The Importance of Students' Attention to Program State: A Case Study of Debugging Behavior

Colleen M. Lewis
Graduate School of Education
University of California, Berkeley
Berkeley, CA 94720
001-510-388-7215

ABSTRACT
To develop a model of students' debugging processes, I conducted a qualitative analysis of young students engaged in debugging computer programs they had written in the programming language Scratch. I present a microgenetic analysis that tracks how one student's attention to elements of computer program state shifted during his debugging process. I present evidence that this student had relevant domain knowledge and claim that his changing attention within the problem, and not his domain knowledge, mediated his debugging process. I hypothesize that a key competence in debugging is learning to identify what elements of program state are important to pay attention to and that this attention, and not only domain knowledge, mediates the debugging process. This hypothesis is consistent with a model of physics reasoning and learning from the Knowledge in Pieces theoretical framework and in this research I build upon education research outside of computer science. The case study analyzes the debugging process of a student entering the sixth grade, but I document an isomorphic case from a pair of college students to show that this pattern extends beyond this age.

Categories and Subject Descriptors
K.3 [Computers and Education]: General

General Terms
Human factors

Keywords
Debugging, programming state, Scratch, case study

1. INTRODUCTION
There are decades of research investigating difficulties experienced by novice programmers [2, 12, 21, 27]. While computer science education research typically identifies learning difficulties at the grain size of individual misconceptions [2, 16, 17, 27], I observed that difficulties with state and state change operations underlie many of the difficulties experienced by middle-school students engaged in programming and debugging.

ICER'12, September 9–11, 2012, Auckland, New Zealand.

State represents the idea, present in all programming environments, of a set of temporary or permanent variables that completely describe the current environment on which a program can act. This includes programmer-defined variables as well as other aspects of the runtime environment such as the current stack frame. Program commands change aspects of the computer program's state and the process of writing programs involves developing sequences of state change operations to achieve a particular goal. How a program changes state can be seen as a more formal definition of the broad notion of what a program "does." It is therefore essential that students recognize how state-change operations may act on a hypothetical state and that students have knowledge and techniques with which to track the current computer program's state.

I present a case study [29] from a qualitative study of students' debugging behavior from a summer enrichment program for students entering the sixth grade. This case study is used to illustrate the hypothesis that a key competence in debugging is learning to identify what elements of program state are important to pay attention to and that this attention, and not only domain knowledge, mediates the debugging process.

In this case study, a student experienced difficulty debugging a programming that involved angles and position in the programming language Scratch. However, once he attended to the relevant element of state, he demonstrated his competence with angles and positions and quickly rewrote his program to modify state correctly. His competence with angles and position challenges the alternative hypothesis that he had insufficient knowledge or a specific misconception regarding angles [14]. I explain his difficulties as a lack of attention to relevant elements of state and not simply a lack of knowledge.

Extrapolating the pattern from the case study, I propose a model of student debugging that depends upon the student's attention within the problem as the determining factor in the debugging process. This model is consistent with a model of physics reasoning and learning from the Knowledge in Pieces theoretical framework. The case study is intended to illustrate details of this model, but is not intended to establish the prevalence of this pattern. However, to show that the pattern from the case study is not unique to young students, I provide an example of a bug encountered by college students that is isomorphic to the bug the student addressed in the case study.

This paper has three primary contributions. First, this paper provides a microgenetic analysis [28] of naturally occurring debugging behavior. The second contribution is the proposed model of debugging behavior as mediated by students' attention to program state. The third contribution is that this model, which

focuses on students' attention, creates a connection between computer science education and education research within the Knowledge in Pieces theoretical framework.

2. PREVIOUS RESEARCH

Based upon the framing of programming as developing sequences of state change operations, an understanding of state and state change operations is essential to successful programming. du Boulay and his colleagues [11, 12] developed this claim through a focus on what they call the notional machine, which is essentially a functional description of how program state can be changed. du Boulay and his colleagues argue that students need a firm understanding of these properties of the machine to be successful writing programs and that these properties should be made explicit to students. More recently, Ben-Ari [1] reiterated the importance of students' understanding of the notional machine and critiqued object-oriented programming languages for obscuring aspects of the machine.

The researchers involved in the design and evaluation of the Alice programming language also emphasize the importance of program state, particularly for understanding and debugging code [4, 6, 22]. "We believe the source of confusion in figuring out what went wrong, in all but the most trivial code, is an inadequate understanding of the program's state." [4, p. 109].

Sajaniemi and Kuittinen [23] have attempted to help students recognize common patterns of state change operations. Their work highlights the roles of variables in programs. They claim that 10 roles account for 99% of all variable roles used in introductory programming texts. In a later study [24], they asked students to represent the state of an object-oriented program at a moment in time. In this work they focus on students' perception of what is relevant to state and how students' perception changes during a programming course. They use what details a student represents as an indication of what aspects of state that student believes to be important. They track how the details that a student represents changes during a programming course.

3. THEORETICAL FRAMEWORK

I preface the analysis with a brief overview of the Knowledge in Pieces theoretical framework [7, 8, 9]. This overview serves to articulate assumptions about the nature of knowledge that guide the research methods and analysis.

Research that adopts the Knowledge in Pieces theoretical framework attempts to build and refine models of knowledge that describe the dynamic process of human reasoning. For example, researchers have developed models of a type of intuitive knowledge [7] as well as types of conceptual knowledge and how individuals apply them [8, 9].

Instead of assuming that individuals have rigid mental models, the Knowledge in Pieces theoretical framework models a student's use of knowledge as a complex emergent process. In a given situation, students' are assumed to make use of a subset of their potentially relevant knowledge, shaped by their perception of what knowledge is relevant and their attention within the context. An assumed challenge of learning is to consistently use the most appropriate knowledge in a given context [8, 9].

The Knowledge in Pieces theoretical framework specifies that a wide variety of subtle changes in a problem may change what an individual attends to within a problem and what knowledge an individual applies in that context [8, 9]. What an individual attends to within a context is both shaped by the individual's knowledge and mediates what knowledge that individual uses.

4. RESEARCH QUESTION

In this study, I address the following research question: How does a student's knowledge and changing attention within a problem shape the process of debugging?

5. METHODS

There are a number of challenges in studying students' debugging behavior. Observing students debugging their own buggy code does not provide any consistency across research participants because the bugs they identify and fix will be unique. However, observing students debugging uniform bugs in code they did not write may be an unfamiliar experience for students and not representative of their behavior debugging their own code. The methods used in this study prioritized observing natural debugging behavior rather than documenting behavior that could easily be compared across research participants.

Data was collected and analyzed using methods of microgenetic analysis [28] and informed by other qualitative methods [7, 8, 9, 13, 15, 25] described below, which are commonly used in conjunction with the Knowledge in Pieces theoretical framework.

5.1 Research Context

With students' and parents' consent, data was collected from two computer programming courses in a summer enrichment program for students entering the sixth grade. There were 50 students in total, all of whom had been accepted to the enrichment program designed for academically advanced children. Few of the students had any computer programming experience, but no other information was collected regarding students' academic background.

During the thirty-six hour course, students used pair programming [3] and learned to program in Logo and Scratch. Related studies have extensively documented this context [18, 19, 20] and therefore additional details are omitted here.

5.2 Data Collection

The data collection was designed to capture the behavior of students engaged in programming and debugging in Logo and Scratch. During class, the computers used by the students recorded a video of the computer screen and audio from a microphone attached to the computer. These recordings could be replayed for analysis and showed students' actions in the programming environment and simultaneous discussion. Three classroom video cameras could be used to disambiguate which student was acting as the driver and navigator within each pair.

Methods of data collection in the Knowledge in Pieces line of work consistently focus on gathering this type of process data frequently involving think-aloud protocols [26] or clinical interviews [10]. In the current study, conversations between pairs served a similar function as clinical interviews, encouraging students to elaborate and discuss their thinking.

5.3 Case Selection

The goal of the study was to examine individual cases of students engaged in debugging. With approximately 900 hours of screen recording data, the analysis began by narrowing consideration to a particularly challenging element of the curriculum, described in Section 5.4. I watched these videos and developed brief, time-stamped summaries of a selection of the video data, which are referred to as content logs [13]. From these content logs I identified recurring patterns of student behavior in the video data. I selected a particular case that was similar to the other cases in many ways except that the student spoke more frequently while he worked.

This case was also selected because the student acknowledged a discrepancy between the behavior of the program and the intended behavior of the program. There were many other instances in which the researcher identified a student as engaged in debugging. However, many of these cases were ambiguous because it was not clear if the student recognized the discrepancy or was taking actions to resolve the discrepancy. In the selected case I could more accurately characterize the student's behavior as debugging because he made reference to the discrepancy and was attempting to find a solution.

5.4 Data Analysis

Conducting qualitative research requires sensitivity to the data and does not typically follow a linear path of data analysis [5]. The analysis involved a grounded approach [5] to develop hypotheses about the students' patterns of debugging. After the case was selected, the video was transcribed and watched and re-watched. From segments of this video, I developed hypotheses about his debugging behavior and sought to generalize and validate the hypotheses by considering the full case [13]. This was an iterative process where hypotheses continued to be refined. The video and my formative hypotheses were presented on multiple occasions to groups of educational researchers who critiqued these hypotheses and offered alternative interpretations.

In the analysis I attempt to provide a systematic account of the student's debugging behavior. Building upon the Knowledge in Pieces theoretical framework [7, 8, 9], I focus on students' attention across the evolving context of the interview and students' resulting patterns of reasoning.

In the analysis of this individual case I used analysis methods from the Knowledge in Pieces theoretical framework [7, 8, 9], which are akin to methods of microgenetic analysis [28] and other detailed studies of learning [25]. In presenting these data I attempted to provide the reader with enough data to evaluate the analysis and hypotheses presented.

Case studies like this one are not intended to prove that a particular pattern of behavior exists within a population. Instead, the data serve to inform and exemplify hypotheses regarding features of learning within a domain.

5.5 Problem Description and Solution

The case study is taken from a video of a student debugging a program written in the programming language Scratch. Scratch, the problem, and a description of a solution to the problem are provided as necessary background for the case study.

In the programming language Scratch, students can modify state by moving a character around a 2D screen. For example, the Scratch command "move 10 steps" will move the character 10 pixels in the direction that it is currently facing. This effectively modifies the x- and/or y-location aspect of the character's state. Other commands in Scratch modify the current state relating to whether the character draws a line tracing its path when it moves. This aspect of state is determined by whether the commands "pen down" or "pen up" have been executed.

Students can use these commands to draw complex images in Scratch. For example, in the case study students were attempting to draw the image shown in Figure 1.

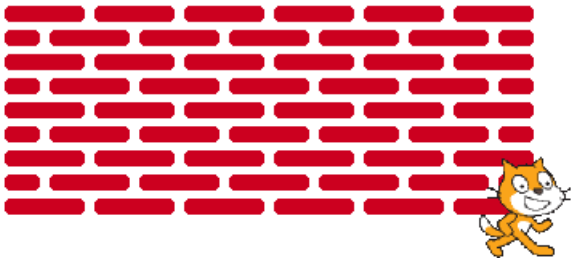

Figure 1: Representation of the goal of the program.

The bricks are created by moving the character forward while the pen is down. The spaces between the bricks are created by moving the character while the pen is up. For example, the script in Figure 2 shows how to draw two bricks that are forty pixels wide, separated by a fifteen pixel space.

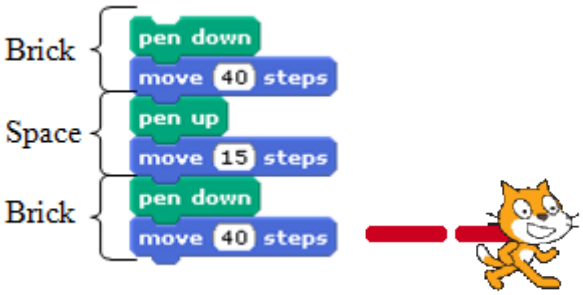

Figure 2 – Script to draw two bricks (left) and the resulting bricks and position of the character (right)

Drawing the entire brick wall involves creating scripts that draw each of the distinct rows. The first row and all odd numbered rows contain six full bricks (Figure 3). All even numbered rows contain five full bricks and a half brick on each end (Figure 4).

Figure 3: Pattern of bricks for all odd numbered rows

Figure 4: Pattern of bricks for all even numbered rows

To draw an entire brick wall, the student must direct the character to navigate between each row. For example, assuming that the character begins at the top left, the program begins by drawing the top row from left to right. The student must then direct the character to turn and move to a lower y-position to draw the second row. The script shown on the top-right of Figure 5 shows the process of turning right, moving down and turning right again to begin the second row, which also moves from an odd numbered row to an even numbered row. Once the character traces the second row, they must turn left, move down and turn left again to

begin the third row. The script shown on the bottom-left of Figure 5 shows this process of turning between the 2nd and 3rd rows, which also moves from an even numbered row to an odd numbered row.

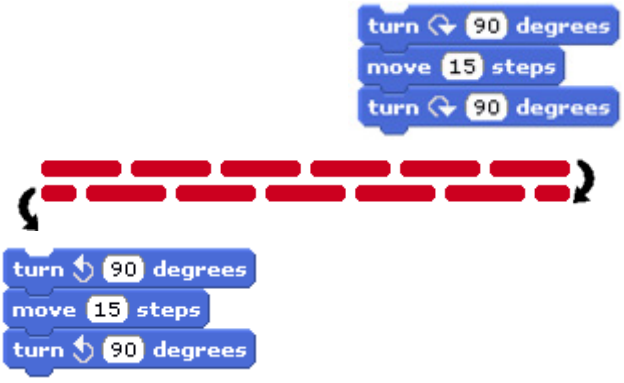

Figure 5 - Script to move from an odd numbered row to an even numbered row (shown on the top-right) and the script to move from an even numbered row to an odd numbered row (shown on the bottom-left) with arrows indicating the direction to turn between rows of bricks

This process of drawing the first two rows of bricks is repeated for each subsequent pair of rows to draw the entire wall.

6. CASE STUDY

The data and analysis are divided into four sequential episodes. Each episode is prefaced with a summary of the episode and followed by an analysis of the episode. In the analysis, I track the students' attention within the programming environment. His actions cause changes in what visual information is available and thereby changes what he pays attention to.

6.1 Excerpt One

6.1.1 Summary

The case study begins after two students had created programs to draw two distinct patterns for the rows of bricks. However, they were unsuccessful at drawing a brick wall because they did not have a way to move between rows such as with the method outlined in Section 5.4. They eventually created the script shown in Figure 6, which can move their character from the first row to the second row, but they incorrectly assumed that it could also move the character from the second to the third row.

Figure 5 shows arrows indicating the correct direction of movement to navigate between rows: turning right after the first row and left after the second row. However, the students used the script shown in Figure 6 to turn right after both the first and second rows. Therefore when they attempted to draw the third row, it retraced over the top row. The result of this is shown in Figure 7.

Figure 6 – Students' script to turn between two rows of the brick wall

Figure 7 – Two rows of bricks with arrows indicating the incorrect character movement between rows

While the students were working in pairs, the excerpt below features only one student. The classroom video indicates that during this time his partner was not paying attention to the work he was doing within Scratch. In the excerpt below, one of the students, Kevin (a pseudonym), had just finished adjusting the vertical distance between the rows of bricks specified by the number of steps moved in the script in Figure 6. He began with 10, then tried 20, and finally in the beginning of the transcript shown below, settled on 15. In the following transcript, Kevin traced the top row 3 times and did not show any indication that he observed the retracing of the first row or had identified why a third line was not drawn.

The following convention is used in the transcripts:

- "executed odd row" means that the script to draw the pattern of bricks for odd numbered rows was executed
- "executed even row" means that the script to draw pattern of bricks for even numbered rows was executed
- "executed turn" means that the script shown in Figure 6 to turn right between the rows was executed

6.1.2 Data

01 Kevin: Fifteen (Modified the move in Figure 6 to be 15 steps) I think I've got it.
02 Kevin: (Executed odd row, executes turn, executes even row, executed turn, executed odd row, which retraced the top row)
03 Kevin: What? Wait. (Cleared the screen)
04 Kevin: Okay (Executed odd row) – that (executed turn), that (executed even row), that (executed turn), that (executed odd row, which retraced the top row)
05 Kevin: (Cleared the screen)
06 Kevin: (Without speaking, executed odd row, executed turn, executed even row, executed turn, executed odd row, which retraced the top row.)
07 Kevin: Oh my god (Cleared the screen)
08 Kevin: Seventeen. (Modified the steps in Figure 6 to be 17 instead of 15.)

6.1.3 Analysis

Kevin retraced the top row three times and appeared unsure why a third line was not drawn. From a purely perceptual stand point, it is possible on the screen to observe the character's vertical location as on the top row. This problem could have been identified by Kevin by attending to the y-position of the character before beginning the second turn.

It appears from his statements of "What? Wait." (line 03) and "Oh my god" (line 07) that he had identified a discrepancy between what he wanted to happen and the actual result of the code, but without attending to the position of the character he seemed to identify only that a third row was not drawn, but not that the first row was retraced.

The modification Kevin made on line 08 was to change the spacing between the rows. Based upon his earlier seemingly purposeful tinkering with the spacing between the rows, I

hypothesize that he did not believe this would solve the problem. However, it is ambiguous what motivated his action in line 08.

Kevin had the perceptual skills to identify the position of the character and later it will be apparent that Kevin had the required skills within Scratch to create the script to move the character to the correct orientation and position. The issue is therefore a matter of attention and needing to refine his knowledge for the purpose of paying attention to the correct variables of state in this context.

In summary, I hypothesize that Kevin was not able to isolate the discrepancy in the first excerpt because he paid insufficient perceptual attention to the position and direction of the character.

6.2 Excerpt Two

6.2.1 Summary

In excerpt two, Kevin accidentally drew a solid second row, as shown in Figure 8, which resulted from superimposing an odd and even row. Kevin had drawn row one and row two, leaving the cat facing left at the far left side of the bottom row. By turning the character right 90 degrees 6 times, the character turned 180 degrees to face right on the second row. When he then executed the script for the odd row it created a solid line.

6.2.2 Data

09 Kevin: (Without speaking executed odd row, executed turn, executed even row, and then separated the turning script from Figure 6. He executed a turn right 90 degrees command 6 times, which turned the cat to face back to the right on the second row. Next he executed odd row, which created the image shown in Figure 8 by retracing the second row with the odd row script)

10 Kevin: What? Oh my god – why isn't it working? (Cleared the screen and reconstructed the turn script shown in Figure 6)

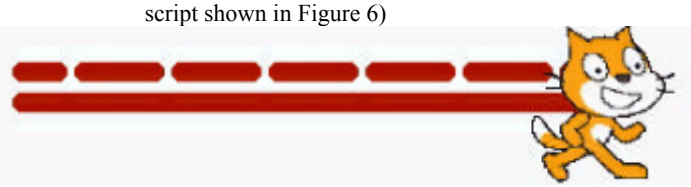

Figure 8 - Result of tracing over the second line with the script to draw the top line

6.2.3 Analysis

In excerpt two, Kevin retraced the second row with the odd row script. By experimenting with the rotation of the character, he appeared to be appropriately attending to the direction of the character. However, for a complete understanding of the bug he also needed to attend to the position of the character.

From Kevin's statements "why isn't it working?" (line 10) I assume that he identified a problem, but had not identified the cause. Kevin had the perceptual skills and experience in Scratch to identify that the vertical position of the character is unchanged by turning the character right 90 degrees 6 times; he was simply not attending to the y-position of the character at this time so as to identify the y-position before he executed the turn blocks.

While attending to the relevant program, the turn script (shown in Figure 6), Kevin had not begun to identify the cause of why the third row was not drawn nor the cause of the discrepancy shown in Figure 8. This is evidence of inappropriate attention to the state of the character. As a possible example of the context dependent nature of knowledge, Kevin had relevant knowledge that he appeared to not apply in this context.

6.3 Excerpt Three

6.3.1 Summary

In excerpt three, Kevin executed the script to draw the first two rows and retraced the first and second row before accidentally tracing over the first row with a copy of the second row.

6.3.2 Data

11 Kevin: (Executed odd row, executed turn, executed even row) It works there.

12 Kevin: (Executed turn, executed odd row, which retraced the top row. Executed turn, executed even row, which retraced the second row. Executed turn, executed even row, which traced over row 1 with row 2)

13 Kevin: Oh. (Cleared the screen)

Figure 9 - Result of tracing over the row 1 with row 2

6.3.3 Analysis

The previous times that Kevin retraced the top row (lines 01 to 08); it appeared simply as if nothing was drawn. In those cases, he could have analyzed the script or attended to the character's position and direction to infer that the character retraced the top row. The fact that Kevin did not appear to have recognized that he was retracing the top row in those instances suggests that he was not attending to the relevant aspect of state, the character's location and direction.

By tracing over the top row with the even row script, Kevin created a solid line. There was then an indication that he has traced over the top row, present in this visual discrepancy shown in Figure 9. It is from this discrepancy that he appeared to orient to the relevant state and ultimately create the necessary program.

In summary, the visual discrepancy presumably made Kevin pay attention to the final location of the character, which indicated that the top line had just been retraced. This redirection of his attention shifted what knowledge he applied, which was ultimately productive as shown in the next excerpt.

6.4 Excerpt Four

6.4.1 Summary

In excerpt four, immediately after responding to the discrepancy with "oh", Kevin redrew rows 1 and 2 and executed the turn script from Figure 6 twice. From there he quickly constructed the opposite turn script to turn the character between even and odd numbered rows.

6.4.2 Data

14 Kevin: (Executed odd row, executed turn, executed even row, executed turn, and executed turn a second time. This left the cat in the same position as immediately following row 2.)

15 Kevin: Oh I see now. (Cleared the screen. Constructed a new turn script that mimics the current turn script from Figure 6, turning left as opposed to right.)

6.4.3 Analysis

In this excerpt Kevin drew the first two rows and then ran the turn script twice. Based upon his previous use of the program, he might have believed that executing the script twice would cause the character to move lower on the screen. However, executing the turn script twice appeared to highlight the location of the character for Kevin. After this change in attention, Kevin created the relevant script to turn left between rows and was able to navigate between the rows in the brick wall to draw a series of rows.

Kevin's facile creation of the second turn script suggests a level of competence in manipulating state that was not demonstrated by his difficulty in identifying the problem. While it may be possible to classify Kevin's previous actions as evidence of a misconception, such as a misunderstanding of turns [14], I argue that Kevin was developing his concept of state within the programming context by learning to pay attention to the appropriate variables of state.

7. EXTENSIBILITY TO COLLEGE STUDENTS

An example of a similar problem was experienced by a pair of students working on a final project in Scratch at the University of California, Berkeley. The students emailed me regarding a problem they were having. The bug was from within their final project after roughly 50 hours of programming experience in Scratch.

This example is not analyzed in depth because the data available was email correspondence and not process data. However this example demonstrates that problems of attention to state are not limited to a single age group.

In the students' email they described a problem that if they stopped the execution of their program in the middle of drawing and started it again from the beginning, that it did not draw in the same spot. This happened because their program only set the position of the character and not the orientation. Therefore, when the program was run with a different initial orientation, it drew in a different location. The students believed this to be a problem in Scratch, and not their code. They wrote: "If you run him again he will draw in a slightly different location, despite the fact he is being sent to coordinates in theory."

The program shown in Figure 10 draws a column of 7 circles. This is a slightly modified version of the students' original program, so as to more clearly demonstrate the problem with state. The original program changed the pen color to draw two-toned circles and the code for this was removed.

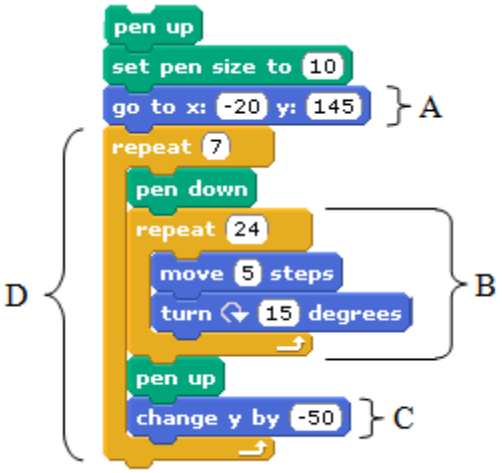

Figure 10 - Code to draw 7 circles (A) Specifies the start location (B) Draws a single circle (C) Lowers the y-position for the next circle and (D) Draws 7 copies of the circle each spaced 50 pixels apart.

The left of Figure 11 shows an incomplete execution of the code where it is stopped before it completed the 3rd circle. The image on the right in Figure 11 shows a subsequent execution of the code, with the pen color changed to make each execution more visible. From this image we can verify the issue described by the students, that a subsequent execution draws the circles in a different location.

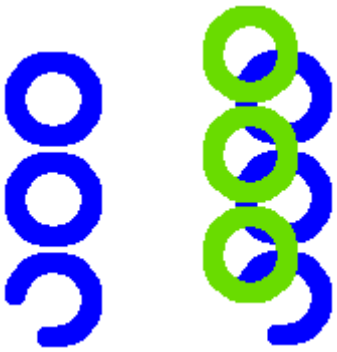

Figure 11 - A partially executed version of the code (left) followed by a second execution of the code that causes the circles to be misaligned (right)

The students here were not considering that an essential element of the position of the circle was not only the starting x- and y-position but also the direction of the character. For example, in Figure 12 we show the result of drawing two copies of a circle starting from the same x- and y-position. The first circle, in blue, began tracing the circle with the character facing the right, shown by the red arrow. The second circle, in green, began tracing the circle with the character facing left, shown by the yellow arrow. Both circles began at the same coordinate position, denoted with an X, however the character's direction before drawing the circle caused the positions of each circle to be different. When the students would stop their script before it had completed, the character would remain with its previous heading/direction. Therefore, the resulting circles would not match the previous

execution because the initial state of the character was different before the two executions.

Figure 12 - Annotated image of drawing multiple circles starting with different initial directions

The students assumed that this was a problem with Scratch rather than their program. They came to this conclusion because they were not attending to the relevant variable of state. The students appear sensitive to state to some extent by resetting the position of state (the x and y location) within the program. However they failed to identify all variables of state that were necessary to initialize to ensure consistent behavior.

The direction that the character is facing is by default represented visually by the character. However, they had selected an option to not display the direction of the character on the screen, which made this state less apparent.

When the students realized the problem of the direction of the character they reported feeling "sheepish". This example highlights an anecdotally observed pattern of reaction to identifying the relevant aspect of state. After attending to the appropriate aspect of state, the solution appears obvious. Unfortunately, it would be easy for an instructor or student to dismiss this as a simple "mistake." However, this "mistake" may better be classified as a problem of state.

This anecdote suggests that problems of state can extend across age groups and shows a second example where students' inattention to particular elements of state and not their content knowledge of the state change operations determined their success debugging. Given that the data contained limited information regarding the students' thinking, a full analysis using rigorous methods, such as in the case study presented, was not conducted.

8. CONCLUSIONS

This work sits within a larger agenda of connecting educational research and educational theory from science and mathematics to understand the learning of computer science. In this work I attempt to strengthen ties between computer science education and science education research by applying the Knowledge in Pieces theoretical framework. This framework provided epistemological and methodological assumptions that guided the data collection and analysis.

This case study was unique in documenting a student's natural debugging behavior because for comparability many studies instead provide specific, artificial bugs for students to find and fix. Students' debugging behavior when working with these artificial bugs may be different because they did not write the buggy code and they may be less motivated to find and fix the artificial bug.

The particular case study showed the importance of what the student attended to within the problem to his process of debugging. The case study tracked how Kevin's tinkering within Scratch made particular features of state salient and how his attention to these features allowed him to identify and eliminate the bug in his program.

Directed and systematic attention to the relevant features of state may develop after students develop competence with the functionality of relevant state change operations. When Kevin eventually attended to the relevant features of state, his identification and elimination of the bug were almost immediate. His facile creation of the second turn script showed remarkable competence in modifying the state of the character's position and rotation in Scratch. This competence was not apparent during much of his debugging process where he was inattentive to the y-position of the character.

Kevin's attention within the problem limited his debugging process despite ample relevant knowledge. I hypothesize that the development of debugging competence involves refining what features of state are relevant to attend to. Developing appropriate attention within a problem is learning challenge identified in mathematics and physics [7, 8, 9], which I believe is applicable to computer science education.

9. ACKNOWLEDGEMENTS

Work described in this paper was partially supported by NSF grant DUE-1044106. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the author's and do not necessarily reflect the views of the National Science Foundation. Reviews by Andrea diSessa, Michael Clancy, Katherine Lewis, and anonymous reviewers have improved the clarity and quality of this work.

10. REFERENCES

[1] Ben-Ari, M. (2001). Constructivism in Computer Science Education. *Journal of Computers in Mathematics and Science Teaching*, 20(1), 45–73.

[2] Clancy, M. (2004). Misconceptions and Attitudes that Interfere with Learning to Program. In Fincher, S. & Petre, M. (Eds.), *Computer Science Education Research* (pp. 85-100). New York: Taylor & Francis.

[3] Cockburn, A. & Williams, L. (2001). The costs and benefits of pair programming. *In Extreme programming examined.* Addison-Wesley Longman Publishing Co., Inc. Boston, MA. 223-243.

[4] Cooper, S., Dann, W., & Pausch, R. (2000). Alice: a 3-D tool for introductory programming concepts. *The journal of computing in small colleges*. ACM. 107-116.

[5] Corbin, J. M., & Strauss, A. C. (2008). *Basics of Qualitative Research.* Thousand Oaks, CA: SAGE Publications.

[6] Dann, W., Dragon, T., Cooper, S., Dietzler, K., Ryan, K., & Pausch, R. (2003) Objects: visualization of behavior and state. *Proceedings of the 8th annual conference on Innovation and technology in computer science education, ITiCSE.* Thessaloniki, Greece, 84-88.

[7] diSessa, A. A. (1993). Toward an epistemology of physics. *Cognition and Instruction*, *10* (2-3), 105-225.

[8] diSessa, A. A., & Sherin, B. L. (1998). What changes in conceptual change? *International Journal of Science Education*, *20*(10), 1135-1191.

[9] diSessa, A. A., & Wagner, J. F. (2005). What coordination has to say about transfer. In J. Mestre (ed.), *Transfer of learning from a modern multi-disciplinary perspective* (pp. 121-154). Greenwich, CT: Information Age Publishing.

[10] diSessa, A. A. (2007). An interactional analysis of clinical interviewing. *Cognition and Instruction.* 25*(4)*, 523-565.

[11] du Boulay, B., O'Shea, T. & Monk, J. (1989). The black box inside the glass box: Presenting computing concepts to novices, in Studying the Novice Programmer (E. Soloway & J.C. Spohrer, Eds.) Hillsdale, New Jersey: Lawrence Erlbaum Associates Inc., 431-446.

[12] du Boulay, B. (1989). Some difficulties learning to program, in Studying the Novice Programmer (E. Soloway & J.C. Spohrer, Eds.). Hillsdale, New Jersey: Lawrence Erlbaum Associates Inc., 283-299.

[13] Engle, R. A., Conant, F. R. & Greeno, J. G. (2007). Progressive refinement of hypotheses in video-supported research. In R. Goldman, R. Pea, B. J. Barron & S. Derry (Eds.), *Video research in the learning sciences* (pp. 239-254). Mahwah, NJ: Erlbaum.

[14] Fay, A. L., & Mayer, R. E., (1988). Learning LOGO: A cognitive analysis, in Teaching and learning computer programming: multiple research perspectives (R. E. Mayer, Ed.). Hillsdale, New Jersey: Lawrence Erlbaum Associates Inc., 55-74.

[15] Hammer, D. (2000). Student resources for learning introductory physics. American Journal of Physics. 68(1), 52-59.

[16] Kahney, H. (1989). What do novice programmers know about recursion? in Studying the Novice Programmer (E. Soloway & J.C. Spohrer, Eds.) Hillsdale, New Jersey: Lawrence Erlbaum Associates Inc., 209-228.

[17] Kurland, D. M., & Pea, R. D., (1989). Children's mental models of recursive logo programs. *Studying the Novice Programmer (E. Soloway & J.C. Spohrer, Eds.)* Hillsdale, New Jersey: Lawrence Erlbaum Associates Inc., 315-323.

[18] Lewis, C. M. (2011). Is pair programming more effective than other forms of collaboration for young students? *Computer Science Education. 21(2).* 105-134.

[19] Lewis, C. M. (2010). How programming environment shapes perception, learning and goals: Logo vs. Scratch, *ACM SIGCSE Bulletin. 41(1)*, 346-350.

[20] Lewis, C. M. & Shah, N. (2012). Building Upon and Enriching Grade Four Mathematics Standards with Programming Curriculum. *ACM SIGCSE Bulletin. 43(1).* 57-62.

[21] Lopez, M., Whalley, J., Robbins, P., & Lister, R. (2008) Relationship between reading, tracing and writing skills in introductory programming. Proceedings of the fourth International Workshop on Computing Education Research.

[22] Powers, K., Ecott, S., and Hirshfield, L. M. (2007). Through the looking glass; teaching CS0 with Alice. *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, Covington, KY, 213-217.

[23] Sajaniemi, J. & Kuittinen, M. (2005) An Experiment on Using Roles of Variables in Teaching Introductory Programming. *Computer Science Education*, 15(1), 59 - 82.

[24] Sajaniemi, J., Kuittinen, M., & Tikansalo, T. (2008) A study of the development of students' visualizations of program state during an elementary object-oriented programming course. Journal on Educational Resources in Computing. 7(4) 1-31.

[25] Schoenfeld, A. H. (2007). Reflections on an assessment interview: What a close look at student understanding can reveal. In A. H. Schoenfeld (Eds.) *Assessing Mathematical Proficiency* (pp. 267-280). Cambridge: Cambridge University Press.

[26] Schoenfeld, A. H. (1985). Making sense of "out loud" problem-solving protocols. *The Journal of Mathematical Behavior,* 4. 171-191.

[27] Soloway, E. & Spohrer, J. C. (1989). Studying the Novice Programmer. Hillsdale, New Jersey: Lawrence Erlbaum Associates Inc.

[28] Siegler, R. S. (2006). Microgenetic analyses of learning. In W. Damon & R. M. Lerner (Series Eds.) & D. Kuhn & R. S. Siegler (Vol. Eds.), *Handbook of child psychology: Vol. 2: Cognition, Perception and Language*. (6th ed) Hoboken, NJ: Wiley, 464–510.

[29] Yin, R. K. (1989). Case Study Research: Design and Methods. Sage Publications, Inc. Thousand Oaks, CA.