

# Debugging for Art's Sake: Beginning Programmers' Debugging Activity in an Expressive Coding Context

Corey Brady, Melissa Gresalfi, Selena Steinberg, and Madison Knowe  
corey.brady@vanderbilt.edu, melissa.gresalfi@vanderbilt.edu, selena.k.steinberg@vanderbilt.edu,  
madison.l.knowe@vanderbilt.edu  
Vanderbilt University

**Abstract:** Debugging is fundamental to the theory, practice, and learning of computing, and recent research suggests that a learning trajectory for debugging can be defined alongside trajectories for other core disciplinary practices. At the same time, other work in computing education has pressed the field to broaden its conception of the contexts where computational thinking occurs, identifying debugging activities and practices across diverse and multi-modal settings. In resolving this productive tension between systematically describing debugging and recognizing its broad reach, we argue researchers should attend to rich descriptions of situated debugging, especially among beginning debuggers. We present data from a week-long, free summer camp, Code Your Art, that engaged middle-school students in creating expressive computational visual effects. Here we find that students' responses to debugging tasks varied sharply across tasks. We argue that debugging work emerges in interaction with features of the environment, and we discuss design refinements we have made to pursue and study this conjecture.

## Introduction

Debugging is fundamental to computing and to computing education, and it is a core feature of professional programming. For computing education, debugging tasks can play a pivotal role in learning as they can be used to introduce or assess important computational ideas. Moreover, learners' interactions with debugging tasks can impact their motivation and sense of belonging in the classroom community and in computing more generally (Lui et al, 2017). Research has made efforts to understand the way debugging, as a practice, might develop over time. For example, Rich and colleagues (2019) have proposed a learning trajectory for debugging, identifying it as a separable constellation of skills and practices.

Meanwhile, recent approaches to computing and computational thinking have broadened the contexts in which learners are seen to engage with computation, and hence with debugging. Though perhaps most strongly visible in work on computational making and e-textiles, this trend toward widening context also appears in increased emphasis on networking, distributed computing, and computing in social context. Such efforts may raise questions about the degree to which debugging can be treated as a monolithic activity or skill, across this range of settings. They may instead suggest situation-specific connections with test-and-refine patterns from other fields (e.g., the design cycle in engineering or the modeling cycle in mathematics). Following this logic, one might be skeptical about the broad applicability of models of debugging as a single, discipline-specific practice centered on virtual constructions in text-based coding environments. We propose that these two trends (carefully describing debugging and broadening its reach) create a productive tension—on one hand producing systematic descriptions of learning and development over time, and on the other extending the reach of the construct to new, rich and multi-modal settings where it is increasingly occurring. Both trends provoke a need in the field for building deep understandings of the responses that beginning programmers exhibit in debugging settings in particular contexts.

In this paper we work to address this need, focusing on a programming context that bridges the traditional and the new: a choice-based learning setting (a summer camp called Code Your Art) that used a text-based coding environment for expressive purposes in creating visual art. We aim to understand the variability of students' debugging activity in this setting and to generate hypotheses about the sources of this variability, as we address the following questions: (1) How variable are beginning programmers' ways of responding to and engaging with debugging tasks in the Code Your Art context? and (2) How did different debugging tasks offer opportunities for learners to tap into knowledge resources?

We found that a wide range of forms and levels of engagement with debugging could emerge even within the particular form of computational creation realized in our camp setting. We propose that students' activity, the resources they draw upon, and ultimately their success in debugging emerge in interaction with features of the environment. Thus, we argue that it is important to consider the circumstances under which debugging is taking place and how beginning programmers' computational habits and experiences intersect with the kinds of tasks and code that they are attempting to debug. Rather than (or alongside) considering stable attributes of "effective

and ineffective novices” (Robins, 2019), then, we argue for attending to situation-specific features of debugging and ways that learners frame their engagement. What students take to be their goals, what kinds of problem they are trying to fix, and what they *need* to attend to, all contribute to the overall “debugging experience” that students have. While there is reason to think that people improve at these tasks over time, we argue it may be premature to think about a trajectory of individual learning independent of factors of learning in context.

## Brief overview of debugging literature

Essentially every effort to create a computational artifact requires debugging of some kind, and rather than framing debugging in terms of errors and deficiencies in expressing one’s knowledge or understanding, computer science educators increasingly see debugging as an organic part of elaborating and expressing ideas in computational media and through computational representation infrastructures. This shift is reflected in the inclusion of debugging in learning trajectories, with a focus on debugging as element of a system of disciplinary competencies including the use of decomposition, iteration, and conditionals (Rich et al, 2017; 2018; 2019).

This literature is rapidly evolving, yet much of the research base on debugging has historically proceeded from descriptions of expert practices and/or contrasts between experts and novices (Bednarik, 2012; Gould & Drongowski, 1974; Lin et al., 2016; Vessey, 1985). Rich characterizations of novice activities on their own are less common. In addition to the tendency of expert-novice approaches to characterize novices in deficit terms and by contrast to experts, there is also a tendency to view novices’ behaviors as indicating stable attributes *of the novices*, rather than responses to features of situations. For instance, an influential categorization of programming behavioral patterns among novices distinguishes “stoppers,” “movers,” and “tinkerers” (Perkins et al, 1986). In this view, movers “keep trying, experimenting with and modifying their code. They can use feedback about errors effectively to solve programs and progress.” and “[t]inkerers are extreme movers who are not able to trace their code and may be making changes more or less at random, with little chance of progress” (Robins, 2019, p. 337).

Furthermore, even traditional accounts of *expert* debugging may themselves be outmoded. Professional debugging practice is evolving, as is the image of expertise that emerges in debugging. This is not an entirely new trend, as some early accounts emphasized fluidity and iteration; Turkle and Papert’s (1992) work revealed a diversity of styles of debugging and programming. Nevertheless, the emergence of tools that encourage fluid debugging (e.g., by enabling “live coding” (cf, Blackwell, McLean, Noble, & Rohrerhuber, 2014)) may increase the view of debugging as blended with and interacting with other programming activities and practices. In this context, we argue that “thick” descriptions of beginning programmers’ responses to debugging challenges are needed as the field works to conceptualize debugging and its role.

## Theoretical approach and design perspectives

While we aim to benefit from the insights in the research base on debugging, we approach the problem of understanding beginning programmers’ interactions with debugging tasks with the assumption that human activity is a joint accomplishment between people and the learning environment, which includes people, tools, norms, and broader social narratives that give significance to interaction (Hall & Jurow, 2015; Nolen, Horn, & Ward, 2015; Wertsch, 1994). This assumption broadens the focus of analysis from questions that ask what an individual is doing to include descriptions of how activity is unfolding. This sociocultural framing of learning has demonstrated that the very concepts of “novice” or “expert” fail to account for the rich and varied performances we see in folks with more and less experience. Perhaps the most well-known re-imagining of expert-novice roles can be seen in the work of Lave and Wenger (1991), who reframed these ideas by considering the path through activity that a newcomer might take in the process of learning and becoming a member of a community. In this account, the change in participation that characterizes learning can only be understood in relation to the opportunities for learning that are present, and the ways that those opportunities shape and sometimes determine what, ultimately, a learner can do. The idea that learning is constituted in participation is an important lens for our work, as it attunes us to the ways that tools, classroom norms, and interactions with others afford particular kinds of opportunities to learn. In broadening our analysis to include these other elements of the activity system, we find ourselves asking different kinds of questions—not “what does novice debugging look like,” but rather, “why do students debug in that way *with that model* and *in this space*.” Looking across activities can often help to highlight the role that these different affordances (Greeno, 1994) can play in the ultimate activity that unfolds.

We build upon this socio-cultural perspective taking a Constructionist (Papert & Harel, 1991) approach to the design of our learning environments, to the significance of debugging, and to the role of debugging in creative computational work. Broadly, Constructionist designs reflect the idea that learning processes can be both supported and illuminated when learners create public artifacts that are personally meaningful (Kafai 2006; Papert & Harel 1991). Debugging in particular also holds a central role in Constructionist approaches to computational thinking and learning. From early articulations in *Mindstorms* (Papert, 1980), the surprise associated with

debugging, and the unexpected potential inherent in “buggy” lines of code, figure as drivers for the creative process of programming as a whole. Programming appears here as an interactive activity with a dynamic representation that gives its participants the opportunity to *debug their thinking* and achieve new perspectives on phenomena described or produced in code: ideas emerge and are shaped in interaction. E.M. Forester’s description of written composition captures the spirit: “How do I know what I think till I see what I say?” (Forester, 1970)

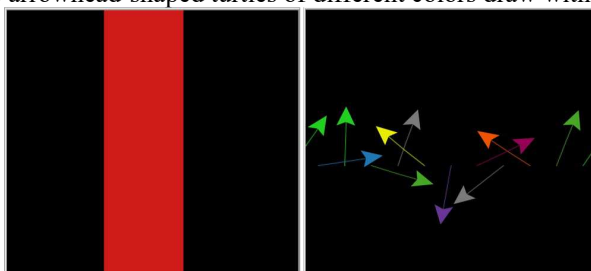
Papert and the Constructionists’ approach also endorses a view of computational thinking that explicitly connects it with expressive and artistic activities. Although the LOGO *turtle*, the central agent of Papert’s initial vision, was clearly a citizen of “MathLand” (Papert, 1980), it was also enthusiastically presented as a tool for the learner to create beautiful and visually compelling creations. Turtles have since been used to explore forms of artistic expressivity, in environments such as TurtleArt (Bontá, Papert, & Silverman, 2010) and more recently, Scratch (Resnick et al, 2009). A fundamental idea behind this strand of the Constructionist tradition is the notion that creating turtle graphics or producing media with turtles can introduce learners to the core principles of turtle geometry and/or programming in playful ways (Papert, 1980; Papert & Harel, 1991). Debugging in this context can be a part of the process of discovery, in which surprises in the computer’s execution of authors’ code suggest possible directions the computational artifact can take. Describing a similar productive tension between expressivity and the medium of representation, writer, Arts-and-Crafts designer, and printmaker William Morris famously said, “You can’t have art without resistance in the materials” (qtd in McGann, 2001).

## Methods and study context

We report on debugging data collected from the CAMPS project (NSF#1742257), which uses design-based research (Cobb et al, 2003) to explore intersections between art, computational thinking, and mathematics with middle-school learners. In this paper, we consider our initial uses of debugging activities, in the first iteration of Code Your Art, which we designed and ran in the first year of this project.

Code Your Art was a one-week (five-day) free summer camp for middle school students, held in the classrooms of a public middle school in a southeastern U.S. city. It consisted of two groups of 16 rising 6<sup>th</sup> -8<sup>th</sup> grade students. Each group was taught collaboratively by a pair of public-school mathematics teachers. These four teachers engaged in an intensive one-week professional development workshop before the camp, in which they were introduced to the programming environment as learners and where they collaboratively adapted an initial set of curricular materials. During the camp itself, the research team (2 professors, 4 graduate students, and 2 computer science undergraduates) provided support for the teachers, resolving any technical issues and responding to questions and other needs of the students.

We used NetLogo (Wilensky, 1999) as the programming environment for Code Your Art. NetLogo is an agent-based modeling platform, offering a powerful means to describe complex systems in nature and society (Wilensky & Rand, 2015). However, we designed our project based on the conjecture that NetLogo could *also* provide a powerful and expressive medium for creating computational art. We leveraged NetLogo’s ability to produce images from (a) the computational state of a Cartesian grid of *fixed* agents (called “patches”) and (b) the state and actions of *mobile* agents (called “turtles”). Turtles can draw on the surface of the patches, change their shape, color, or size, and move information from one location to another. To illustrate the two agent-types in action, the red region in Fig 1a involved 297 individual patches (out of the entire population of 1089) changing color to red. In Fig 1b, large arrowhead-shaped turtles of different colors draw with pens on black patches.



**Figure 1.** Patches and Turtles in NetLogo (images from the debugging activities discussed in the analysis).

Code Your Art was project oriented, with students’ work culminating in creating a visual “computational performance” using NetLogo patches and turtles. In the first three days of camp, as students’ project ideas emerged, the teachers and research team agreed that a set of debugging activities might be useful. In a meeting following the third day of the camp, we designed a group of eight activities as an addition to the curriculum for use the following morning. Our goals with these activities, based on insights about the students’ thinking from our intense observation of them, were (a) to mirror back to the learners some of the key concepts they had learned,

and (b) to solidify their familiarity with NetLogo commands and syntax as a foundation for their project work. In this paper, we focus on two of the eight activities: *Make a Stripe* (which foregrounded patches, see Fig 3) and *Draw Parallel Lines* (which foregrounded turtles, see Fig 2). We analyze four students' work across these activities (pseudonyms Ethan, Brandon, Zaair, and Brianna). These four students constitute all of the consented students in one classroom who attempted both activities.

Each of the debugging activities is framed in terms of helping a “client” to achieve a desired effect. Each consists of a pair of buttons—a setup button that returns the environment to an initial state and creates agents if necessary; and a second button whose label indicates a result the client was attempting to achieve. In each case, there is a fundamental problem with the client’s code, which the students are aiming to remedy.

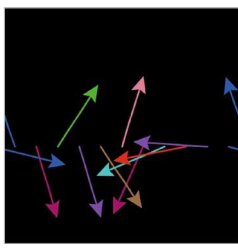
Button Name	Buggy Code	Run of Buggy Code
Set Up	ca ask patches with [pycor = 4] [sprout 1]	
Draw Parallel Lines	ask turtles [pd] repeat 15 [ask turtles [fd 1] wait .5]	

Figure 2. *Draw Parallel Lines*, the third of eight debugging challenges.


Button Name	Buggy Code	Run of Buggy Code
setup	ca	
vertical stripe	ask patches with [pycor < 20 and pxcor > 10] [set pcolor red]	

Figure 3. *Make a Stripe*, the seventh of eight debugging challenges.

We expected *Make a Stripe* to be the easier challenge for our students for several reasons. First, the camp had been working with patches for three days and with turtles for only one. Moreover, the first debugging activity in the set of eight, *White Square*, had dealt with similar concepts and was in fact more complex (involving filtering of *both* *pxcor* and *pycor*). All four of our focal students had produced a solution to that first problem (though all had required substantial help of some kind from a teacher or researcher). Thus, while a compound Boolean filter (“with [pxcor < 10 and pxcor > 20]”) is a challenging programming construct and inequalities are a challenging mathematics concept, *Make a Stripe* was a *second* exposure to these ideas in the debugging activities. Meanwhile, *Draw Parallel Lines* involved compressed-syntax commands (*pd* for “pen-down” and *fd* for “forward”) and a loop (*repeat 15*) that both addressed turtles and triggered a delay with a decimal-number argument (*wait .5*). Nevertheless, students had substantially more success with *Draw Parallel Lines* than with *Make a Stripe*. Why?

## Data collection and analysis

Throughout the camp, we collected a variety of sources of data to understand students’ experiences. These included pre-post questionnaires, interviews, classroom video from standing and mobile (GoPro) cameras, and on-computer screen-recordings. For this paper, we focus on the screen-recording data, which enabled us to take a microgenetic view of the debugging process as it unfolded.

Because we were concerned with the way students made sense of NetLogo-specific constructs in the context of expressive visual effects, we used methods of inductive coding and constant-comparative analysis (Charmaz, 2006; Strauss & Corbin, 1990), rather than applying theoretically motivated codes derived from the debugging literature. We created analytical memos (Hatch, 2002) to increase our ability to see connections across the video corpus. This exploratory work led us to identify an “edit-test pair” as a unit of debugging action and to develop conjectures about “phases” in students’ unfolding debugging work on a given problem. In the larger context of the full suite of debugging activities, we noticed the recurrence of patterns in edit-test pairs and phases

across students and activities (e.g., repeatedly changing constants and reverting the change), which suggested patterns in the relations between students and the task environments. At the same time, we were struck by fundamental differences in students' activity and affect as they worked on different problems. Indeed, we remarked more than once that students seemed "like different kids" in *Make a Stripe* and *Draw Parallel Lines*.

## Findings

In this section we describe students' work on these two debugging challenges. We begin by providing an account of Ethan's work across the two challenges as an illuminating example, and then we describe patterns in work across the student group as a whole.

Ethan begins the *Draw Parallel Lines* challenge by running the code and commenting, "They're all not even in the...even in the same way, and I don't like it, and I don't like it, and I don't like it." He then returns to a still-open NetLogo window containing his solution to a prior challenge. He spends the next several minutes talking with peers at his table-group and showing them his solution. In the course of these exchanges, Ethan demonstrates facility with NetLogo code and the environment. For example, the prior challenge required him to turn all the turtles in a model red: Ethan's companion asks if he could make them blue instead. Rather than altering the code of his solution, he enters completely new code into NetLogo's Command Center, and configures it to send a command to all turtles, writing: **turtles> set color blue**. He dramatically presses Enter and adds "Boom! Are you happy now?" He later opens each button of the model, reverts the code to its original buggy state, and reimplements his fixes while explaining them to his companion. Overall, we get a strong image here of Ethan's confidence and "fluency" with syntax and with agent-based thinking, as well as of his ability to use NetLogo code and the NetLogo environment to communicate ideas with fellow students.

Ethan then returns to *Draw Parallel Lines* saying, "I'm gonna move on to the other, third one, because the third one is probably also easy." Once he starts editing, Ethan's progress is rapid. Opening the buggy button, he immediately cuts all of the code (placing it on the clipboard) and types: **ask turtles [ set heading 90 ]** He then pastes the original code back below this new line. After setting up, he runs the new code (see Figure 4a), saying, "Boom!" He then reflects, "They're not parallel, but they're in the same place, so I'm gonna...[2 sec pause]. That looks NICE, actually." He runs this code several times, finding the visual effect interesting. Because the turtles move in steps equal to the distance between them, they can be 'read' as moving to the right *or* as changing colors in place. Ethan next arranges the buttons in the interface, seeming to invest in the appearance of his solution.

A teacher passes by and asks "What should we do to those arrows [the turtles] to make them vertical?" Ethan responds immediately by opening the button and changing "90" to "180," which produces Fig 4b. The teacher celebrates Ethan's success with him and then moves on to other students. At this point, Ethan extends his solution to explore what he can make with it. He says "I made a cage door!" Next, he decides to create a "forever" performance integrating the horizontal and vertical movements he has programmed. By running his buttons one after the other, he creates a symmetric pattern on the screen (Fig 4c). He celebrates: "Yes! Yes! I've made a work of art." After reflecting for a moment, he says "I'm going to turn the third one into my own work of art. Wait, I already did that, so I'm just gonna do a setup button. He creates a "Set Up 2" button to contain the code "**ask patches [ set pcolor white ]**" and then edits it to "**ask patches [ set pcolor red + 3 ]**," using his knowledge of the NetLogo color space to create a shade of pink (Fig 4d). He then calls his peers over, "Look at this work of art I've made....out of the third one." He ends with "I'm going to save this" before closing the model.

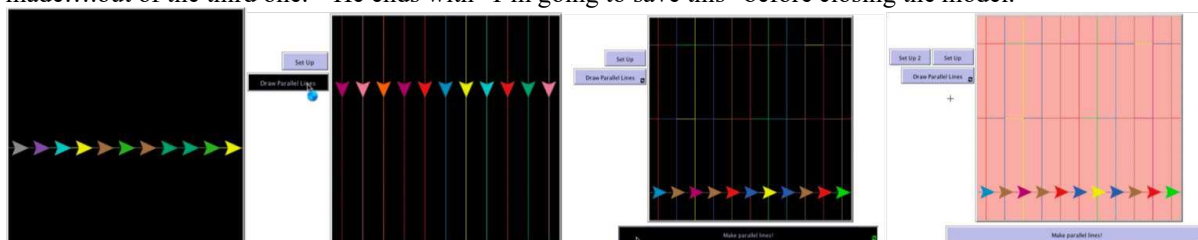


Figure 4. Ethan's progressive solutions and extensions to *Draw Parallel Lines*.

In contrast, Ethan begins *Make a Stripe* with an unsuccessful sequence of rapid alterations to the code: what we have described in our data as "tweaks." Below, the highlighted characters show his changes (cf Fig 5):

- \* ask patches with [pycor < 20 and pxcor > 10] [set pcolor red]
- 1 ask patches with [pxcor < 20 and pxcor > 20] [set pcolor red]
- 2 ask patches with [pxcor < 20 and pycor > 20] [set pcolor red]
- 3 ask patches with [pycor < 20 and pxcor > 20] [set pcolor red]

- 4 ask patches with [pycor > 20 and pxcor > 20] [set pcolor red]
- 5 ask patches with [pycor > 20 and pxcor < 20] [set pcolor red]
- 6 ask patches with [pycor = 20 and pxcor = 20] [set pcolor red]
- 7 ask patches with [pycor = 20 and pxcor < 20] [set pcolor red]

Change #6 creates a single red patch on the screen (at coordinates (20, 20)). Seeing this, Ethan audibly expresses frustration for the first time in this activity (“mmm!”). After Change #7 the button turns red (indicating a syntax error). Ethan closes NetLogo, opens another copy of the model, goes to the Command Center, and types:

```
observer> ask patches [ if pcolor = black [set pcolor white]]
```

Given how work in the Command Center figured into his peer interactions above, we speculate he does this as a way to re-establish confidence. He then opens yet another copy of NetLogo and says to himself, “I wanna give up SO bad!” When a researcher comes up to him, he says “I tried like everything but it never made a stripe.”

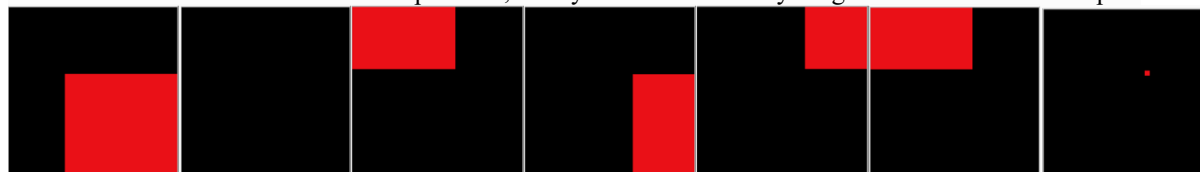


Figure 5. Results of Ethan’s “tweaks” corresponding to the start state and to Changes 1-6, above.

The researcher asks Ethan which patches he *wants* to turn red; then over the next 3 minutes he guides Ethan through selecting a series of strategically selected patches from the grid and testing whether each of them would turn red based on code of the button and whether they *should* turn red, based on the client’s goal. Responding to this structured questioning, Ethan identifies and fixes the error. He responds to the researcher’s encouraging “Awesome, right?” with silent assent. Though he does run the solution several times after the researcher leaves, he does not show signs of positive affect comparable to his response to *Draw Parallel Lines*. While working on *Make a Stripe*, Ethan does not engage with his peers or sing to himself as he had done earlier. His code changes do not suggest the confident fluency that he evinced during *Draw Parallel Lines*. He responds to the researcher’s questions and solves the problem, but it does not appear to give him joy. And the difference across activities extends to problem-solving strategies as well as affect and social interaction. Indeed, in the categories described in Perkins et al (1986), Ethan would present as a “mover” in *Draw Parallel Lines* and as a “tinkerer” and/or “stopper” in *Make a Stripe*. These contrasts in his behavior and affect are consistent with a general trend across the group of focal students we observed in these two activities.

### Broader picture: Responses of our four focal students overall

All four students solved the *Draw Parallel Lines* problem. Three received some intervention from an adult: two received support on syntax and one received encouragement to improve an initial solution (Ethan). All four students gave spontaneous signs of positive affect or pride in their solution (see Fig 6). Three of the four extended their solution to create additional visual effects.



Figure 6. Celebrations: Brandon’s happy dance. Zair: “Yes! I did it: I am so smart! I did it.” Brianna “Yay!”

In addition to celebrating their successes, all four of the students produced work on *Draw Parallel Lines* that went beyond a simple solution. Brandon, in partnership with a researcher, explored different angles and altered the setup to make the turtles start on a vertical line (Fig 7a). Zair, though he did not extend his solution, made his turtles draw squares *before* settling on parallel lines (Fig 7b). And Brianna worked on her own to re-enact and modify code changes that a researcher had supported her in creating (Fig 7c).

In contrast, only two of the four focal students solved the *Make a Stripe* problem. Both received substantial intervention from an adult to do so, while an additional student rejected help. One intervention (with Ethan) involved a strongly-guided exploration of coordinates of patches, while the other intervention consisted of



leading questions to support the student in decoding the buggy line of code. One student (Ethan) expressed negative affect; one expressed mixed positive and negative affect; and one expressed a muted positive reaction to a teacher’s congratulations on solving the problem. Neither of the two students who solved *Make a Stripe* extended their solution or added visual effects, though one student who did not solve it for lack of time attempted to use turtles to create a solution that we did not anticipate.



Figure 7. Extensions: Brandon changes starting state and explores angles. Zaaair draws squares. Brianna fills the screen by setting the turtles’ heading to 295.

## Discussion and conclusion

How can we explain the stark contrast in students’ success, strategies, and affective responses across the two debugging activities? As mentioned above, we expected the *reverse*, based on our sense of the difficulty of the syntax and programming structures involved and because turtles were new to the students while patches were more familiar. We can generate conjectures about causes. For instance, we have considered that the mathematics of inequalities (and even the symbols) might have been opaque enough to cause *Make a Stripe* to be inaccessible, even after *White Square*. We have also considered the accessibility of turtles versus patches. Papert (1980) described the *syntonicity* of turtles—that they encourage learners to project into them—and this affordance may have overridden the students’ longer exposure to patches. Both activities involved reasoning about agent-sets (rather than single agents), but the number of patches that needed to be coordinated in *Make a Stripe* was an order of magnitude greater than the size of the turtle-set in *Draw Parallel Lines*. Moreover, patches are less individually visible or visually distinguishable than turtles, and so the inaccessibility of *Make a Stripe* could connect with difficulties with aggregate reasoning.

These conjectures all have some plausibility and each likely offers a partial explanation. However, all of these conjectures point to the interpretation that what was going on in these activities was a form of *mediated debugging*. We have no explanation for our data that is based purely on increasing expertise or on a change in their mastery of situation-independent debugging practices. This is not to say that learners do not develop debugging practices and strategies: rather, it suggests that for our beginning NetLogo learners, something about their sense of the activity mediated their work more strongly than these factors.

As a clue to factors that mediated the character of the focal students’ debugging, we note that, in the two activities discussed above and throughout the debugging set, students demonstrated a higher level of knowledge and confidence in the activities they were able to take up in a spirit of artistic expressivity. Though their work to date within the camp had focused on patches, the data above show that this prior experience did not prevent them from taking up turtle-based debugging activities with creativity and enthusiasm. And conversely, even though these students used patches in expressive ways both before the debugging activities and in their final projects afterwards, it was still possible for a patch-based debugging activity such as *Make a Stripe* to “fall flat” for them.

These findings are highly relevant to our goals in the Code Your Art camp, and they have informed refinements to our design that we have undertaken this past year. In ongoing work, we are aiming to integrate debugging tasks with invitations to modify, extend, or remix the “client’s” code. In this way, we intend to encourage creative responses that demonstrate not only the ability to “fix” the code but also to manipulate its effects, exploring the space of where a given visual effect embodied in code *could* be taken. We argue that better understanding how debugging (or debugging-and-extending) tasks can invite and support different kinds of participation is an important contribution to the understanding of debugging competencies among learners. Our own future work will continue to explore these ideas by examining students’ debugging behavior under different mediations and refined forms of the design principles we are developing, across iterations of Code Your Art.

## References

Bednarik, R. (2012). Expertise-dependent visual attention strategies develop over time during debugging with multiple code representations. *International Journal of Human-Computer Studies*, 70(2), 143–155.

- Blackwell, A., McLean, A., Noble, J., & Rohrerhuber, J. (2014). Collaboration and learning through live coding (Dagstuhl Seminar 13382). *Dagstuhl Reports* 3(9), Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik
- Bontá, P., Papert, A., & Silverman, B. (2010). Turtle, art, turtleart. In *Proc. of Constructionism 2010 Conference*.
- Charmaz, K. (2006). *Constructing grounded theory: A practical guide through qualitative analysis*. Thousand Oaks, CA: Sage.
- Cobb, P., Confrey, J., diSessa, A., Lehrer, R., & Schauble, L. (2003). Design Experiments in Educational Research. *Educational Researcher*, 32(1), 9–13.
- Forester, E. M. (1970). *Aspects of the Novel*. New York: Penguin Books.
- Gould, J., & Drongowski, P. (1974). An exploratory study of computer program debugging. *Human Factors*, 16, 258–277.
- Greeno, J. (1994). Gibson's affordances. *Psychological Review* 101(2), 336–342.
- Hall, R., & Jurow, A. S. (2015). Changing concepts in activity: Descriptive and design studies of consequential learning in conceptual practices. *Educational Psychologist*, 50(3), 173–189.
- Hatch, J. A. (2002). *Doing qualitative research in education settings*. Albany, NY: SUNY Press.
- Lui, D., Anderson, E., Kafai, Y. B., & Jayathirtha, G. (2017, October). Learning by fixing and designing problems: A reconstruction kit for debugging e-textiles. In *Proceedings of the 7th Annual Conference on Creativity and Fabrication in Education* (p. 6). ACM.
- McGann, J. (2001). The rationale of hypertext. In *Radiant Textuality* (pp. 53–74). New York: Palgrave.
- Kafai, Y. B. (2016). From computational thinking to computational participation in K–12 education. *Communications of the ACM*, 59(8), 26–27.
- Lave, J., & Wenger, E. (1991). *Situated learning: Legitimate peripheral participation*. Cambridge Univ. Press.
- Lin, Y., Wu, C., Hou, T., Lin, Y., Yang, F., & Chang, C. (2016). Tracking Students' Cognitive Processes During Program Debugging—An Eye-Movement Approach. *IEEE Transactions on Education*, 59(3), 175–186.
- Nolen, S. B., Horn, I. S., & Ward, C. J. (2015). Situating motivation. *Educational Psychologist*, 50(3), 234–247.
- Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc.
- Papert, S., & Harel, I. (1991). *Situating constructionism*. *Constructionism*, 36(2), 1–11.
- Perkins, D. N., Hancock, C., Hobbs, R., Martin, F., & Simmons, R. (1986). Conditions of learning in novice programmers. *Journal of Educational Computing Research*, 2(1), 37–55.
- Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., ... & Kafai, Y. B. (2009). Scratch: Programming for all. *Communications of the ACM*, 52(11), 60–67.
- Rich, K. M., Strickland, C., Binkowski, T. A., & Franklin, D. (2019). A K-8 Debugging Learning Trajectory Derived from Research Literature. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 745–751. <https://doi.org/10.1145/3287324.3287396>
- Rich, K. M., Strickland, C., Binkowski, T. A., Moran, C., & Franklin, D. (2017, August). K-8 learning trajectories derived from research literature: Sequence, repetition, conditionals. In *Proceedings of the 2017 ACM conference on international computing education research* (pp. 182–190). ACM.
- Rich, K. M., Binkowski, T. A., Strickland, C., & Franklin, D. (2018, August). Decomposition: A K-8 Computational Thinking Learning Trajectory. In *Proceedings of the 2018 ACM Conference on International Computing Education Research* (pp. 124–132). ACM.
- Robins, A. (2019) Novice Programmers and Introductory Programming. In S. Fincher & A. Robins (Eds.). *The Cambridge Handbook of Computing Education Research*. (pp. 327–376). Cambridge, UK: Cambridge University Press.
- Strauss, A., & Corbin, J. (1990). *Basics of qualitative research* (Vol. 15). Newbury Park, CA: Sage.
- Turkle, S., & Papert, S. (1992). Epistemological pluralism and the revaluation of the concrete. *Journal of Mathematical Behavior*, 11(1), 3–33.
- Vessey, I. (1985). Expertise in debugging computer programs: A process analysis. *International Journal of Man–Machine Studies*, 23, 459–494.
- Vygotsky, L. S. (1978). *Mind in society: The development of higher psychological processes*. Cambridge, MA: Harvard U Press.
- Wertsch, J. V. (1994). The primacy of mediated action in sociocultural studies. *Mind, culture, and activity*, 1(4), 202–208.
- Wilensky, U. (1999). NetLogo [Computer Software]. Evanston, IL: Northwestern U.
- Wilensky, U., & Rand, W. (2015). *An introduction to agent-based modeling. Modeling Natural, Social, and Engineered Complex Systems with NetLogo*.

## Acknowledgments

This work is supported by the National Science Foundation under Grant No. 1742257.