Crescendo: Engaging Students to Self-Paced Programming Practices

Wengran Wang, Rui Zhi, Alexandra Milliken, Nicholas Lytle, Thomas W. Price
North Carolina State University
Raleigh, NC
wwang33,rzhi,aamillik,nalytle,twprice@ncsu.edu

ABSTRACT

This paper introduces Crescendo, a self-paced programming practice environment that combines the block-based and visual, interactive programming of Snap*!*, with the structured practices commonly found in Drill-and-Practice Environments. Crescendo supports students with Parsons problems to reduce problem complexity, Use-Modify-Create task progressions to gradually introduce new programming concepts, and automated feedback and assessment to support learning. In this work, we report on our experience deploying Crescendo in a programming camp for middle school students, as well as in an introductory university course for non-majors. Our initial results from field observations and log data suggest that the support features in Crescendo kept students engaged and allowed them to progress through programming concepts quickly. However, some students still struggled even with these highly-structured problems, requiring additional assistance, suggesting that even strong scaffolding may be insufficient to allow students to progress independently through the tasks.

ACM Reference Format:
Wengran Wang, Rui Zhi, Alexandra Milliken, Nicholas Lytle, Thomas W. Price. 2020. Crescendo: Engaging Students to Self-Paced Programming Practices. In *The 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20), March 11–14, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3328778.3366919

1 INTRODUCTION

The past 40 years have seen significant changes in the design and implementation of introductory programming courses and materials, with a focus on the design of programming environments. Two types of programming environments have been commonly employed for students: 1) Novice Programming Environments (NPEs) such as Scratch [38], Alice [5] and MIT AppInventor [35] and 2) Drill-and-Practice Systems (D&PSs) such as CodeWorkout [9], CloudCoder [21] and Problets [24].

NPEs have been shown to improve students' engagement [10, 29] and grades [7] compared to traditional instruction. These environments feature block-based interfaces that can increase students' performance and learning [36, 43]. They allow students to create interesting and meaningful programs, such as visual and interactive games [38] and are traditionally paired with open-ended curricula (e.g. BJC [16]), letting students explore programming concepts with open-ended labs. Although exploration can create powerful learning experience for novices, it can also be overwhelming, creating a large cognitive load and requiring much instructor support. For example, Lee and Ko [26] studied instructional approaches in their Gidget programming game by comparing an "open-ended, creation-oriented" curriculum based on their puzzle designer to a highly-structured set of levels, and found students learned considerably more in the latter.

Drill-and-Practice Systems (D&PSs) such as CodingBat [33], CloudCoder [21], CodeWorkout [9], Infandango [22], and Problets [24] offer different advantages. They offer automated assessment and immediate feedback [9, 21, 22, 33], which is a critical component of learning [11]. They employ smaller assignments that isolate a single programming concept and allow students to focus on practicing one programming concept or strategy at a time. The structure of these environments also makes it easier to apply ideas from intelligent tutoring systems [41], such as mastery learning [6], adaptivity [12], and automated formative assessment [1], which can dramatically increase students' learning.

In this paper, we present Crescendo, a self-paced programming practice environment that attempts to blend the advantages of both NPEs and D&PSs. Crescendo's practice problems allow students to create interesting programs with visual and interactive output using block-based programming (as in NPEs), while focusing on smaller, single-concept tasks and offering automated feedback and assessment (as in D&PSs). Crescendo adds additional scaffolding to support students' independent learning, including a Use-Modify-Create scaffolding approach [25, 27], and the use of Parsons problems to increase learning efficiency [44]. The system also features detailed logging that can be used to better understand students' progress. The goal of Crescendo is to leverage all of these features to create a self-paced programming practice experience that: 1) engages students in creating interesting programs; 2) supports students to work independently; 3) provides a scaffolded progression of tasks that slowly increases in difficulty.

2 RELATED WORK

Novice Programming Environments: Novice programming environments (NPEs) such as Scratch [38], Alice [5], and Snap*!* [16] provide features such as graphical feedback and block-based programming languages to help beginners learn programming. These environments are useful for making games, animations, and other programs with visual and interactive output. They are widely used

SIGCSE '20, March 11–14, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-6793-6/20/03…$15.00
https://doi.org/10.1145/3328778.3366919

in introductory Computer Science curricula (e.g. [2], [18]). Features of NPEs, such as block-based programming and visual output, have been shown to improve student performance, learning and engagement. Compared with textual programming environments, block-based programming environments are shown to promote not only greater performance and learning gains [37, 43], but also students' longer-term interest toward future programming courses [43]. Additionally, visual output offered by NPEs has been shown to engage students and improve their programming performance [28].

On the other hand, other elements of NPEs have seen less empirical evaluation. For example, the costs and benefits of open-ended programming, in which students retain some control and creativity over their solution, is more widely debated. These open-ended programming tasks are popular in K-12 curricula (e.g. [16]), as well as informal learning settings, such as camps [45] and after school settings [30]. Proponents argue that these tasks offer students an engaging context [5], afford the creation of meaningful projects[38] and enable "differentiated learning" [16]. However, empirical results suggest that novices may need more structure: Lee and Ko [26] compared three different tools for programming learning: 1) A Python course on CodeAcademy; 2) a programming debugging puzzle game, Gidget with tutorials and step-by-step goals; and 3) the Gidget Puzzle Designer, with open-ended and creation-oriented design. A population of crowd workers were randomly assigned to each condition, and pre/post-tests showed significant learning gains only for the more structured CodeAcademy and Gidget Game groups, not for the open-ended Gidget Puzzle Designer group, suggesting that open-ended instruction alone may be ineffective. Additionally, Meerbaum-Salant et al. [31] discovered two detrimental programming habits among middle school students while completing open-ended Scratch projects - "bottom-up programming" strategies, as well as "Extremely Fine-Grained Programming". These findings suggest that there may be a need to supplement open-ended programming with more structured practice.

Drill-and-Practice Systems: In contrast to NPEs, which host open-ended programming tasks, Drill-and-Practice Systems (D&PSs) feature smaller and closed-ended tasks, grouped by concept categories. Such systems aim to help students practice isolated programming concepts one at a time. The closed-endedness of tasks within D&PSs allows the integration of immediate feedback through test cases at scale [9, 21, 22, 33]. Systems with such immediate feedback have been well-received among programming students: students in a CS1 course found the CodeWorkout system - an online D&PSs that features autograding and automated feedback - intuitive, and thought it helped improve their programming skills [9]. In addition, D&PSs afford explanations of step-by-step program execution, which could help novices trace code and develop their program models [24]. D&PSs could also blend in Intelligent Tutoring System(ITS) elements, such as the automatic code hints found within the ITAP system - a D&PS that teaches Python programming [39]. Intelligent Tutoring Systems simulate a human tutor, integrated with Artificial Intelligence, and are shown to improve students' programming learning more than traditional instruction [32]. DPSs with such ITS elements could be powerful learning tools for students.

NPE Features + D&PSs' Learning Structure:
Our goal with Crescendo is to combine the benefits of NPEs with D&PSs, a goal shared with some prior systems. For example, the Lambda Autograder [3] adds an autograding feature to Snap*!*, allowing instructors to author test cases that check students' block-based code for certain functionality. As detailed later, Crescendo integrates Lambda's autograding framework, adding additional functionality for testing purposes. BlockPy [4] is an NPE, with a hybrid block/text programming and visual data analysis output, which also offers real-time, adaptive feedback [19]. Similarly, iSnap adds on-demand, data-driven, contextualized hints and feedback to traditional open-ended programming based on a block-based programming environment [37]. In a classroom pilot study, Price et al. found that the hints in iSnap help students finish open-ended assignment objectives. Code.org's AP CSP curriculum uses small fixed self-paced curriculum lessons with added flexibility for students to create visual, interactive programs, and integrates automated feedback to give students real-time support.

Crescendo builds on these prior environments by adding additional support, including the Use-Modify-Create (UMC) framework [25] and Parsons problems [34]. The UMC framework works by beginning with high scaffolded activities (using pre-made code) and slowly fading scaffolding until students engage in open-ended creation tasks. This resembles other educational theories such as the Zone of Proximal Development [42], which argue that students should engage in activities with carefully-designed instructional support, that allows them to develop their skills to the point of needing less and less help. The UMC framework helps to keep students in the Zone of Proximal Development by starting with code that helps students understand the underlying concept (Use), scaffolding the completion of a new problem with similar structure (Modify), and finally requiring them to create their own solution from scratch (Create). As an educational framework to promote students' learning and computational thinking development, the UMC framework leads students to progress from using existing programs to creating their own programs [25]. Lytle et al. evaluated the UMC framework in a quasi-experimental study by comparing a UMC curriculum with a traditional curriculum with middle school students. They found that students in the UMC condition finished tasks faster and seemed to be more engaged in the class activity, and the UMC framework helped students build their sense of ownership over the programming artifacts they created [27]. Additionally, teachers perceived that the UMC curriculum was easy to teach. Block-based environments like Scratch with their social sharing components afford this Using and Modifying behavior, and users who engage in heavy "remixing" or modifying of prior projects interact with a wider variety of blocks and structures in their coding projects [8, 23].

In addition to a UMC progression that supports students across tasks, we also integrated Parsons problems into each Crescendo task, with the intent to help students engage in programming and learn efficiently. Parsons problems break a correct solution into code pieces and require students to re-arrange those mixed-up code blocks to restore the original solution. Previous work has

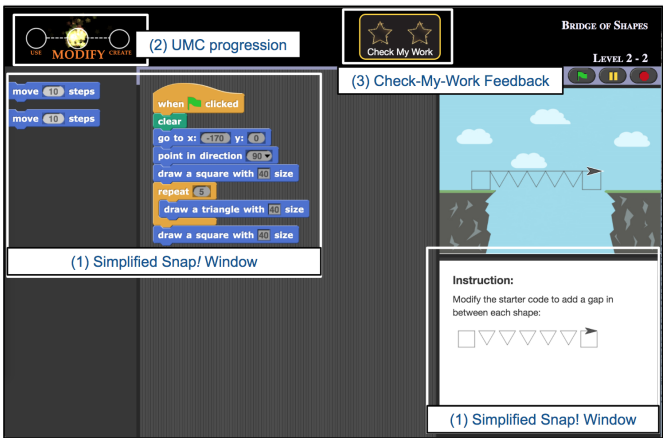

Figure 1: Crescendo Task Interface.

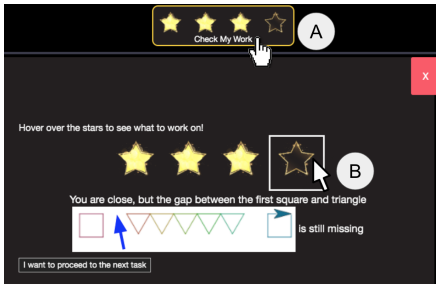

Figure 2: After (A) clicking on the Check-My-Work button, students can (B) hover on the stars to view the objectives.

shown that Parsons problems are engaging [13, 15, 17] and can improve students' programming efficiency [12, 14, 20, 44]. Ericson et al. designed and integrated Parsons problems into an ebook and found that more students attempted Parsons problems than multiple choice questions [13]. In a classroom study, Zhi et al. found that students spent less time on solving the block-based Parsons problems than equivalent code writing, and performed as well on post-test problems [44].

3 CRESCENDO SYSTEM

Learning in Crescendo is organised into a hierarchy: "Concepts – Challenges – 'Use-Modify-Create' Tasks" (Figure 1). It organizes small programming tasks into *challenges* based on programming *concepts* such as loops and conditionals. Each concept may have multiple challenges and each challenge covers a single learning objective of a concept. In each challenge, students accomplish three programming *tasks* following the UMC scaffolding. Crescendo is designed as a platform to host a variety of Snap*!* programming challenges and tasks. Instructors can plan the programming concepts, design challenges and tasks.

Figure 1 shows an example interface of a task. Crescendo augments and revises the Snap*!* programming interface with three important changes: (1) Parsons problem and simplified Snap*!* interface with short instructions; (2) the UMC task progression[27]; and (3) the Check-My-Work automated feedback. We describe each change below.

Parsons Problems and Simplified Snap*!* Interface: The tasks in Crescendo are Parsons problems, providing students with a limited number of blocks needed to solve the task. Figure 1-(1) shows the Parsons problem design based on that of Zhi et al. [44], which restricts user interactions with the programming environment, such as making the block inputs non-editable and removing interface elements that are irrelevant to the problem and may distract students. To simplify students' experience, they were also provided with only one or two lines of instructions with images in each task. In addition, many elements of the Snap*!* interface, such as creating new Sprites, have been removed for simplicity. Crescendo also builds on the logging feature from iSnap [37], which logs students' code snapshots and interactions with the system, such as click events (e.g. clicking a button) and code edits (e.g. dragging and dropping code).

The UMC Task Progression: Each challenge has three tasks, which follow the idea of "use, modify, create" (UMC) [25]. A specific example of a UMC task progression is presented in Section 4. The general design of the UMC progression starts from a *Use* task that asks students to run an existing program, understand its output, and sometimes make a small modification that reflects this understanding (e.g. change a literal value). In Crescendo's *Use* tasks, We conclude by asking students a short multiple-choice question about the code (e.g. "In the given example code, which block controls how many squares and triangles are drawn?"). Students can attempt the question until they get it correct. The *Modify* tasks in Crescendo give students incomplete or incorrect code, requiring them to improve the code according to task objectives. The *Create* tasks ask students to design their own code independently, giving them minimal help or support. To progress from *Modify* to *Create*, or from *Create* to the next challenge, students need to successfully finish all the objectives of the task.

The Check-My-Work Feedback: In each task, students can click on the Check-My-Work button to check their current progress. The button contains stars that represent the objectives of the current task. While solving a task, students can either run their code or click on the Check-My-Work button to update the stars and see the objectives that they have accomplished (filled golden stars), and have not (unfilled hollow stars). The drop-down output of the Check-My-Work button gives students an overview of the objectives they have or have not achieved (shown in Figure 2). When a student finishes the last objective, all stars will be activated, and a "Continue" button shows up, allowing her to click on it to move to the next task.

4 CRESCENDO CONTENT DESIGN

While Crescendo supports arbitrary programming tasks, in this work we implemented a small set of practice tasks for new programmers, which we deployed in two use cases (described below). These practice tasks focus on two introductory concepts: Loops and Conditionals. Described in Table 1, the 6 deployed challenges exemplify the challenges supported by Crescendo: each challenge enables the practice and learning of a small programming concept, which is necessary prerequisite for the next challenge. For example,

to complete Loops1, students need to understand code sequences inside a loop and the number of times a loop executes, which is prerequisite for Loops2 - focusing on how code executes inside and outside of a loop. Instructors can set order of the challenges while planning the course. And we recommend the challenges of different concepts to be interleaved when students are to practice more than one concept, as prior work suggests that students can learn better with interleaved exercises than blocked [40].

Table 1: 3 Loops Challenges and 3 Conditionals Challenges, with learning objectives.

Challenge	Learning Objective
Loops1 *+	Loops repeat code sequentially a given number of times
Loops2 *+	Combine code inside and outside of loops
Loops3 *+	Differentiate patterns inside nested loops
Conditionals1 *	Understand how a conditional pathway operates
Conditionals2 *	Differentiate conditional pathways inside nested conditionals
Conditionals3 *	Understand the sequence of actions in a sequence of conditionals

* Used in the Camp study.
+ Used in the CS0 course.

Inside each challenge, we designed a UMC task progression to provide tasks that are engaging, closed-ended, and slowly increase in difficulty. As an example, Figure 3 shows the progressions of the Loops3 challenge. Its goal is to help students understand and create programs with nested loops. The instructions are shown at the lower part of the figure, while the upper part – the "stage" of the program – is where students can execute the program and observe the output immediately. The "stages" in Figure 3 are outputs of starter code from the 3 tasks. For example, in the first (Use) task, given the starter code that already draws 3 rows of triangle series with 4 triangles in each row, students only need to change the number in the inner and outer repeats to achieve the goal.

The UMC tasks in this Loops3 challenge progress from: 1) helping students identify the repeating pattern of the inner and outer loop (Task 1); 2) reinforcing understanding of inner and outer loops by making changes to an already-existing nested loop (Task 2); 3) Independently creating nested loops that reproduce the pattern described in the instructions (Task 3). The other challenge designs follow a similar pattern of scaffolding, and our goal is to populate each Crescendo challenge with such progressive tasks, which focus on the same learning objectives, increasing slowly in difficulty.

5 USE CASES

In this section we present two use cases for Crescendo: an activity in a middle-school summer camp and a homework assignment for an undergraduate CS0 course. We deployed Crescendo in both, and we summarized our observations and findings. These two deployments give us complementary understanding about Crescendo: in the summer camp classroom we observed students' interactions with the system, and in the CS0 course we collected additional survey data.

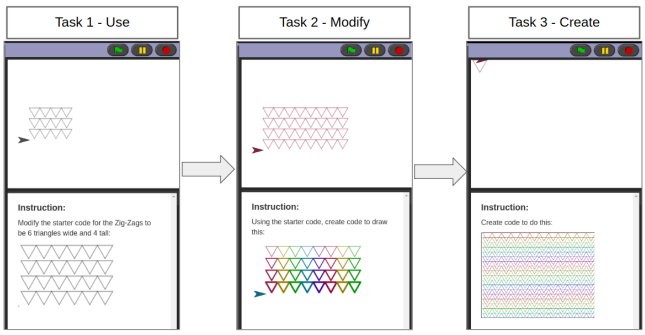

Figure 3: The UMC task design for Loops3, a challenge on nested loops. Students start by using existing code to create zig-zag shaped patterns. Output from students' starter code is displayed on the "stage."

5.1 Summer Camp

Students and Instructors: In the summer camp classroom, we had 23 middle-school students (12-14 years old) who were attending a coding summer camp held at a large, public university in the Southeast United States. The 2 instructors in the summer camp were local high school teachers from non-programming disciplines. A month before the week of the camp, we hosted a half-day training for the instructors where they programmed two introductory assignment (Draw Square and Click Alonzo) using Snap*!* and reviewed the camp schedule.

Activities: On the first day of the camp, the instructors gave the students a 54-minute Snap*!* tutorial to familiarize them with the programming environment, and to guide them through programming their first assignment (Draw Square). Then the students completed 2 of the 6 Crescendo challenges as a warm-up practice. After completing both Loops1 and Conditionals1, they completed two open-ended labs in the standard Snap*!* environment - Daisy Design (focusing on Loops) and Guessing Game (focusing on conditionals and some basic loops knowledge). After the two labs, students completed the remaining 4 challenges in Crescendo: Loops2, Conditionals2, Loops3 and Conditionals3. We observed engaged students working productively and successfully, employing new concepts throughout the day, though certainly tired by the end of the programming session.

5.2 CS0 Course Homework

Students and Instructor: The 50 students who completed the CS0 Crescendo homework were undergraduate non-major students at the same university where the summer camp was held. This course introduces basic programming concepts in Snap*!*, by integrating the Beauty and Joy of Computing curriculum [16]. Students had only taken 30 minutes in class to receive a Snap*!* tutorial before using this tool. While the instructor was the designer of Crescendo, he only gave students an instruction sheet and asked the student to complete the tasks in Crescendo as homework.

Activities: The instructor in CS0 planned the challenges to include only challenges around the concept of Loops: two required challenges (Loops1 and Loops2), and one optional challenge (Loops3). Since students were completing Crescendo tasks outside of class, we collected students' programming log data, as well as their likert-scale ratings of the system's difficulty, helpfulness and engagement.

5.3 Findings and Observations

To investigate whether Crescendo succeeded in its goal of creating a scaffolded and engaging learning experience for students, we present some quantitative findings, collected from Crescendo's log data, that help us summarize students' overall experience. We also report our own observations from Crescendo's deployment in the summer camp and the collected rating data from the CS0 homework activities.

Were tasks quick and consistent?

Recall that one of our goals for Crescendo was to provide a scaffolded progression of tasks that can be accomplished quickly, and slowly increases in difficulty. If this is the case, we would expect students to spend a relatively short and similar amount of time on each task. To investigate this, we calculated the average time students spent to finish each task in Crescendo with their standard errors (see Figure 4). The x-axis presents us the tasks, with L1U representing the Use task of Loops1, C2M representing the Modify task of Conditionals2, etc. The tasks with a "+" sign after their names were implemented in the CS0 homework activities. We marked the total number of students who successfully completed the task on the bottom of each bar, and their average time spent on the top of each bar [1]. We noted that most tasks took a fairly short amount of time (1-5 minutes). But the 3 tasks with feedback errors (not shown in Figure 4) took considerably more time (L3C: $m = 6.41$ min; C2M: $m = 16.805$ min; L3C+: $m = 8.608$ min). We believe this was because these later tasks were more difficult and not because the errors may have slowed students down.

Did students use the automated feedback?

Remember that one of our goals for Crescendo was to provide the benefits of immediate feedback. While we have no control group, one way of measuring this is to look at how frequently the Check-My-Work button was used. We hope that we would find frequent usage of the Check-My-Work button, indicating that the feature is useful to the students. From our data, we see a range of 0.14 to 3.03 among students' average frequency of clicking the button across different tasks, indicating that students did make good use of help features, but that it was concentrated on more difficult problems.

What do students think about Crescendo?

Another goal of Crescendo was to engage students to solve scaffolded progression of tasks that slowly increases in difficulty. If this is the case, we would see a gradual increase in students' perceived task difficulty, without losing their perceived engagement and task helpfulness.

To understand students' perceptions of their experience, we prompted the 50 students in the CS0 course to answer the following rating questions after they finish each task: From scale 1-5, how would you rate this task that you have just finished? 1) How difficult was the task you just completed? (difficulty) 2) How helpful was the task for your learning? (helpfulness) 3) How engaging did you find this task? (engagement). Figure 5 shows students' perceived mean difficulty, engagement, and helpfulness of each task, y-axis ranging from 1.5 to 4.5. We have included the number of rating records we collected for each task at the bottom of each data point. We noticed students experiencing gradually increasing difficulty level between each challenge, as well as inside the UMC progression of each challenge. Their average ratings on engagement and helpfulness kept relatively high (above 3.5). Comparing with their trends on the difficulty ratings, their engagement and helpfulness ratings had a similar but smaller-scaled increase across and within challenges.

While students' ratings on different tasks allowed us to understand students' experience with task progressions, we also wanted to understand how they perceive their experience with Crescendo overall. Compared with a normal Snap*!* interface, we wanted to see whether Crescendo may be more engaging and helpful for students' learning. We evaluated their overall perceptions with the system at the end of the Crescendo experience by asking them to do a likert scale rating from strongly disagree to strongly agree for the following question: This version of Snap*!* is different from the one you used in class (e.g. it checks your work and only gives you the blocks you need for each task). How much do you agree with the following statements: 1) This version of Snap*!* is more motivating for me; 2) This version of Snap*!* is more helpful to me. Among 48 students from whom we have collected the answers, many agreed or strongly agreed that learning in Crescendo is more helpful (68.75%), as well as more motivating (58.33%), with 2 (4.17%) and 3 (6.25%) saying disagree, none strongly. Additionally, we have also found 30 students (60%) in the CS0 course successfully completed the optional, third challenge (Loops3). This also suggested that students were engaged enough with Crescendo to continue working voluntarily.

6 DISCUSSION AND IMPLICATIONS

Based on the findings presented in the above section, in this section, we discuss the value added by Crescendo to its related work on current NPEs and other programming practice environments. The results have given us positive take-aways for future tool designs and classroom implementations, offering information for instructors to accommodate our tool to their own classrooms.

Interactive programs can still be engaging even when they are short and closed-ended.

One concern with moving from open-ended block-based programming practices to shorter closed-ended ones is that this would reduce students' engagement, but our rating results show that most students found Crescendo *more engaging* than their regular, more open-ended lab experience. Additionally, many CS0 students were engaged enough to complete optional tasks in Crescendo. This suggests that we can still reap the benefits of visual, interactive programming, even in quick and closed-ended tasks. We therefore recommend tool designers and teachers to add closed-ended tasks into block-based activities, for they not only serve as an effective practice opportunity, but also an engaging experience.

[1] We excluded students from tasks if they requested to skip the task or were still working when time ran out. We removed 1 task which only 6 students completed (C3C: $m = 11.95$ min).

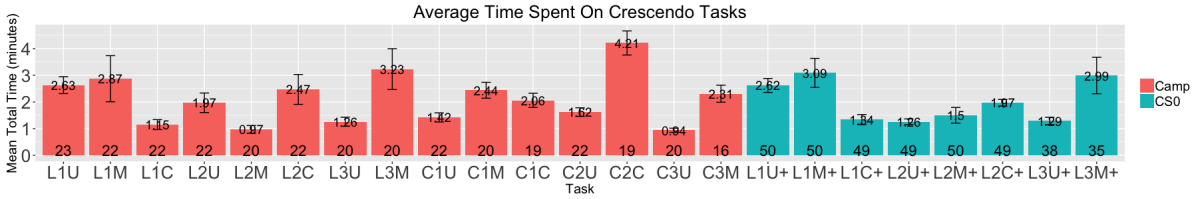

Figure 4: Average time spent on each Crescendo tasks for both Camp and CS0 course students with standard error bars.

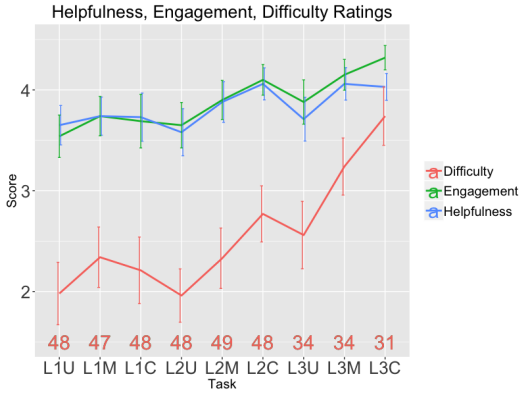

Figure 5: Students' average ratings on each category across different tasks with error bar.

The UMC scaffolding may keep students in the Zone of Proximal Development.
Within each challenge, from the Use task, through the Modify task, to the Create task, we found a desired, gradual increasing self-rated difficulty. This is consistent with the findings by Lytle et al. [27], showing that in a Use-Modify-Create progression, students' perceived difficulty grows slowly, instead of introducing sudden spikes of difficulty.

While students perceived increased difficulty, they did not take more than 5 minutes to complete most of the tasks, suggesting that they learned the necessary skills to tackle the difficult tasks. This is consistent with the theory of the Zone of Proximal Development [42], which encourages tutors and tools to keep students working on tasks at the edge of their ability, while offering scaffolded instructions to allow them to accomplish these tasks. Students' high helpfulness ratings suggest that they received this needed help from Crescendo. Our findings highlighted the observed scaffolding effect of the UMC task progression, whose content design ideas could be easily adapted in any classroom, even without support from Crescendo.

Immediate feedback allowed students to progress independently.
In both the camp and the CS0 homework activities, we observed that students were able to progress independently through Crescendo tasks, with most students able to finish the tasks within the given time period, receiving minimal help from the instructors.

The instructors also welcomed Crescendo, as it allowed them to act as facilitators of knowledge as students progressed independently, rather than lecturers. The automated and immediate feedback provided by Crescendo could be useful to reduce the grading burden on instructors, and could also help new, in-service programming teachers who are still transitioning from a non-programming background to more accurately assess students' work.

7 LESSONS LEARNED

Throughout the experience, we found some deficiency inside the system, that has cost students' struggle and confusion.

When providing starter code, it is important to distinguish code that should be modified from code that should not.
The starter code we provided to the students was designed to be fully modifiable. But that flexibility caused some students to overly focus on modifying the starter code, making them feel frustrated when they broke working starter code. In the future, the system should distinguish between code that should be modified and code that should not, for example, by fixing the starter blocks, and only allowing the blocks that can be modified to be draggable.

When students work at their own pace, some students may get left behind.
Another deficiency of the system we found in our middle school classroom is that students accumulated bigger and bigger differences in the time they took to complete Crescendo tasks. Although faster students were provided with non-Crescendo-based bonus tasks to be worked on after they finished the required Crescendo tasks, slower students still started to feel anxious when they were left behind. In the future, we want to make Crescendo support pair programming, add more bonus tasks to students, as well as give students more encouragement and positive feedback in the future task design, to help lower-performing students feel more motivated to progress inside the system.

8 CONCLUSION

This paper demonstrated the design and deployment of Crescendo, an initial attempt towards combining the benefits of Novice Programming Environments with Drill and Practice Systems. Adding the Use-Modify-Create task progression and Parsons problems on top of it, our case study shows Crescendo particularly effective in leading students into independent, motivating, and rewarding programming experience. Our future work will include adding more intelligent and adaptive elements to Crescendo, providing more personalized support to each individual student, as well as instructor-oriented features to help quickly create engaging and meaningful content.

REFERENCES

[1] Bita Akram, Wookhee Min, Eric N Wiebe, Bradford W Mott, Kristy Boyer, and James C Lester. 2018. Improving Stealth Assessment in Game-based Learning with LSTM-based Analytics. *Proceedings of the 11th International Conference on Educational Data Mining* (2018).

[2] Owen Astrachan and Amy Briggs. 2012. The CS principles project. *ACM Inroads* 3, 2 (2012), 38–42.

[3] Michael Ball. 2018. *Lambda: An Autograder for snap.* Technical Report. Technical Report. Electrical Engineering and Computer Sciences University of California at Berkeley.

[4] Austin Cory Bart, Javier Tibau, Eli Tilevich, Clifford A Shaffer, and Dennis Kafura. 2017. Blockpy: An open access data-science environment for introductory programmers. *Computer* 50, 5 (2017), 18–26.

[5] Stephen Cooper, Wanda Dann, and Randy Pausch. 2000. Alice: a 3-D tool for introductory programming concepts. In *Journal of Computing Sciences in Colleges*, Vol. 15. Consortium for Computing Sciences in Colleges, 107–116.

[6] Albert T Corbett. 2000. *Cognitive Mastery Learning in the ACT Programming Tutor.* Technical Report. 1–6 pages. http://www.aaai.org/Papers/Symposia/Spring/2000/SS-00-01/SS00-01-007.pdf

[7] Wanda Dann, Dennis Cosgrove, Don Slater, Dave Culyba, and Steve Cooper. 2012. Mediated transfer: Alice 3 to Java.. In *SIGCSE*, Vol. 12. Citeseer, 141–146.

[8] Sayamindu Dasgupta, William Hale, Andrés Monroy-Hernández, and Benjamin Mako Hill. 2016. Remixing as a pathway to computational thinking. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*. ACM, 1438–1449.

[9] Stephen H Edwards and Krishnan Panamalai Murali. 2017. CodeWorkout: short programming exercises with built-in data collection. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*. ACM, 188–193.

[10] Shelly Engelman, Brian Magerko, Tom McKlin, Morgan Miller, Doug Edwards, and Jason Freeman. 2017. Creativity in Authentic STEAM Education with EarSketch. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. ACM, 183–188.

[11] Michael L. Epstein, Amber D. Lazarus, Tammy B. Calvano, Kelly A. Matthews, Rachel A. Hendel, Beth B. Epstein, and Gary M. Brosvic. 2002. Immediate Feedback Assessment Technique Promotes Learning and Corrects Inaccurate first Responses. *The Psychological Record* 52, 2 (01 Apr 2002), 187–201.

[12] Barbara J Ericson, James D Foley, and Jochen Rick. 2018. Evaluating the Efficiency and Effectiveness of Adaptive Parsons Problems. In *Proceedings of the 2018 ACM Conference on International Computing Education Research*. ACM, 60–68.

[13] Barbara J. Ericson, Mark J. Guzdial, and Briana B. Morrison. 2015. Analysis of Interactive Features Designed to Enhance Learning in an Ebook. *Proceedings of the eleventh annual International Conference on International Computing Education Research - ICER '15* (2015), 169–178. https://doi.org/10.1145/2787622.2787731

[14] Barbara J Ericson, Lauren E Margulieux, and Jochen Rick. 2017. Solving parsons problems versus fixing and writing code. In *Proceedings of the 17th Koli Calling Conference on Computing Education Research*. ACM, 20–29.

[15] Barbara J Ericson, Kantwon Rogers, Miranda Parker, Briana Morrison, and Mark Guzdial. 2016. Identifying design principles for CS teacher Ebooks through design-based research. In *Proceedings of the 2016 ACM Conference on International Computing Education Research*. ACM, 191–200.

[16] Dan Garcia, Brian Harvey, and Tiffany Barnes. 2015. The beauty and joy of computing. *ACM Inroads* 6, 4 (2015), 71–79.

[17] Stuart Garner. 2007. An Exploration of How a Technology-Facilitated Part-Complete Solution Method Supports the Learning of Computer Programming. *Issues in Informing Science and Information Technology* 4 (2007), 491–501.

[18] Joanna Goode, Gail Chapman, and Jane Margolis. 2012. Beyond curriculum: the exploring computer science program. *ACM Inroads* 3, 2 (2012), 47–53.

[19] Luke Gusukuma, Virginia Tech, Austin Cory Bart, Virginia Tech, Dennis Kafura, Virginia Tech, Jeremy Ernst, and Virginia Tech. 2018. Misconception-Driven Feedback : Results from an Experimental Study. 1 (2018), 160–168.

[20] Kyle J. Harms, Noah Rowlett, and Caitlin Kelleher. 2015. Enabling independent learning of programming concepts through programming completion puzzles. *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC* 2015-Decem, October 2015 (2015), 271–279. https://doi.org/10.1109/VLHCC.2015.7357226

[21] David Hovemeyer and Jaime Spacco. 2013. CloudCoder: a web-based programming exercise system. *Journal of Computing Sciences in Colleges* 28, 3 (2013), 30–30.

[22] Michael Hull, Dan Powell, and Ewan Klein. 2011. Infandango: automated grading for student programming. (2011).

[23] Prapti Khawas, Peeratham Techapalokul, and Eli Tilevich. [n. d.]. Unmixing Remixes: The How and Why of Not Starting Projects from Scratch. ([n. d.]).

[24] Amruth N Kumar. 2006. Explanation of step-by-step execution as feedback for problems on program analysis, and its generation in model-based problem-solving tutors. *Technology, Instruction, Cognition and Learning (TICL) Journal* 4, 1 (2006).

[25] Irene Lee, Fred Martin, Jill Denner, Bob Coulter, Walter Allan, Jeri Erickson, Joyce Malyn-Smith, and Linda Werner. 2011. Computational thinking for youth in practice. *Acm Inroads* 2, 1 (2011), 32–37.

[26] Michael J Lee and Andrew J Ko. 2015. Comparing the effectiveness of online learning approaches on CS1 learning outcomes. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*. ACM, 237–246.

[27] Nicholas Lytle, Veronica Cateté, Danielle Boulden, Yihuan Dong, Jennifer Houchins, Alexandra Milliken, Amy Isvik, Dolly Bounajim, Eric Wiebe, and Tiffany Barnes. 2019. Use, Modify, Create: Comparing Computational Thinking Lesson Progressions for STEM Classes. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*. ACM, 395–401.

[28] Nicholas Lytle, Mark Floryan, and Tiffany Barnes. 2019. Effects of a Pathfinding Program Visualization on Algorithm Development. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. ACM, 225–231.

[29] Brian Magerko, Jason Freeman, Tom McKlin, Scott McCoid, Tom Jenkins, and Elise Livingston. 2013. Tackling engagement in computing with computational music remixing. In *Proceeding of the 44th ACM technical symposium on Computer science education*. ACM, 657–662.

[30] John H Maloney, Kylie Peppler, Yasmin Kafai, Mitchel Resnick, and Natalie Rusk. 2008. *Programming by choice: urban youth learning programming with scratch*. Vol. 40. ACM.

[31] Orni Meerbaum-Salant, Michal Armoni, and Mordechai Ben-Ari. 2011. Habits of programming in scratch. *ITiCSE'11 - Proceedings of the 16th Annual Conference on Innovation and Technology in Computer Science* (2011), 168–172. https://doi.org/10.1145/1999747.1999796

[32] John C Nesbit, Olusola O Adesope, Qing Liu, and Wenting Ma. 2014. How Effective are Intelligent Tutoring Systems in Computer Science Education?. In *2014 IEEE 14th International Conference on Advanced Learning Technologies*. IEEE, 99–103.

[33] Nick Parlante. 2019. CodingBat. Retrieved from ttps://codingbat.com/java.

[34] Dale Parsons and Patricia Haden. 2006. Parson's programming puzzles: a fun and effective learning tool for first programming courses. In *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52*. Australian Computer Society, Inc., 157–163.

[35] Shaileen Crawford Pokress and José Juan Dominguez Veiga. 2013. MIT App Inventor: Enabling personal mobile computing. *arXiv preprint arXiv:1310.2830* (2013).

[36] Thomas W Price and Tiffany Barnes. 2015. Comparing textual and block interfaces in a novice programming environment. In *Proceedings of the eleventh annual International Conference on International Computing Education Research*. ACM, 91–99.

[37] Thomas W Price, Yihuan Dong, and Dragan Lipovac. 2017. iSnap: towards intelligent tutoring in novice programming environments. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. ACM, 483–488.

[38] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, et al. 2009. Scratch: programming for all. *Commun. ACM* 52, 11 (2009), 60–67.

[39] Kelly Rivers and Kenneth R. Koedinger. 2017. Data-Driven Hint Generation in Vast Solution Spaces: a Self-Improving Python Programming Tutor. *International Journal of Artificial Intelligence in Education* 27, 1 (2017), 37–64. http://link.springer.com/10.1007/s40593-015-0070-z

[40] Kelli Taylor and Doug Rohrer. 2010. The effects of interleaved practice. *Applied Cognitive Psychology* 24, 6 (2010), 837–848.

[41] Kurt Vanlehn. 2006. The Behavior of Tutoring Systems. *Int. J. Artif. Intell. Ed.* 16, 3 (Aug. 2006), 227–265. http://dl.acm.org/citation.cfm?id=1435351.1435353

[42] Lev Vygotsky. 1978. Interaction between learning and development. *Readings on the development of children* 23, 3 (1978), 34–41.

[43] David Weintrop and Uri Wilensky. 2017. Comparing block-based and text-based programming in high school computer science classrooms. *ACM Transactions on Computing Education (TOCE)* 18, 1 (2017), 3.

[44] Rui Zhi, Min Chi, Tiffany Barnes, and Thomas W Price. 2019. Evaluating the Effectiveness of Parsons Problems for Block-based Programming. In *Proceedings of the 2019 ACM Conference on International Computing Education Research*. ACM, 51–59.

[45] Rui Zhi, Nicholas Lytle, and Thomas W Price. 2018. Exploring Instructional Support Design in an Educational Game for K-12 Computing Education. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. ACM, 747–752.