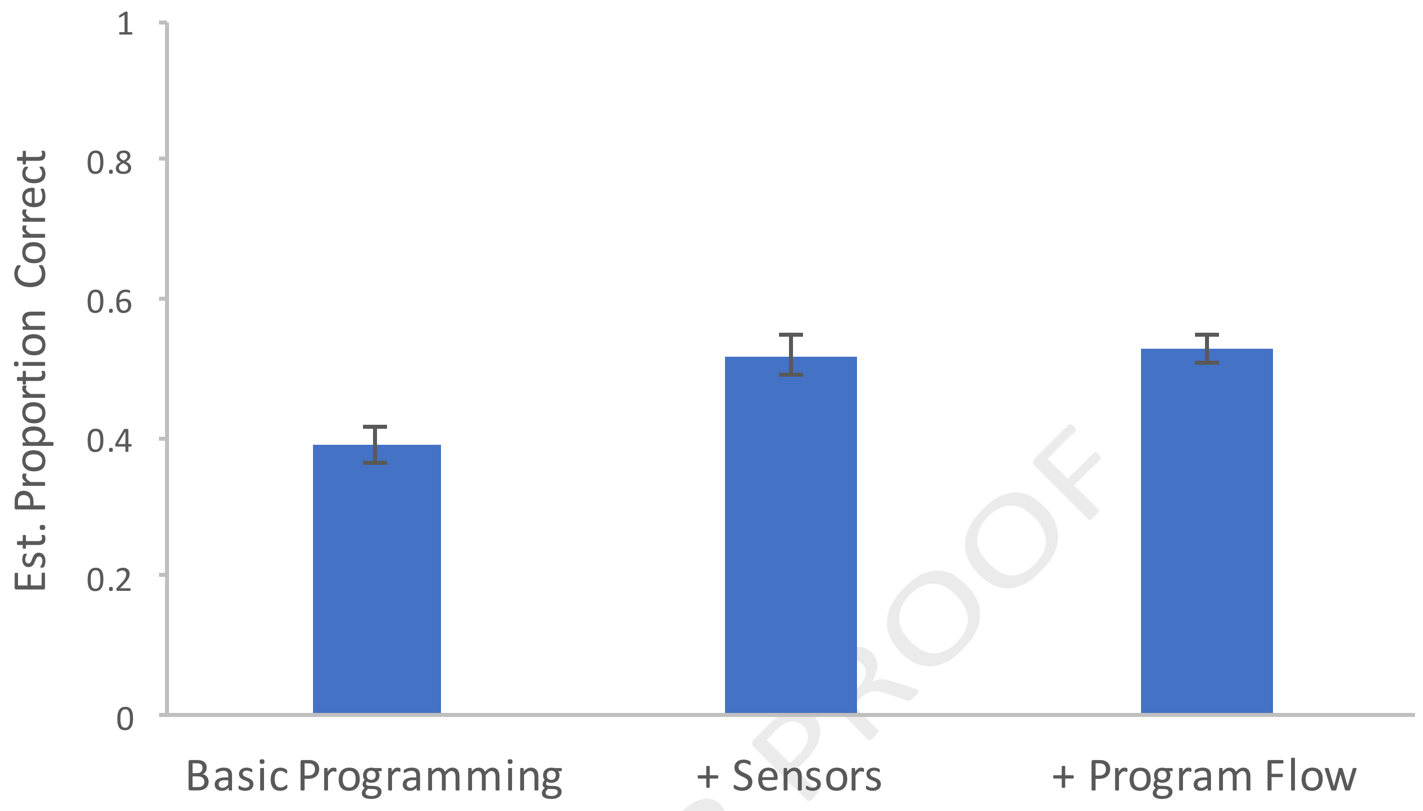


Figure 4. Mean overall skill postscore (with SE bars) by classroom condition, controlling for pre.

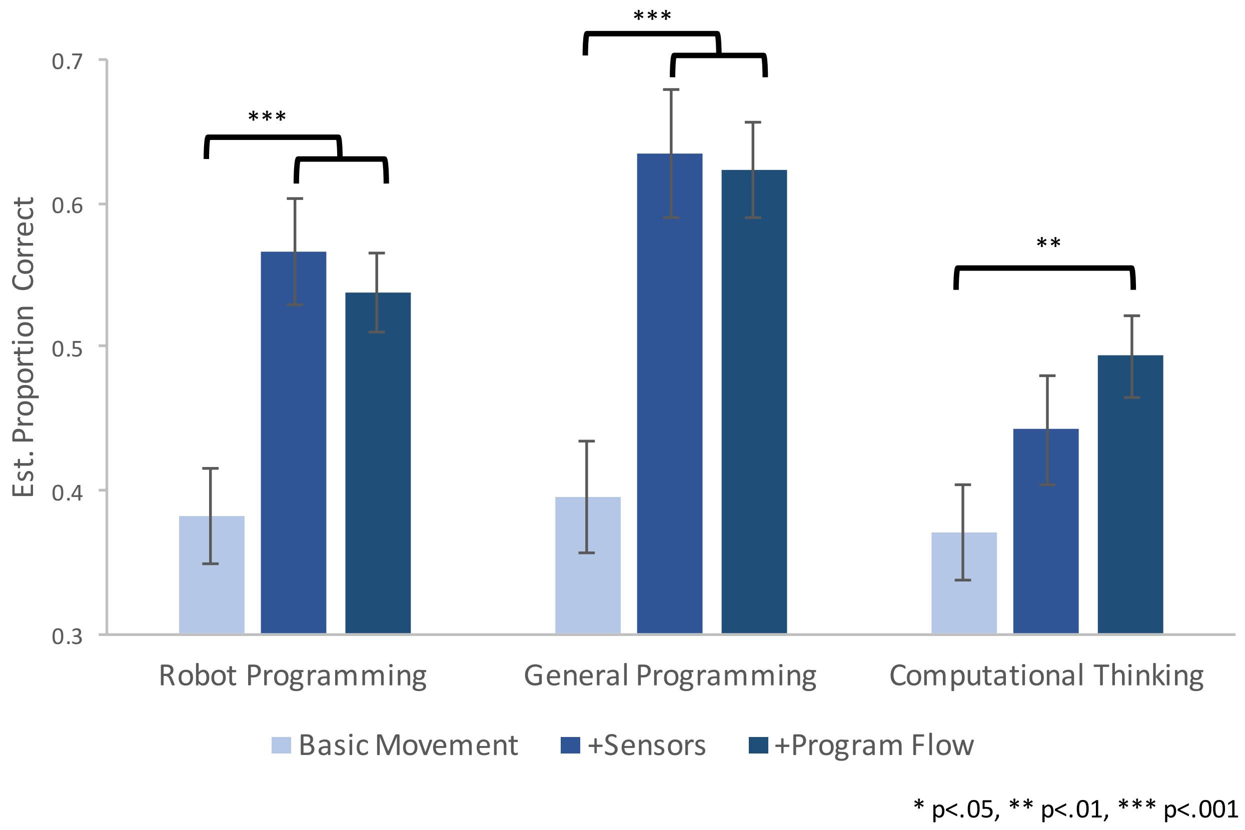


Figure 5. Post-test means (with SE bars) in each skill measure, controlling for pre-test, for each condition.

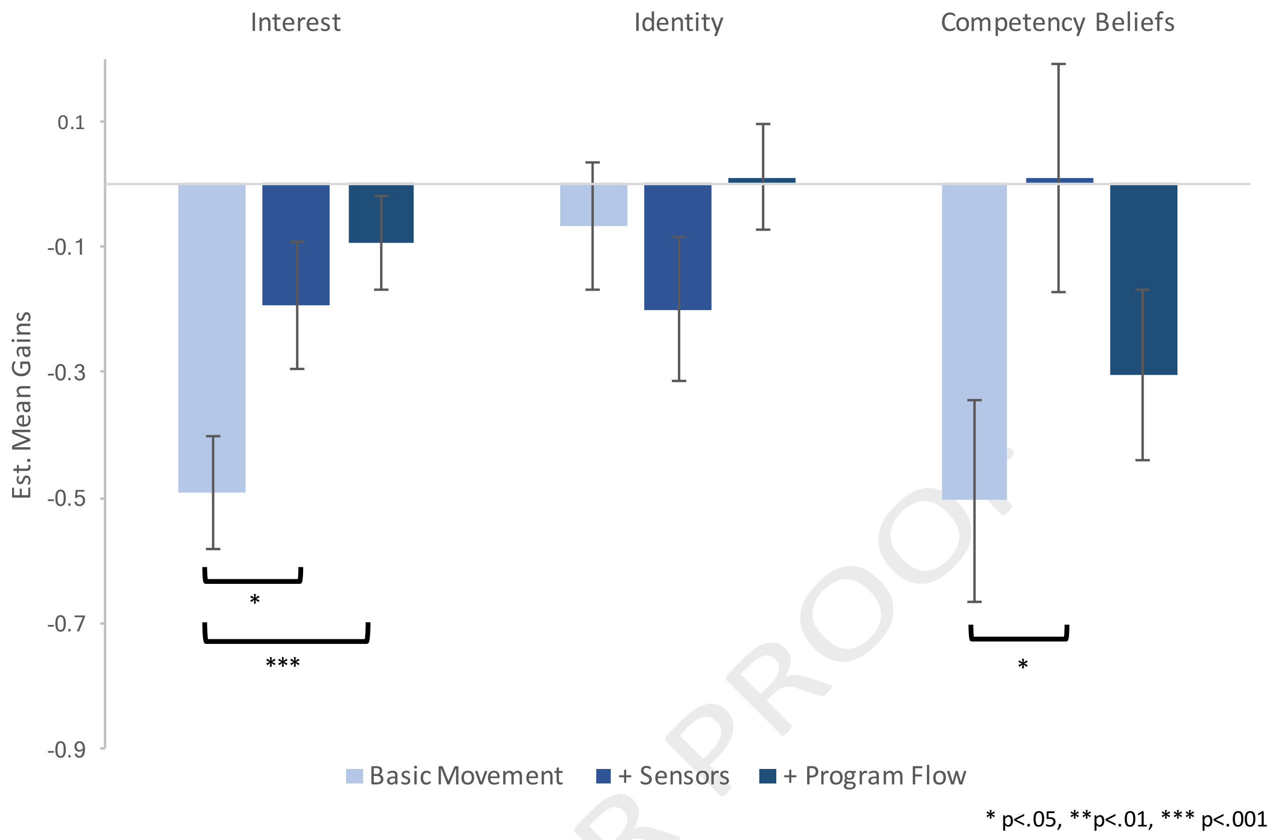


Figure 6. Mean pre-post changes in Interest, Identity, and Competency Beliefs, by amount of progress through the curriculum.

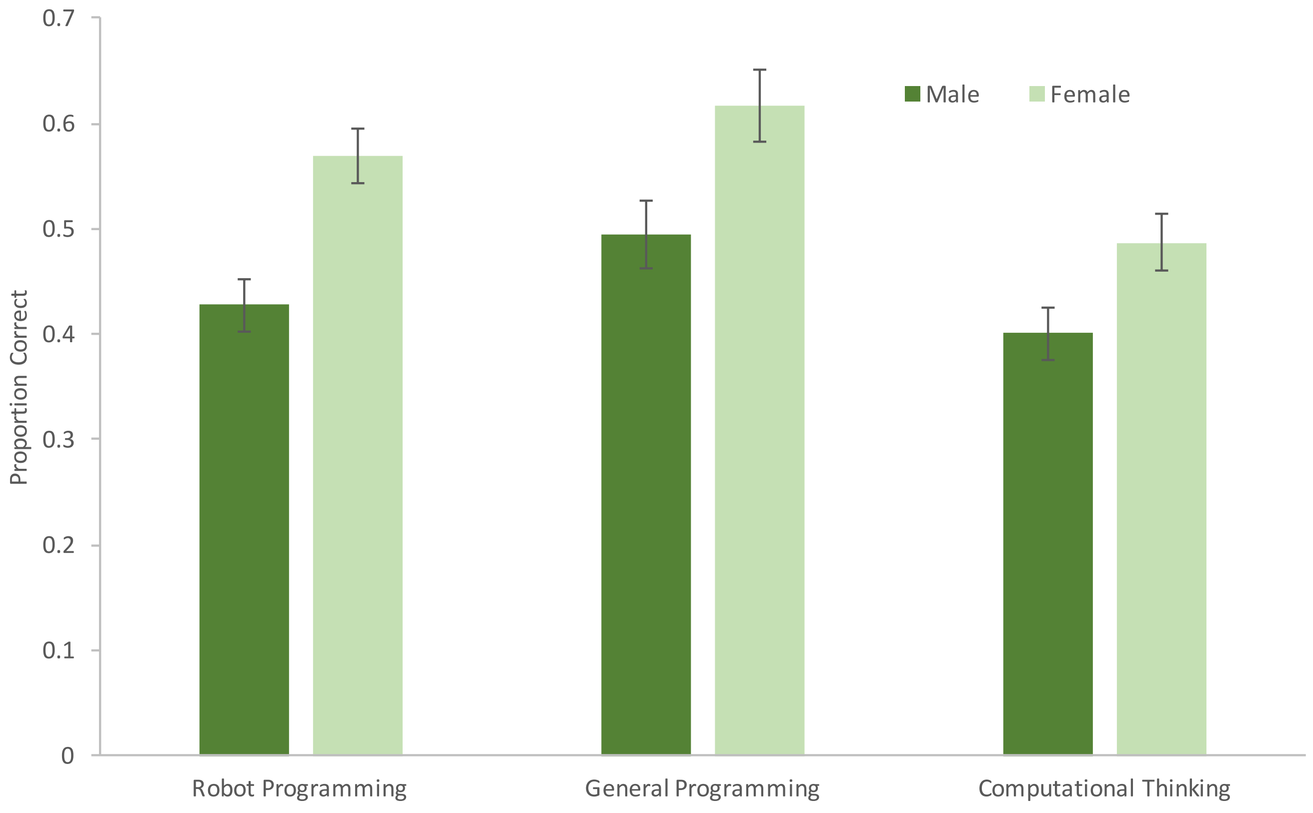


Figure 7. Post-test means (with SE bars) in each skill measure, controlling for pre-test, by gender.

Table 1. Analysis of challenge tasks and programming concepts for each curricular unit

Unit	Chapter	Challenge Task	Programming Concepts
Basic Movement	Moving Forward	Static	Sequences
	Turning	Static	Sequences
Sensors	Forward Until Near	Dynamic	Sequences, Conditions
	Turn for Angle	Static	Sequences, Conditions
	Color Sensor	Dynamic	Sequences, Conditions
Program Flow	Loops	Dynamic	Sequences, Conditions, Iteration
	If-Else	Dynamic	Sequences, Conditions, Iteration
	Repeated Decisions	Dynamic	Sequences, Conditions, Iteration

Table 2. An overview of the three functional dimensions and the three programming concepts of the Programming Assessment.

Dimensions	**Items**	**Concepts**	**Items**	**Example content**
Robot Programming	6	Sequences	*2*	*What sequence of movements will get the robot to the end of the maze?*
		Conditions	*2*	*At what distance sensor value will the robot stop moving?*
		Loops	*2*	*Which actions will the robot repeat if the bumper sensor is pushed in?*
General Programming	7	Sequences	*2*	*In what order will a program run if additional lines are added to it?*
		Conditions	*3*	*Can a condition always be evaluated as either true or false?*
		Loops	*2*	*Can program be written that will make a loop repeat in reverse?*
Computational Thinking	12	Sequences	*3*	*Will the removal of this line of the program change the display on a heart monitor?*
		Conditions	*5*	*At what combination of blood pressure readings will this heart monitor emit an alarm?*
		Loops	*4*	*Which of these two programs will identify the correct blood pressure in the least number of iterations?*

Table 3. Maximum values, pre-post grand means (and SD), and measure intercorrelations, with pre-post correlations on the diagonal in square brackets, post-post correlations above the diagonal, and pre-pre correlations below the diagonal.

	Pre		***Post***								
	N	***M (SD)***	***N***	***M (SD)***	***Max***	**1**	**2**	**3**	**4**	**5**	**6**
1. RP[1]	128	2.6 (1.3)	107	3.0 (1.3)	6	[.49]**	.51**	.43**	.15	.05	.23
2. GP[2]	128	3.1 (1.6)	107	4.0 (1.8)	7	.54**	[.37]**	.52**	.12	-.05	.12
3. CT[3]	127	4.5 (2.2)	107	5.3 (2.4)	12	.40**	.31**	[.41]**	.18	.08	.31*
4. Int.[4]	130	2.4 (.66)	107	2.2 (.75)	4	.13	.15	.24*	[.73]**	.68**	.64**
5. ID[5]	130	1.9 (.69)	107	1.8 (.75)	4	.09	.14	.17	.72**	[.70]**	.60**
6. CB[6]	130	4.3 (1.0)	107	4.0 (1.3)	6	.17	.13	.17	.64**	.62**	[.70]**

Note. * p <.01 ** p <.001

[1] Robot Programming [2]General Programming [3]Computational Thinking [4] Interest [5] Identity [6] Competency Beliefs

Appendix A.

Sample robotics programming items

Take a look at the program plan below. How will each individual line of code be run once it is programmed?

Line 1: Move forward for 5 seconds, at 100% speed
Line 2: Turn left 1 rotation, at 50% speed
Line 3: Move forward for 5 seconds, at 50% speed
Line 4: Turn right 1 rotation, at 50% speed

Select one:

O Only the first command runs
O The commands are run in order according to their line numbers
O All commands run at once
O The commands are run in a random order

Sample general programming item

Which of the following is true about conditions?

Select one:

O They must always end up either true or false
O They represent decision-making logic in a program
O You can write a condition that is always true or always false
O All of the above

Sample computational thinking item

Scenario C: Personal Fitness Devices
Personal fitness devices use electronic sensors to continuously monitor and track data about a user's heath such as steps taken, calories burned, and heart rate.
The BP-Sure company is developing a new feature for their fitness device that also measures the user's blood pressure, using sensors that detect a user's heartbeat. When the heart pushes blood through the arteries, the device records "Pressure 1", and when the heart is resting, the device records "Pressure 2".

The device can determine if a user's blood pressure is in the Normal, Medium or High range, by comparing blood pressure readings to the chart below.

Use the chart below to answer questions #19, #20 and #21.

Blood Pressure	Pressure 1 (p1)		Pressure 2 (p2)
Normal BP	p1 <= 120	**AND**	p2 <= 80
Medium BP	121 <= p1 <= 139	**AND**	81 <= p2 <= 89
High BP	p1 >= 140	**OR**	p2 >= 90

A new programmer on the team writes the following series of steps to determine the display when a user is in the "Normal BP" range:

```
(Line 1)   IF (p1  <=  120 AND
(Line 2)       p1  <=  121 AND
(Line 3)       p2  <=  80  AND
(Line 4)       p2  <=  81)
(Line 5)   THEN set  display = "Normal BP"
```

Which lines can be removed to make the code more efficient, while not changing the code output?

Select one:
O Line 1 and Line 4
O Line 2 and Line 3
O Line 2 and Line 4
O Line 1 and Line 3

Appendix B. Sample motivational survey items

Sample interest items

I wonder about how computer programs work.

O Never
O Once a month
O Once a week
O Every day

In general, when I work on programming, I:
O Hate it
O Don't like it
O Like it
O Love it

In general, I find programming:
O Very boring
O Boring
O Interesting
O Very interesting

After a really interesting programming activity is over, I look for more information about it.
O NO!
O No
O Yes
O YES!

Sample identity items

Please rate according to Not me and Exactly me.

	Exactly Me (1)	(2)	(3)	Not Me (4)
I am a "computer programming person".	O	O	O	O

8. Please rate these according to YES!, Yes, No, and NO!

	YES!	yes	no	NO!
a. My family thinks of me as a "programming person".	O	O	O	O
b. My friends think of me as a "programming person".	O	O	O	O
c. My teachers/instructors think of me as a "programming person".	O	O	O	O

Sample competency belief items

	Strongly Agree	Somewhat Agree	Agree	Disagree	Somewhat Disagree	Strongly Disagree
a. I am sure I could do advanced work in programming.	O	O	O	O	O	O
b. I am sure that I can learn programming.	O	O	O	O	O	O
c. I could do a good job as a programmer for an afterschool robotics team at my school.	O	O	O	O	O	O
d. I could get an A on a programming assignment in a technology class.	O	O	O	O	O	O