Semantic Reasoning in Young Programmers

David S. Touretzky
Computer Science Dept.
Carnegie Mellon University
Pittsburgh, PA 15213
dst@cs.cmu.edu

Christina Gardner-McCune
Dept. of Computer & Info.
Science & Engineering
University of Florida
Gainesville, FL 32611
gmccune@ufl.edu

Ashish Aggarwal
Dept. of Computer & Info.
Science & Engineering
University of Florida
Gainesville, FL 32611
ashishjuit@ufl.edu

ABSTRACT

Reading, tracing, and explaining the behavior of code are strongly correlated with the ability to write code effectively. To investigate program understanding in young children, we introduced two groups of third graders to Microsoft's Kodu Game Lab; the second group was also given four semantic "Laws of Kodu" to better scaffold their reasoning and discourage some common misconceptions. Explicitly teaching semantics proved helpful with one type of misconception but not with others. During each session, students were asked to predict the behavior of short Kodu programs. We found different styles of student reasoning (analytical and analogical) that may correspond to distinct neo-Piagetian stages of development as described by Teague and Lister (2014). Kodu reasoning problems appear to be a promising tool for assessing computational thinking in young programmers.

CCS Concepts

•**Social and professional topics** → **Model curricula; K-12 education;** *Computational thinking;*

Keywords

Kodu Game Lab; formal reasoning; programming idioms

1. INTRODUCTION

The strong scaffolding provided by drag-and-drop editors has removed syntax as an obstacle to children's programming, but semantic errors can never be completely eliminated regardless of programming environment. Explicit instruction in language semantics might help students reason about programs more effectively and develop their abstract reasoning skills. We investigated this idea in the context of Microsoft's Kodu Game Lab [4], a rule-based language whose high-level primitives allow interesting programs to be constructed in just 2-3 lines. Here we report results of two four-session "Kodu camps" conducted with third graders. Participants were asked to demonstrate their understanding by predicting the behavior of a character given a code fragment and initial state of the world. In the second camp we introduced four novel "Laws of Kodu" (Figure 1) which concisely capture key features of Kodu's semantics. While this appeared to help students reason more correctly, we also saw evidence that the laws could be misapplied.

SIGCSE '17, March 08-11, 2017, Seattle, WA, USA

DOI: http://dx.doi.org/10.1145/3017680.3017787

2. REASONING ABOUT PROGRAMS

Research on program reading and comprehension stretches back to the mid-1980s. One of the initial arguments was by Deimel [1], who claimed that **code reading** is just as important as learning to code. Soloway suggested that teaching **effective reasoning strategies** should be emphasized, writing that *"learning to program amounts to learning [both] how to construct mechanisms and how to construct explanations"* [6]. More recently, Guzdial wrote that it is important for students to develop a robust "**notional machine**" (a type of mental model described by du Boulay et al. [2]) that will allow them to understand how programs work [3].

Teague and Lister [7, 8] offered a neo-Piagetian theory of programmer cognitive development and identified three important developmental stages in novices. At the **sensorimotor** stage, students hold many misconceptions, are unable to reason effectively about programs, and code by trial and error. At the **preoperational** stage they can trace code but can only reason intuitively; they do not reliably see relationships between program components. At the **concrete operational** stage they reason more abstractly and also recognize higher order relationships such as *conservation*, *transitivity*, and *reversibility*. But progress is multi-faceted: the components of reasoning advance at different rates. Thus, students may become capable of demonstrating concrete operational reasoning for some types of problems while remaining preoperational for other types.

We have developed a computational thinking curriculum using Kodu whose primary goal is to get children into the habit of reasoning about programs. Even young children have experienced this type of reasoning when they discuss the rules of card games such as Go Fish, or board games such as Chutes and Ladders. It's not implausible that they could learn to reason abstractly about computation as well, given a developmentally appropriate computing framework.

3. THE SEMANTICS OF KODU

Kodu programs are collections of WHEN-DO rules organized into pages. We consider only single-page programs here. The WHEN part of every rule begins with a predi-

Figure 1: The Laws of Kodu.

cate tile; the DO part begins with an action tile. Additional tiles supply arguments and modifiers. The rules on a page run repeatedly, in parallel, with each cycle taking a few milliseconds. The apparently continuous behavior of a Kodu character is the result of many discrete rule firings.

We say that a rule "can run" if its WHEN part is true. When a rule runs, it may "fire" its action, unless the action has been overridden by another rule (discussed below).

We developed new graphical notation to illustrate to students the status of a rule (Figure 1). A green checkmark indicates that the rule's WHEN part is satisfied, while a red cross together with a grayed-out WHEN part indicates that the WHEN part could not be satisfied. If the DO part is grayed out, this indicates that the action was not fired. If the entire line including the rule number is grayed out and there is neither a check nor a cross, which can only occur with indented rules (Fourth Law), the rule was not eligible to run because its parent could not run.

Conflict resolution is essential to understanding Kodu semantics. Variable binding conflicts can arise on the WHEN side: in a world with multiple apples, which one should a "see apple" pattern select? Action conflicts can arise on the DO side: if two rules are trying to move a character in different directions, where should it go? We formulated four laws to explain how conflicts are resolved and how the rule interpreter functions. Additional laws are possible.

The First Law of Kodu explains how variable binding works. Our initial wording of the First Law was "Rules pick the closest matching object." After observing some students mistakenly thinking that this was a joint decision (the "collective choice fallacy"), we adjusted the wording to "*Each* rule picks the closest matching object." This change had not yet been made at the time of the camps reported here.

The Second Law indicates that rules do not run sequentially; any rule with a true WHEN part can run, and if a rule can run, it will. We introduced this law to counter the "sequential procedure fallacy" that a page of rules is an ordered sequence of actions, as is the case with Scratch or Python procedures. Kodu's graphical convention of numbering the rules on a page also unfortunately affords misinterpreting them as ordered steps. We have seen evidence from past studies that some students held the sequential procedure fallacy. To convince them otherwise, the graphic for the Second Law illustrates that "Pursue and Consume" rules produce the same behavior even if their order is reversed. ("Pursue and Consume" is the first idiom or design pattern students learn in our curriculum [9].)

It is tempting to summarize the Second Law as "rule order doesn't matter," but rule order does matter when actions conflict. In such situations the earliest (lowest numbered) fire-able action wins. This is expressed as the Third Law.

The graphic for the Third Law was designed to counter a misunderstanding about the First Law. Although the blue apple is closer to the kodu than the red one, the kodu is proceeding to the latter. The First Law applies to each rule individually, resolving conflicts specific to that rule's WHEN pattern. It does not mediate conflicts between rules, which can only arise in the DO part. Thus, the First Law determines which apple each rule selects, but it is the Third Law that causes the kodu to move toward the red apple rather than the blue one. The yellow arrow in the Third Law graphic in Figure 1 indicates that rule 2's action (grayed out) has lost to rule 1's action.

The Fourth Law explains that rule indentation establishes a dependency relationship: an indented rule can run only if

its *parent* can. The parent of an indented rule is the first preceding rule with less indentation.

Rule dependency has several uses in Kodu. If the child rule has an empty WHEN part and both rules have non-empty DO parts, the DO parts form a compound action similar to a two-line block in the THEN part of a procedural language, controlled by the parent rule's predicate. This is the basis of the Do Two Things idiom, which is the second idiom students learn in our curriculum [10]. Other uses of rule dependency were not covered in the study reported here.

4. SCAFFOLDING STUDENT REASONING

The idioms or design patterns at the heart of our curriculum scaffold the understanding of programs in terms of more meaningful chunks than individual statements. Students were explicitly taught these idioms, and provided with tangible idiom catalogs in the form of flash card decks that were always available to them. Assessment tasks both reinforced the idioms and measured understanding. For example, when studying Pursue and Consume, we asked students to read a rule and classify it as either a pursue rule or a consume rule. More challenging assessment tasks presented variations on an idiom, such as Pursue and Consume with two consume rules, and asked students to predict the result.

Mental simulation (or program tracing) is an essential skill for predicting program behavior. Given a code fragment and initial state of the world, students must recognize that a character's starting position determines the results of only the first execution step. Once it starts moving, determining which rules run and how conflicts are resolved requires keeping track of the character's location. We've tested for this by asking children to indicate the trajectory the kodu takes when running a Pursue and Consume program to eat all the apples. A child who mistakenly orders the apples by their distance from the kodu's *starting* location is not doing this kind of mental simulation. We've found that asking children to first *draw* the trajectory before placing numbers next to the apples is more likely to elicit a correct response [10]. This is an example of scaffolding mental simulation.

One of the issues raised by our study is how to tell when students are actually doing mental simulation vs. reasoning in some other mode, such as by making analogies to programs they've seen previously. Our assessment questions were designed with this goal in mind.

5. STUDY DESCRIPTION

Two after-school Kodu camps were held at the same school, one in Nov./Dec. 2015 and one in Mar./Apr. 2016. Each camp consisted of four 80 minute sessions spaced 1 week apart, except for a two week gap between sessions 2 and 3 (Camp 1) or 3 and 4 (Camp 2) due to scheduling issues. Camp 1 had 20 participants; Camp 2 had 19. Participants were in the third grade, where they were learning Scratch in school. They used their personal tablet computers, which were supplied to all students by the school district, plus Xbox 360 game controllers[1] that we loaned to them for the duration of the study.

Sessions consisted of a mix of activities: teacher demonstrations, hands-on exercises, written assessments, and review of the answers. The first three sessions ended with small programming assignments to be completed at home and turned in at the following session. The written assessments, administered during sessions 2–4, tested recall of the material and ability to apply it to predict program behavior.

This study is part of a larger design-based research study on how to best scaffold elementary school students learning to program in and reason about Kodu. After each curriculum implementation we analyze the results and make changes based on issues in the learning and instruction that we've identified. In this case, analysis of the Camp 1 results led us to formulate a set of explicit "Laws of Kodu" with graphic representations as shown in Figure 1.

Students in Camp 2 were taught the same material as Camp 1, but were also presented with the new, explicit representation of the laws. Camp 1 students had been introduced to the ideas encapsulated in the laws, but less systematically, e.g. they verified by experimentation that rules picked the closest matching object. They saw a demonstration that changing the order of pursue and consume rules did not affect behavior. They also saw a demonstration that with multiple pursue rules, the first pursue rule shut out the second one until the first rule could no longer run. Camp 2 students experienced these same activities, but the graphic depictions of the laws were projected on a whiteboard, discussed verbally, and referred to during the demonstrations. Printouts of the laws were left on the whiteboard throughout the class. During class discussions, Camp 2 students made spontaneous references to the laws, which suggests that they found them useful for guiding their thinking.

6. DATA COLLECTION AND ANALYSIS

The written assessments collected in sessions 2–4 were scored and analyzed for evidence of misconceptions. In previous work we identified two sources of incorrect answers: (1) the sequential procedure fallacy, and (2) "negative transfer," by which we mean reasoning by analogy to previous experience rather than mentally simulating the actual code fragment presented. We call the former strategy *analogical* as opposed to *analytical* reasoning and hypothesize that this reflects different approaches to problem solving between students. Analogical reasoning is an important skill, but in formal domains such as mathematics or computer programming, imprecise analogies will not yield correct conclusions. Students need to be guided to think analytically. In neo-Piagetian terms we might say they have to progress from preoperational to concrete operational reasoning.

Since we used the same assessments in both camps, we could look for evidence of the effects of teaching explicit laws. One drawback to this backwards-compatible design is that the Camp 2 assessments did not refer directly to the laws or ask students to *restate* the laws or *explain behavior* by citing appropriate laws. We should also acknowledge that the instructor and his assistants were new to Kodu, so Camp 2 was somewhat more polished and ran more smoothly, which could contribute to better outcomes.

7. RESULTS

7.1 Basic Understanding

Students in both camps could distinguish between pursue and consume rules, recognize in which category a given rule belonged, and select the correct rule out of three graphical alternatives based on a verbal description.

[1]Kodu can also be run with a mouse or touch screen.

Students in both camps also understood the heavily emphasized principle that rules pick the closest matching object (First Law), and they could correctly indicate the trajectory a character would take to consume a collection of apples. Thus, given the right scaffolding, students could perform at least one type of mental simulation.

7.2 Order of Execution

More students in Camp 2 recognized that a consume-and-pursue rule pair would behave the same as pursue-and-consume. Figure 2 shows two questions used to measure this, and Table 1 summarizes the results.

M1Q8:

What will the kodu do in the coin world given the rules shown at left? Circle your answer:

a. Sit around waiting for a coin to bump into it.

b. Eat one coin and then stop.

c. Eat all the coins; it doesn't matter that the consume rule comes first.

d. Go to the first coin and get stuck there.

M2Q10:

What will the flying fish do in this world? Circle your answer.

a. Eat one starfish and then stop.

b. Eat all the starfish. The order of the rules doesn't matter.

c. Go to the nearest startfish and get stuck there.

d. Sit around waiting for a starfish to bump into it.

Figure 2: Rule ordering assessment: module 1 question 8 (M1Q8) and module 2 question 10 (M2Q10).

Problem	Group	A	B	C	D	Total
M1Q8	Camp 1	*11*	0	**9**	0	20
	Camp 2	*1*	0	**17**	0	18
M2Q10	Camp 1	1	**11**	1	*7*	20
	Camp 2	2	**16**	0	*1*	19

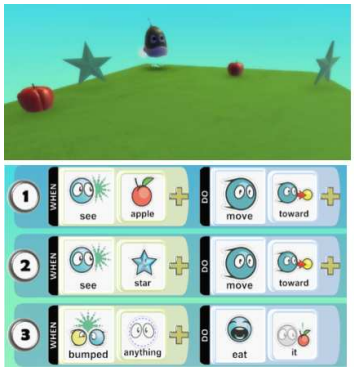
Table 1: Rule ordering results: correct responses in blue, sequential procedure fallacy in red.

In module 1 question 8 (M1Q8), answer C indicates a correct understanding of the Second Law: "Any rule that can run, will run." Answer A suggests either that the student didn't recognize this as a pursue and consume design pattern, or they held the sequential procedure fallacy. In Camp 1, only 9 of 20 students correctly chose C, but in Camp 2, 17 of 18 did. The module 2 assessments were 1-2 weeks later. For problem M2Q10 (Figure 2, bottom) answer B is correct, while answer D is consistent with the sequential procedure fallacy. In Camp 1, only 11 students out of 20 correctly chose B, but in Camp 2, 16 of 19 students did.

Six of the 11 Camp 1 students who answered A on M1Q8 also answered D on M2Q10, indicating that they held firmly to the sequential procedure fallacy. Likewise, the one Camp 2 student who incorrectly answered M1Q8 also incorrectly answered M2Q10. But overall the students in Camp 2 were less prone to the sequential procedure fallacy.

7.3 Conflict Resolution

To measure students' understanding of conflict resolution we gave them problems with two pursue rules (e.g., one to pursue apples and one to pursue stars, as in Figure 3).



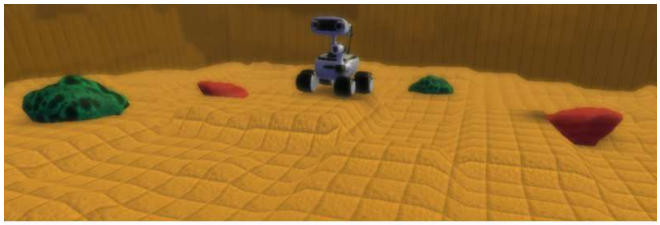
M1Q9:

With these three rules, what will the kodu eat first? Circle your answer.

a. Stars
b. Apples
c. Whichever thing is closest, no matter what kind.
d. It will choose randomly.

When will the kodu eat its first star?

a. When there are no apples left.
b. Right after it eats its first apple.
c. It will never eat a star; it will keep looking for apples forever.
d. It will only eat a star if it bumps into one by accident.



M2Q11:

With these three rules, what will the rover grab first? Circle your answer.

a. A red rock
b. A green rock
c. It will grab any rock at random.
d. The closest rock no matter what color.

When will the rover grab its first green rock?

a. When the red rocks are gone.
b. Right after it grabs a red rock.
c. It will never grab a green rock; it will keep looking for red rocks.
d. It will only grab a green rock if it bumps into one by accident.

Figure 3: Two pursue rules: module 1 question 9 (M1Q9) and module 2 question 11 (M2Q11).

Problem	Group	A	B	C	D	Total
M1Q9 part 1	Camp 1	1	**14**	*5*	0	20
	Camp 2	1	**12**	*2*	3	18
M2Q11 part 1	Camp 1	**17**	0	2	*1*	20
	Camp 2	**7**	0	0	*12*	19
M1Q9 part 2	Camp 1	**14**	*4*	*0*	*0*	20*
	Camp 2	**11**	*5*	*0*	*1*	18*
M2Q11 part 2	Camp 1	**18**	*1*	*0*	*1*	20
	Camp 2	**7**	*5*	*2*	*5*	19

Table 2: Two pursue rules: correct responses shown in blue; collective choice fallacy in green; sequential procedure fallacy in red; misapplying the Third Law in magenta. *Includes blank responses.

Multiple pursue rules cause a conflict that is resolved by the Third Law: the first pursue rule must be exhausted

before the second pursue rule can fire its action. Students who have not mastered the Third Law may think the pursue rules will alternate (in accordance with the sequential procedure fallacy.) Alternatively, they might think the pursue rules jointly choose the closest object (the collective choice fallacy), and then whichever rule matches gets to run.

Table 2 summarizes the results. The two groups performed comparably in the first assessment where the objects were of two different types (apples and stars in M1Q9). They had recently seen a demo of this situation. For part 1, Camp 1 had 14 correct out of 20, while Camp 2 had 12 correct out of 18. But in the second assessment, where the question involved a single type of object with different colors (rocks in M2Q11), Camp 1 outperformed Camp 2 on part 1: Camp 1 had 17 correct out of 20, while Camp 2 had only 7 correct out of 19, with the remaining 12 responses indicating that the Mars rover would choose the closest rock no matter the color (collective choice fallacy).

Four of the six Camp 2 students who responded incorrectly to M1Q9 part 1 also responded incorrectly to M2Q11 part 1. The erroneous Camp 2 responses suggest that students were mis-applying the First Law, which had been heavily emphasized in class and was presented in its more ambiguous form ("Rules pick" instead of "Each rule picks".) Although students in both camps had seen examples where a character consumed all objects of one color before pursuing objects of a second color, this seemed to have less influence on the students in Camp 2.

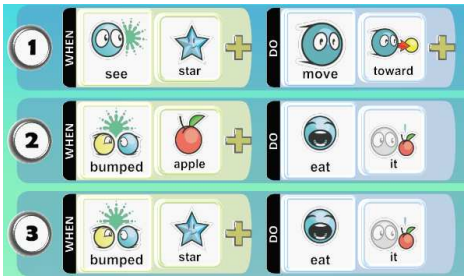
Part 2 asked when the first lower-priority object (stars in M1Q9, green rocks in M2Q11) would be consumed. For M1Q9 part 2 the two camps performed comparably, but for M2Q11 part 2 Camp 1 again outperformed Camp 2. In Camp 1, 18 out of 20 students correctly answered A (when the red rocks are gone) on M2Q11 part 2. In Camp 2 only 7 of 19 students got this right; 5 chose B (right after it grabs a red rock), 2 chose C (never), and the remaining 5 chose D (only if it bumps one by accident). Answer B suggests the sequential procedure fallacy, while answers C and D are consistent with *mis-applying* the Third Law: not realizing that rule conflict ends when the red rocks have been exhausted. Answer D might also be explained by negative transfer because in module 1 students had seen a pursue rule with two consume rules, and the "only if it bumps into one by accident" answer was correct.

7.4 Anomalous Rule Sequences

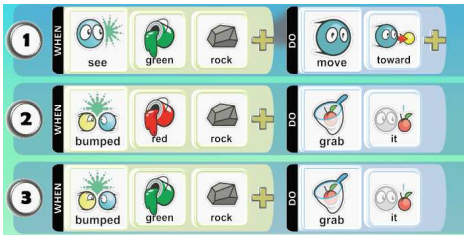
To further test students' reasoning abilities, we gave them problems with one pursue rule followed by two consume rules (Figure 4). Two consume rules do not cause a conflict because they will never be simultaneously runnable. But only the second one was supported by a matching pursue rule, so the kodu will never pursue the first type of object; it can still consume it if it bumps into one by accident while pursuing something else. Students had not encountered this situation in any of the demonstrations they saw or the programs they wrote themselves. Answering these problems correctly requires careful attention to what the rules say and how the laws govern them. Reasoning by analogy to previous programs will not work in this situation.

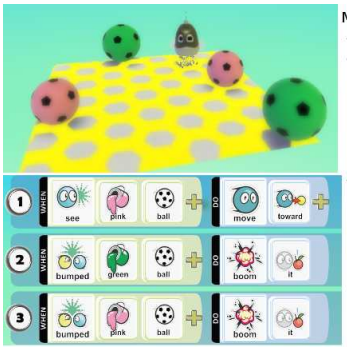
Table 3 summarizes the results. Part 1 of each problem asked what the kodu (or Mars rover) would consume first. Students performed roughly equally across camps, but the percentage of correct answers *declined* over time. Combining part 1 figures from both camps, correct answers declined from 35/38 (92%) for M1Q10, to 31/39 (79%) in M2Q12, down to 20/38 (53%) in M3Q6. Most incorrect responses were consistent with the sequential procedure fallacy. The initially high success rate might be explained by students paying attention to just the first rule, which was always the pursue rule, and assuming by analogy to previous programs that whatever is pursued will be consumed. As they gained experience and were more likely to attend to all the rules, they became confused by the presence of the unsupported consume rule, which always preceded the supported one.

M1Q10: same world as M1Q9 (Fig. 3)



M2Q12: same world as M2Q11 (Fig. 3)





M3Q6:

With these three rules, what will the kodu boom first?

a. A green soccer ball
b. The closest soccer ball of any color
c. A pink soccer ball
d. Nothing

When will the kodu boom a green ball?

a. It will boom green balls first.
b. It will boom green balls when all the pink balls are gone.
c. It will never boom a green ball unless it bumps one by accident.
d. It's random; you can't predict it.

Figure 4: Problems involving anomalous rule sequences: one pursue and two consume rules.

Problem	Group	A	B	C	D	Total
M1Q10 part 1	Camp 1	**19**	*1*	*0*	0	20
	Camp 2	**16**	*2*	*0*	0	18
M2Q12 part 1	Camp 1	*4*	**16**	0	*0*	20
	Camp 2	*3*	**15**	0	*1*	19
M3Q6 part 1	Camp 1	*8*	*0*	**9**	2	19
	Camp 2	*6*	*1*	**11**	1	19
M1Q10 part 2	Camp 1	*12*	*2*	**5**	*1*	20
	Camp 2	*6*	*3*	**6**	*3*	18
M2Q12 part 2	Camp 1	*8*	*2*	*1*	**9**	20
	Camp 2	*6*	*4*	*1*	**8**	19
M3Q6 part 2	Camp 1	*6*	*3*	**7**	2	19*
	Camp 2	*2*	*6*	**9**	2	19

Table 3: Two consume rules. Coding as in Table 2.

Part 2 of each problem asked when the kodu or rover would consume whatever the non-pursued object was (apple, green rock, or green soccer ball). Students in both camps had trouble with this question. The correct answer is "Never, unless it bumps into one by accident." Only about one third answered correctly for M1Q10 part 2, improving to roughly one half for M2Q12 part 2 and M3Q6 part 2. Incorrect responses consistent with the sequential procedure fallacy can arise from two types of imperfect analogy, depending on whether one focuses on rules 1 and 2 or on rules 2 and 3. The third possible incorrect answer, "When all the [pursued objects] are gone," suggests a faulty analogy to two-pursue-rules problems. Students may be aware that with two pursue rules, the first must be exhausted before the second can take control. But applying this knowledge to a situation with two consume rules is a misapplication of the Third Law.

Inconsistent responses on the two parts of M3Q6 support the hypothesis that some children were reasoning preoperationally. In part 1, 6 Camp 2 students said (incorrectly) that the kodu would boom a green ball first, but in part 2 only 2 of those 6 said the kodu would boom green balls first. Why weren't they consistent? Preoperational reasoners don't understand the relationships between rules but can interpret rules individually. In part 1 they can scan the list of rules and find that the first consume rule is for green balls. But part 2 asks a more complex question whose answer is not an object, but a state of the world during program execution. Preoperational reasoners can only approach such questions indirectly. Two used their answer on part 1 to answer part 2 consistently ("green balls first"). Two answered "when the pink balls are gone," reasoning analogically to earlier programs they'd seen. One chose "it's random; you can't predict it," and one correctly answered "never," contradicting their answer on part 1.

8. CONCLUSIONS

Kodu's high level primitives afford writing non-trivial programs that are only 2-3 lines long. This allowed us to demonstrate that reasoning about program behavior is accessible to third graders. To correctly predict program behavior students must engage in mental simulation following the laws of Kodu semantics.

All students demonstrated scaffolded mental simulation by trajectory tracing, and some showed true analytical reasoning in answering the more challenging assessment questions. But other students appeared to cling to the sequential procedure or collective choice fallacies, or to rely on analogies to previously seen programs rather than applying the laws learned in the study. We attribute this to differences in reasoning style combined with limited time on task. This may reflect the different rates at which students were progressing from the sensorimotor to preoperational to concrete operational stages of reasoning within the Kodu domain.

The results of this study suggest several ways to better scaffold the cognitive development of young Kodu programmers. One is to introduce exercises focused specifically on the laws, such as showing some bit of behavior and asking students to explain it in terms of the laws, or showing some anomalous behavior and asking students which law would be violated if a character did behave that way. Another strategy is to present two-pursue-rule and two-consume-rule programs as design patterns in their own right and investigate the differences in their behavior.

Kodu is a rich medium for exploring student reasoning beyond pursue and consume programs. Its rule dependency relation (Fourth Law) supports both block structure and if/then/else constructs (our Do Two Things, Count Actions, and Default Value idioms). Its support for state machines affords reasoning about abstract concepts such as state reachability. Investigating student understanding of these constructs will require studies with considerably more contact hours.

How can the skills children develop through this curriculum transfer to procedural languages such as Scratch or Python? Because they use simpler primitives, it is more difficult to construct short nontrivial programs to reason about, but it may be possible.[2] And Kodu's WHEN-DO rules bear some resemblance to Scratch event handling. In any case, mental simulation and analytical reasoning are important in all types of programming. If children become concrete operational thinkers about Kodu programs, we expect they will then be quicker to reach this developmental stage when learning other languages.

Acknowledgments

Funded by a gift from Microsoft Research. Thanks to Stephen Coy of Microsoft FUSE Labs for suggesting the check/cross imagery and for many helpful discussions.

9. REFERENCES

[1] L. E. Deimel Jr. The uses of program reading. In *SIGCSE Bull.*, volume 17, pages 5–14, 1985.

[2] B. du Boulay, T. O'Shea, and J. Monk. The black box inside the glass box: presenting computing concepts to novices. *Int J. Man-Machine Studies*, 14:237–249, 1981.

[3] M. Guzdial. *Learner-Centered Design of Computing Education: Research on Computing for Everyone.* Morgan & Claypool, 2015.

[4] M. B. MacLaurin. The design of Kodu: A tiny visual programming language for children on the Xbox 360. In *Proceedings of POPL '11)*, pages 241–246, 2011.

[5] D. Parsons and P. Haden. Parson's programming puzzles: a fun and effective learning tool for first programming courses. In *ACE '06 Proceedings of the 8th Australian Conference on Computing Education*, volume 52, pages 157–163, 2006.

[6] E. Soloway. Learning to program = learning to construct mechanisms and explanations. *Comm. ACM*, 29(9):850–858, 1986.

[7] D. Teague. *Neo-Piagetian Theory and the Novice Programmer.* PhD thesis, Queensland University of Technology, 2015.

[8] D. Teague and R. Lister. Programming: reading, writing, and reversing. In *Proc. ITiCSE '14*, pages 285–290, 2014.

[9] D. S. Touretzky. Teaching Kodu with physical manipulatives. *ACM Inroads*, 5(4):44–51, 2014.

[10] D. S. Touretzky, C. Gardner-McCune, and V. Aggarwal. Teaching 'lawfulness' with Kodu. In *Proceedings of SIGCSE'16*, pages 621–626, 2016.

[2]Parson's problems [5] might be seen as a procedural language analogue of the problems we developed for Kodu.