DEVELOPING COMPUTATIONAL REASONING SKILL AND MENTAL SIMULATION ABILITY IN ELEMENTARY SCHOOL STUDENTS USING MICROSOFT KODU

By

ASHISH AGGARWAL

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2017

To the Great Land of Bharat,

whose civilizational values of Dharma will continue to motivate generations

&

To my Guru, Grandparents and Parents,

who have encouraged, supported and guided me every time

ACKNOWLEDGMENTS

I would first like to sincerely thank my thesis advisor Dr. Christina Gardner-McCune of the C.I.S.E Department at the University of Florida for her continuous support of my master's study and research, for her patience, motivation, enthusiasm, and immense knowledge. Her guidance at every step has helped me become a better student and a curious researcher. I will be highly indebted to her for everything I have learnt from her.

Besides my advisor, I would like to thank Dr. David S. Touretzky of the Computer Science Department at Carnegie Mellon University. His support and guidance in analysis and understanding concrete ideas has helped me grow immensely in my academic journey. I am also extremely thankful to my committee members, Dr. Kristy E. Boyer and Dr. Shaundra B. Daily for their patience and valuable feedback.

My sincere thanks go to all the individuals who helped me conduct research studies. I want to thank Dr. Fred Ball and Mrs. Claire Robinson for supporting me to conduct studies. I want to thank Kyuseo Park and Jiyoung Kang and my lab mates especially Joseph Isaac and Ashley Cahill for their assistance in teaching and designing research.

Finally, I would like to express my profound gratitude to my grandparents, parents and my sister for providing me with unconditional support and continuous encouragement throughout my years of study. This accomplishment would not have been possible without them. Thank you!

TABLE OF CONTENTS

page

LIST OF TABLES

LIST OF FIGURES

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

DEVELOPING COMPUTATIONAL REASONING SKILL AND MENTAL SIMULATION ABILITY IN ELEMENTARY SCHOOL STUDENTS USING MICROSOFT KODU

By

Ashish Aggarwal

May 2017

Chair: Christina Gardner-McCune
Major: Computer Science

We believe that computational reasoning is an aspect of programming literacy and computational thinking which can be defined as the ability to read, write, trace, debug, and predict programs. This thesis aims to understand the development of computational reasoning in elementary school students. We use Microsoft's Kodu Game Lab and a formal reasoning skill-development curriculum to assess how students reason about programs and the challenges which they have to overcome in their journey to become computational thinkers.

We present results from three different studies which focus on refining instructions, identifying and removing misconceptions and fallacies, and understanding common reasoning patterns in the students. We find that students have misconceptions and fallacies which can be addressed by explicitly explaining them how to interpret rules and simulate programs. This research is valuable as it will help educators and researchers to gain insight into elementary school students' models of reasoning and provide strategies that have been found to reduce misconceptions and fallacies which negatively affect students' ability to reason about programs.

CHAPTER 1
INTRODUCTION

Over the last decade, researchers and educators have developed different curriculum, resources, and strategies to foster computer science learning in K-12 education. Many of these curricula have been developed around visual programming environments like Scratch, Alice, App Inventor etc., to introduce basic CS concepts to K-12 students. Recent national attention to the deficit of CS in K-12 has opened up opportunities for these curricula and programming environments to be integrated into the K-12 curriculum to address the challenge of getting CS into students' regular curricula [95]. As a result, more and more schools are adopting CS courses and introducing CS at different levels, especially at elementary and middle schools [33, 42]. Currently most of the research on CS education and practice has focused on undergraduates and high school students. This is a cause of concern as we do not know what type of computing activities and assessments will be helpful in the elementary and middle school level. Moreover, as computing principles are introduced at elementary and middle school levels, there also is a need for more focused research on how these students are learning CS concepts and computing principles.

Visual programming environments such as Alice [18] and Scratch [70] have been helpful in introducing younger students to programming and reducing the inherent barriers in learning text-based programming languages. These programming environments scaffold students' abilities to learn programing and support artifact design by minimizing the syntactical complexities of programming and hiding the how programs are compiled and executed.

Often curricula associated with these visual programming environments focus on engaging students in the development of creative artifacts to foster motivation and sustained engagement in programming as well as teaching students CS concepts. But there is a lack of focus on teaching students about how computers interpret and execute program instructions. Likewise, there is a lack of research on how students develop their understanding on how computers interpret and execute program instructions. Students' face challenges in developing ability to read programs written by others, trace, mentally simulate, predict, and debug program behavior. Moreover, these skills are essential for students to master as they advance in programming expertise and develop their ability to reason about programs. In this thesis, we define Computational Reasoning as the ability to read and write programs and interpret and predict program behavior.

This thesis aims to address the gap in the literature about how elementary students reason about programs. It describes several studies conducted to track, evaluate and qualitatively analyze the development of computational reasoning ability in elementary students using Microsoft Kodu Game Lab using a computational thinking curriculum developed for Kodu. The findings presented in this thesis describe students' ability to read and write programs, and mentally simulate and predict program behavior. It also explores different instructional conditions and resources needed to foster the development of computational reasoning skills in elementary school students.

CHAPTER 2
BACKGROUND AND LITERATURE REVIEW

2.1 Novice Programmers

Programming has often been considered a difficult topic for novices who have faced several difficulties while learning to program [48, 54, 55, 71, 80]. Often, the first barrier to textual programming is mastery of syntax [79]. However, as students learn to program, they begin encountering challenges in their understand of CS concepts like variables [40, 73], loops [17], boolean conditions and control structures [5, 30], message passing [62], and concurrency [36, 37]. A frequent source of a students' misconceptions about programming has been the linguistic transfer of terminology from English to the syntax of programming languages where the terms don't mean the same thing. For example, the words "while" and "then" do not mean the same in Pascal as they do in English [80, 81]. Other common sources of confusion include overgeneralization of algebraic notations (e.g. assignment operators and variables) and previous programming experience in a different language [12, 25].

While learning to program, novices also face difficulties in problem solving. McCraken's 2001 ITiCSE working group [55] found that many students have fragile knowledge of basic programming principles which negatively affects their ability to systematically perform programming tasks like tracing. Possible sources of students' problem-solving difficulties have been reported in various studies described by Winslow [98] who suggests that novices:

- lack an adequate mental model of the problem [35],
- have "fragile knowledge" (concepts a student understands but fails to use) of programming [64],
- use general problem-solving strategies (i.e., copy a similar solution or work backwards from the goal to determine the solution) rather than problem-dependent problem-solving strategies,

- tend to approach programming through control structures, and
- use a line-by-line, bottom-up approach to problem solution [6]

However, difficulties novices face when learning to program have also been attributed to misconceptions related to input, recursion, previous programming experience, and mathematical notions. One of the most common misconceptions is novices' misapplication of analogy or analogical reasoning [12], which can be understood by an example cited by Du Boulay [22] where he says "A variable is like a box": a box can hold more than one thing, but some students think that a variable can also hold more than one thing. Pea [63] observes that novices view programming as analogous to conversing with a human. As a result, the novice assumes that a computer will do what they mean for it to do instead of what they have commanded it to do. Halasz et al. [29] predicted that analogical models of computing are doomed to fail because "no simple analogical model is sufficient to completely explain the operation of a computer system". Corney et al. [14] found students' misconceptions related to variables, assignment statements, and "if" statements by using a pre-test screening.

Research by Lister [47] and colleagues [84] aimed to measure the cognitive development of novice programmers using a neo-Piagetian theory of skill and reasoning development. Their conjecture was that there are four main stages in the cognitive development of novice programmers. The first stage is the Sensorimotor stage, the stage in which a "novice programmer cannot reliably manually execute a piece of code and determine the final values in the variables" (p. 41), which is due to "misconceptions about programming language semantics and the inability to trace" (p. 41) the code [84]. The second stage is the Preoperational stage, the stage in which novices have learned

to trace code accurately and efficiently, yet still struggle to understand the relationship between different lines of code and the program as a whole—in this stage, novices lack an understanding of how several lines of code work together to perform a computational process. The third stage is the Concrete Operational stage, the stage in which novice programmers are "capable of deductive reasoning" and "should not need to perform an explicit, complete written trace to arrive at the answer" (p. 41) [84]. However, at this stage novices are only able to reason about short pieces of code which "perform relatively familiar computational processes" (p. 41) [84]. The last stage is the Formal Operational stage which involves the most abstract type of reasoning, a type of reasoning which is a genuine representation of how expert programmers reason. Lister and colleagues used this neo-Piagetian framework to study novice programmers' abilities to read and comprehend programs. They also looked at the relationship between program literacy and program tracing to validate the utility of categorizing the development of novice programmers' reasoning abilities.

While this framework is useful for understanding how novice undergraduate students reason about text-based programs, there have been concerns about the effectiveness of CS assessments, which can identify misconceptions and evaluate CS learning of students. Tew [85] claims that "the field of computing lacks valid and reliable assessment instruments for pedagogical or research purposes" (p. xiii). Thus, for any research to evaluate CS learning, there is a need to have valid assessments and curriculum.

The research discussed in this section highlights the need for a similar classification and assessment of elementary-student's reasoning ability when it comes

to visual-programming languages. This thesis aims to explore elementary-school students' ability to reason about programs using visual programming languages by studying appropriate assessment instructions, resources, and instructional practices.

2.2 Visual Programming Languages and Student Learning

There has been much debate over the comparative suitability of programming languages used to introduce programming to novices [9, 28], but Java and C++ have been most popular in both the industry and at universities [19, 82]. Much of the debate has been focused on whether to use industry languages (e.g., Java, C, and C++) or teaching languages (e.g., Pascal and Python) [74, 75,93]. Milbrandt [57] found that programming languages designed for teaching should have simple syntax, be easy to learn, powerful, structured in design and universal in use. Text-based languages such as Pascal and Logo were first designed to fulfill these educational requirements, and many studies suggest that they were useful for educational purposes [74, 75]. Over time, more educational languages have come about. Most recently, visual-programming languages and environments such as Scratch, Alice, and Kodu have been developed to fulfil educational requirements [31].

2.2.1 Scratch

One of most widely used programming environment for kids, Scratch is a two-dimensional programming environment, developed by MIT, which enables children to use instructions in a drag-and-drop type of editor to make interactive media and games [70]. Scratch is composed of scripts which are created by dragging-and-dropping blocks which represent elements of programs like expressions, conditions, statements, and variables. In Scratch, the "code blocks only lock into place in syntactical valid ways, therefore 'bugs' are always semantic error or a misremembered detail of a language" (p.

346) [44]. Scratch makes concepts like flow of control or, loops and conditionals more natural [61]. Naturalizing these features in a novice minimizes syntactical errors, and helps introduce novice programmers to the overall fundamentals of CS.

In a quest to understand the effectiveness of Scratch, Meerbaum-Salant et al. [56] found that middle school students who used Scratch without explicit instructions tend to concentrate on designing and modifying costumes and sound of sprites without paying attention to CS concepts. They found that “10% of the projects uploaded to the Scratch website use the conditional looping constructs *repeat until <condition>* and only about 20% used variables” (p. 240), thus they were interested in investigating the use of Scratch to teach CS concepts with explicit instructions. Their research showed that students were able to achieve a reasonable level of understanding of CS concepts using Scratch. Though “difficulties were encountered in teaching specific topics such a s repeated execution, variables, and concurrency” (p. 261), they suggested that these difficulties could be solved “by careful teaching, where the relationships between concepts and their implementation in language constructs are taught explicitly and in detail” (p.261) [56]. These results suggest that while Scratch may provide an engaging visual environment, not every kind of student engagement with Scratch transfers into valuable learning of CS concepts. These results also suggest that explicitly teaching a concept like loops and conditions results in building the reasoning ability of students.

Maloney et al. [53] studied the use of Scratch by middle school students in an after-school clubhouse and found that “without realizing it, most Scratch users make use of multiple threads” (p. 368). According to Malan and Leitner [52], undergraduate students found the use of Scratch to be successful in introducing the fundamentals of

programming to inexperienced students. On comparing Scratch with Logo, Lewis [44] found that “students that learned Logo had on average higher confidence in their ability to program and students were no more likely to plan to continue to program after the course or view the learning of topics as difficult if they learned Logo or Scratch” (p. 346). This was contrary to the original hypothesis, which stated that a student using Scratch would have a more positive attitude towards programming because of the number of affordances it offers. Such results help to “isolate the features of student experience impacted by the content of programming and those impacted by the programming environment” (p. 350) [44]. In the following discussion, we discuss other visual programming environments: AgentSheets and Alice.

2.2.2 AgentSheets

AgentSheets is a rapid-visual, agent-based game-authoring environment [66] which can be used to create games and simulations especially by novice programmers. Its syntax follows “if-then” semantics, which provides high-level abstraction because novices are not required to be concerned about low-implementation details. An “If” statement represents a condition and a “then” statement represents an action. Students are able to put the conditions directly in the ‘if’ and ‘then’ part while hiding how it will be implemented, which results in providing high-level abstraction. AgentSheets was created by Repenning et al. [67, 68] as a part of creating computational thinking tools which have educational purpose and not programming. It follows the three stages of the computational-thinking process: Abstraction, automation and analysis. Basawanpatna et al. [7] studied students’ use of AgentSheets, and found that it “has the ability to get middle-school students as well as college-level students interested in the field of computational science” (p. 228).

2.2.3 Alice

Alice is a 3D, interactive, animation environment used to introduce object-oriented programming to novices. It supports storytelling in the “Storytelling Alice” version which provides a set of high-level animations, a collection of 3D characters and scenery designed to spark story ideas, and a tutorial which introduces users to writing Alice programs using story-based examples [34]. Kelleher et al. [34] studied the experiences of middle school girls and compared them by using a version of Alice with one having the storytelling support (Storytelling Alice) and another without storytelling support (Generic Allice). They found that “users of Storytelling Alice and Generic Alice were equally successful at learning basic programming constructs” (p. 1455). They also found that “users of Storytelling Alice were more motivated to program; they spent 42% more-time programming, were more than 3 times as likely to sneak extra time to work on their programs, and expressed stronger interest in future use of Alice than users of Generic Alice” (p. 1455). These findings suggest that features such as “storytelling” promote engagement and motivation.

2.2.4 Summary

Research on novice programmers, covered in this section, highlights the difficulties and misconceptions that novices encounter in text-based programming environments. These include challenges with syntax, conditional looping, concurrency, analogical reasoning etc., which result in fragile knowledge. This section also highlights the use of visual-programming languages such as Scratch, Alice, and AgentSheets to provide engaging programming environments. These languages are aimed to promote conceptual learning without having students grapple with the syntax. However, persistence of conceptual issues with variables and conditional looping suggest that

these concepts need to be explicitly taught to help students develop programming proficiency. In addition, investment in effective methods to teach these concepts are also necessary, since students will not learn them on their own. This thesis aims to bridge the lack of research on how elementary students learn to reason about programs. The goal of this thesis is to study the challenges which an elementary school student faces while learning to reason about programs, and then to identify some of the strategies which might be useful to overcome these challenges.

2.3 Computational Thinking

As computing is pervading more of society, from social communication to business, there has been more of a push for everyone to develop an understanding of how the technology we live with works and how to become producers of technology as opposed to just consumers [69]. The term Computational Thinking (CT) was first used by Seymour Papert in 1996 in his article on mathematics education where he discussed the use of computers to “forge ideas” (p. 116) allowing people to better analyze and explain problems, solutions, and connection between them [60]. In 2006, Jeannette Wing, in her now widely discussed article and talks on Computational Thinking, suggested that CT is “a universally applicable attitude and skill set everyone, not just a computer scientist, would be eager to learn and use” (p. 33) [96]. She later defined CT as “the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be carried out by an information-processing agent” (s. 60) [97]. Since then, CT has gained a lot of attraction and many researchers have defined it in their own way [13, 28]. Vee et al. used the term “computational literacy” and defined it as the ability “to break a complex process down into simple small procedures and then express-or ‘write’-those procedures using the

technology of code that may be ‘read’ by a non-human entity such as computer” (p. 47) [94].

One of the most accepted definition of CT has been developed by International Society for Technology and Education (ISTE), which is a global organization serving educators, and the American Computer Science Teachers Association(CSTA), which is an organization that supports and promotes the teaching of CS and has existed since 2004. Their definition identifies nine essential concepts important for K-12 education: data collection, data analysis, data representation, problem decomposition, abstraction, algorithms, automation, parallelization, and simulation.

ISTE and CSTA has proposed an operational definition for CT as a “problem solving process that includes (but is not limited to) the following characteristics [32]:

- Formulating problems in a way that enables us to use a computer and other tools to help solve them.
- Logically organizing and analyzing data
- Representing data through abstraction such as models and simulations
- Automating solutions through algorithmic thinking (a series of ordered steps)
- Identifying, analyzing, and implementing possible solutions with the goal of achieving the most efficient and effective combination of steps and resources
- Generalizing and transferring this problem-solving process to a wide variety of problems”

These skills are “supported and enhanced by a number of dispositions or attitudes that are essential dimensions of CT” such as “confidence in dealing with complexity, persistence in working with difficult problems, tolerance for ambiguity and ability to deal with open ended problems” [16].

Despite the debate over its precise definition, CT has remained central to recent approaches that engage K-12 students in Computer Science. CT is focused on how to solve problems and construct solutions by analyzing data to identify and analyze

solutions, and then to implement them. However, we cannot infer any use of programming as an essential component of CT. In fact, program understanding might be seen independent of CT, as it is just one way to solve a problem. In this thesis, I am interested in studying computational-programming abilities and reasoning abilities of inexperienced novice programmers.

2.4 Computational Reasoning

Where CT focuses on computational understanding and program writing as a general problem-solving skill or process attainable by anyone, researchers for the past three decades have studied the skills novice programmers need in order to develop proficiency in programming and the challenges novice programmers have in understanding and reasoning about programs. Deimel [20] argued that code reading is as important as code writing. Other researchers have suggested that composing and coordinating different pieces of code and "putting the pieces together" is a major problem for novice programmers [80, 81]. Sheard et al. [77] found that students' ability to explain programs correlated positively with their ability to write code which is important in the development of novices' conceptual understanding and construction of programs. Research by Lister et al. [49] highlighted the dependence of students' code tracing ability on their code explaining ability and confirmed that "students who cannot trace code usually cannot explain code" (p. 161). They also found that "students who tend to perform reasonably well at code writing tasks have also usually acquired the ability to both trace code and explain code" (p. 161).

Perplexed by the challenges that novice programmers face, Soloway suggests that novice programmers need to be taught effective reasoning strategies and that "learning to program amounts to learning how to construct mechanisms and how to

construct explanations" (p. 851) [79]. Most recently, Guzdial [27] suggests that development of a robust "notional machine", e.g, the mental model of how programming environments work [23], will help students understand how programs work and help them to effectively read, write, trace, and understand programs.

2.4.1 Reading and Writing Programs

According to Fitzgerald, program comprehension happens mainly at two distinct levels, the syntactic and the semantic [26]. While the syntactic level involves the identification of the components that constitute a program, the sematic level involves the process followed by the computer to execute a program [26]. It has also been found that expert programmers know more than just syntax or semantics [1, 10, 50], and have the exposure to "large libraries of stereotypical solutions to problems as well as strategies for coordinating and composing them" (p. 850) [79]. This indicates how experts not only possess basic program reading-and writing-skills, but also advanced comprehending skills, and that they've developed these skills by building an understanding of design patterns or common patterns which are used in solving problems.

2.4.2 Program Tracing

In addition to program reading and writing, researchers suggest that students need to develop their ability to trace programs [1, 78, 92]. Program Tracing refers to the process of manually going step-by-step through the execution of program code. Lopez et al. studied the relationship between code reading, code writing and tracing, and found "strong statistical support for an association between code tracing and code writing skills" and "support for an association between code reading and code writing skills" (p. 109) [51]. Successful code tracing has been linked to students' ability to successfully

read and understand code. However, Lister et al. [48] have found that novices' tracing skills are poor.

Program tracing helps students develop a representation of the flow of control in a program. Tracing can be done at two levels [21]: concrete and symbolic. Concrete tracing, which deals with the actual assignment of values of a variable like "a gets the value 6" which means that a student mentally runs the program and monitors the changing values of variables. Symbolic tracing refers to the invariant relationships between variables like "a gets the value of b", which means that a person simulates the different steps in which the program is executed and monitors the changing values of variables as they are symbolically passed. Perkins et al. [65] found that concrete tracing is mentally demanding and many students do not do it carefully.

Vainio et al. [92] have discovered that factors such as an inability to use external representations and a confusion in functions and structures are among the factors responsible for a novice's poor tracing skills. Lister et al [48] found that 'doodling' or drawing diagrams and making other annotations as part of determining the function of the code, which exhibited the proof of tracing the code, is correlated with success in programming-ability diagnostics. Similarly, Kumar [41] also found evidence for correlation between solving code-tracing problems and code-writing skills. Lewis [45] has found substitution techniques like simulating execution and accumulating pending calculations which may support students in more accurately tracing execution. Fitzgerald et al. [26] observed that "success seems to be determined not only just by which strategies were employed but rather by "how well" the particular strategies were employed" (p.78).

2.4.3 Mental Simulation

Mental simulation is the ability to simulate the step-by-step execution of a program and maintain the changing state of programs' variables, data structures, and outputs. It is an essential component of computational reasoning as it builds on the ability to read, trace and predict the behavior of a program [27, 28]. Kollmansberger has studied mental simulation and suggested that it is a subset of Mental Model of Computation which has been recognized as an important learning outcome [38]. He suggests that "beyond planning and general problem solving, a consistent mental model of computation is essential for developing programming ability" (p. 128) [38]. Pennington [64] found that the best programmers constructed a mental representation of the real-world problem that they use to reason about programs and their behavior. The ability to develop an abstract mental model seems necessary for any novice programmer who seeks to increase their programming proficiency.

2.4.4 Debugging

Novice programmers' ability to trace and mentally simulate programs is closely related to their ability to debug programs. Debugging refers to the process of eliminating defects from a program. As programmers learn to debug programs, they need to leverage their understanding of the overall purpose of a program and their understanding of the program's structure in terms of what each segment of code is supposed to do. It has been found that novice programmers spend a significant amount of time on debugging [59]. Chmiel et al. recommend that to help students improve their debugging skills instructors should integrate debugging activities throughout an entire curriculum [11].

So, we see how different elements of programming, i: e., code reading, writing, tracing, mental simulation and debugging are related and essential. These skills are essential milestones in a journey of a novice programmer to become an expert.

2.5 Conclusion

As novice programmers develop their programming proficiency, they need to develop their abilities to read and write correct syntax, trace and debug the program, and predict the behavior of a program through mental simulation. The mastery of these skills signifies programming literacy and supports programmers' ability to reason computationally. Thus, this thesis uses the term Computational Reasoning to describe the ability to read and write syntactically correct programs, trace program execution, use the knowledge of program behavior to debug and even mentally simulate programs.

Much of this research presented in this chapter has been identified from in-depth studies of expert and novice programmers. This thesis aims to explore how elementary-school students can develop this ability to reason computationally about programs, to validate misconceptions and fallacies that affect students' abilities to reason, and to design and use assessment questions to evaluate students' abilities to engage in computational reasoning.

CHAPTER 3
KODU AND ITS CURRICULUM

3.1 Microsoft's Kodu Game Lab

Microsoft Kodu Game Lab is a rule-based visual programming language made specifically for 3D game development. It is designed to be accessible to children and enjoyable for anyone [83, 91]. The programming environment runs on the Xbox, Windows PC, and tablet PC platforms. It provides students with a 3D world to visualize the behavior of their programs and a rule editor to design and rapidly iterate on their programs using an Xbox game controller or keyboard for input (Figure 3-1).

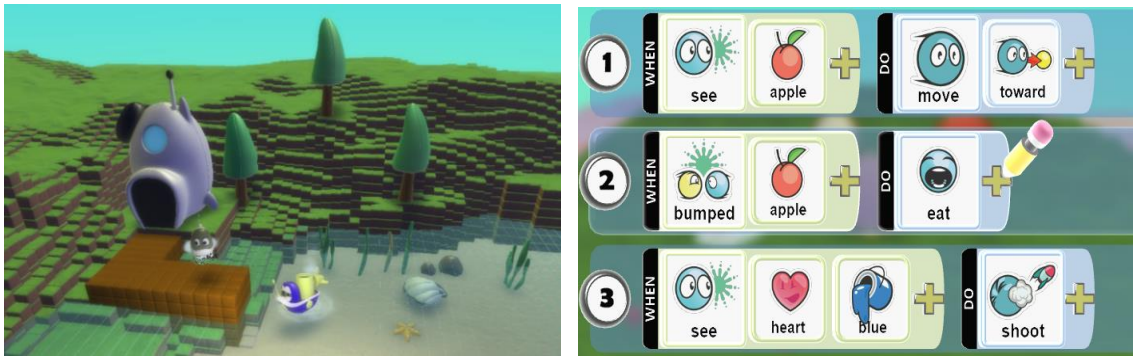


Figure 3-1. Kodu's 3D world (left) & Kodu' rule editor (right)

Kodu uses a rule-based language based on WHEN-DO conditional rules to control characters and objects in 3D worlds. The rules are organized using a sequence of tiles (e.g., objects, perceptions, and actions) to create conditional statements. For example, the equivalent of a 'Hello World' program for Kodu is the two-rule program "WHEN see apple DO move toward; WHEN bumped apple DO eat it." These two rules tell the kodu to go to the nearest apple and eat it and it will run until every apple in the world has been eaten. Every rule in Kodu repeatedly gets evaluated 50-100 times per second on the currently active page. Kodu's worlds are built on top of a real physics engine that includes gravity, inertia, friction, elastic collision, wind, water and waves

[89]. It also has built-in sound effects. The real physics behind every Kodu world allows programmers to create realistic 3D games and add a measure of interactive engagement.

Note- In this thesis, the word ‘Kodu’ refers to the 3D programming environment and the word ‘kodu’ refers to the main character present in the environment. The kodu character is often programmed and used in the activities of the curriculum.

3.2 Kodu’s Curriculum

Touretzky [86,87] has developed a comprehensive Kodu curriculum for elementary through high school students. This curriculum aims to help students to develop mastery of lawfulness in Kodu. Touretzky defines lawful reasoning or lawfulness as the ability to reason through laws which govern how a program will be interpreted and executed in any programming language [89]. The curriculum is organized around six modules that teach basic programming and game design patterns (idioms), which are presented on flashcards (Figure 3-2) and Laws of Kodu computation, which are presented on magnets (Figure 3-4 & 3-5). In addition, the curriculum includes tile manipulatives (Figure 3-3) to help students learn how to construct rules prior to navigating through the expansive menu of tiles in the Kodu rule editor [88].

3.2.1 Kodu Flashcards

Kodu Flashcards are a tangible collection of design patterns for programming games in Kodu that are laminated and bound together for quick reference and use. The front-side of each flashcard provides a conceptual description of the design pattern and a graphical representation of the resulting behavior (Figure 3-2, left). The back-side of each flashcard shows the corresponding rules for the design pattern using the rule

notations that students will see in the rule editor (Figure 3-2, right). Figure 3-2 shows the first design pattern students learn in the Kodu curriculum: Pursue and Consume (P&C). This is the 'Hello World' of Kodu described in Section 3.1. The P&C design pattern instructs a character to move toward the closest object that satisfies the rule (e.g., "WHEN see apple DO move toward"), and to consume it upon contact (e.g., "WHEN bumped apple DO eat it").

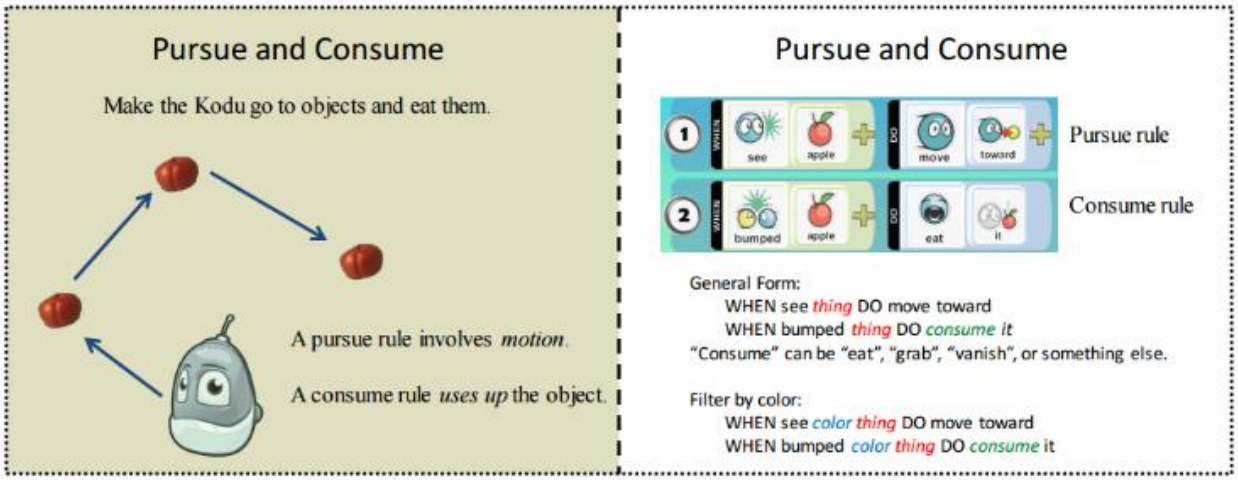


Figure 3-2. Flashcard showing Pursue and Consume idiom

We consider the Kodu flashcards as physical manipulatives because their shape and size allow students to easily identify them on a work surface and quickly flip through them, referring back to them when needed. While the reference function of the flashcards is similar to that of quick reference guides on single laminated sheets, their compact flip-able structure allows students to quickly navigate through the focal concepts and reduces visual or conceptual distractions when trying to program.

3.2.2 Kodu Tiles

Tiles are the second form of physical manipulative used in the curriculum. They are puzzle-shaped pieces created to model the WHEN-DO template and the graphical tiles in Kodu's rule editor (Figure 3-3). The WHEN part of each rule is green, and the DO

part is blue. The tile set also features a special indentation tile to help students understand indentation in Kodu, which functions similarly as indentation in Python. The primary goal of the tile manipulatives is to help students recognize and construct syntactically correct rules. In particular, they help students understand which tiles go in which part of the WHEN-DO template.

Figure 3-3. Tiles showing Pursue (1) and Consume (2) rules

In general, the curriculum is designed for instructors to use the tiles to model how to construct rules from the flashcards and discuss the types of objects that are found in the two parts of the WHEN-DO template. The students are then expected to practice constructing rules with the tiles and then implement them in the rule editor. Later when students are working on activities, they are expected to refer back to the tiles as a reminder of how the concepts work and how to construct the rules. In this way, the tiles and flashcards are designed to reinforce the principles of pattern recognition and rule construction in Kodu Game Lab.

3.3 Laws of Kodu

In Kodu, every rule is checked to see if the condition in the WHEN-part of the rule is true and if it is true, then action or the DO-part of the rule is evaluated to see if the action can be performed. Kodu executes these rule evaluations 50-100 times per

second. This means that there are no explicit iterations which govern the execution of the rules. Thus, Kodu is more similar to event driven programming languages where actions executed when event conditions are triggered (e.g., MIT App Inventor and autonomous robot programs). Thus, Kodu program execution is not sequential in nature.

Kodu further governs its way of reading and interpreting programs to what we call it as “Laws of Kodu Computations” or simply “Laws of Kodu” [89]. These laws basically determine how the rules are being interpreted by the Kodu’s compiler and how decision making is done. It is important for students to understand the Laws of Kodu Computation because it helps them to understand the conditions under which rules run, the order in which rules run, parallelism in program execution, pattern matching, and conflict resolution. There are four Laws of Kodu computation that students learn in the Kodu curriculum.

3.3.1 First Law of Kodu- “Each rule picks the closest matching object”

The 1st law explains that “Each rule picks the closest matching object” which means that every rule looks to match the object which is referred to in the When part of the Kodu rule. In the context of the pursue rule (e.g., When see apple, Do move toward), it means kodu is programmed to see a particular object (e.g., apples), then it will choose the closest matching object (e.g., apple) to the kodu character. Thus, if there are multiple objects of the same type (e.g., 5 apples) in a world, the 1st law of Kodu says that kodu will pursue the apple that is closest in distance to itself. Consider the following examples that students often encounter in the Kodu curriculum, let’s suppose there are blue and red apples in the world. If the kodu is programmed to pursue an apple (e.g., When see apple Do move toward it), it will go towards the closest apple irrespective of

their color. However, if the rule is to pursue the blue apple (e.g., When see blue apple Do move toward it), it will go to the closest blue apple. Mastery of this law helps students to predict the behavior of kodu when there are multiple objects in a world that satisfy the conditional criteria of a rule. In essence, the 1st law of Kodu (Fig 3-4 left) codifies how a computer executes pattern matching on each rule.

Figure 3-4. The First (left) and the Second (right) Laws of Kodu

3.3.2 Second Law of Kodu- "Any rule that can run, will run"

The 2nd Law of Kodu states that "Any rule that can run, will run." In Kodu, every rule is checked 50-100 times a second to see if it can run (having a true WHEN-condition) and if it runs, then the rule's action can be performed. Consider for example, that the kodu character is programmed with the simple pursue and consume rule (Figure 3-3) and is placed in a world full of apples. The pursue and consume rules will be interpreted by the Kodu rule editor in the following way. First the WHEN part of pursue rule (e.g., WHEN see apple) will be evaluated to true because there is at least 1 apple that satisfies the WHEN-condition of the rule which makes the rule eligible to run. Since the rule runs, the rule's Do action (e.g., DO move towards) is executed which will then allow the kodu character to perform its action. This results in the movement of kodu character towards the nearest apple (satisfying 1st Law). This pursue rule will

continue to evaluate as true until there are no more apples in the word for the kodu character to perceive (Figure 3-4 Right).

When the kodu reaches an apple and bumps it, the WHEN part of the consume rule (WHEN bumped apple) now evaluates to true allowing its Do action (DO eat it) to now run allowing the kodu character to eat the apple. Once the kodu eats the apple, it is no longer bumping the apple, so the consume rule's WHEN-condition (When bumped apple) evaluates to false until the next time the kodu character bumps an apple. Thus, consume rules are only eligible to run occasionally when the rules WHEN-condition is true. Whereas, the pursue rule runs continuously as long as matching objects specified in the pursue rules WHEN-condition are visible in the world (Figure 3-4, right). Now if we reverse the rules and have the consume rule as the first rule and pursue rule as the second rule (Figure 3-5), the resulting program behavior is the same as the regular pursue and consume rules because the consume rule can only run when the kodu character is bumping an apple and the pursue rule can run as long as there are apples in the world to eat. Thus, the 2nd law of Kodu says that the Kodu programming environment allows any rule that can run to run regardless of its order in the program. The 2nd law aims to develop students' reasoning about conditionals, parallel interpretation and execution of the rules rather than sequential.

Figure 3-5. Inverse pursue and consume rules

3.3.3 Third Law of Kodu- “When actions conflict, the earlier wins”

After students master the understanding of the 2nd law, it is essential for them to understand how the Kodu programming language manages conflict resolution in situations when there are two rules that are eligible to run (i.e., both rules have When parts that evaluate to true) but their Do actions cannot be executed simultaneously. The 3rd Law of Kodu explains that if two rules actions are in conflict, then the earlier rule or the rule which has the lower rule number will run first until the WHEN-condition of the lower numbered rule is no longer true. Usually this occurs when there are no more objects in the world that match its condition. Thus, the rule is exhausted. Consider, for example, a world where there are blue and red apples, and a kodu character has been programmed with three rules: rule 1 is a pursue-red-apple rule, rule 2 is a pursue-blue-apple rule (Figure 3-6, left), and rule 3 is a consume apple rule. The 3rd Law of Kodu magnet in Figure 3-6 on the left, highlights the first two rules of this program. Notice that the ‘WHEN’ conditions of the two pursue rules (rule 1 & 2) are true, however, only the ‘DO’ part of rule 1 is available to be executed. The 3rd Law of Kodu says that if two rules have conflicting actions as in this case “move toward a red apple” and “move toward a blue apple”, kodu will execute the earlier rule’s action (rule 1’s action) – “move toward a red apple”. It is important to note that while the 1st law decides which apple is closest to pursue for each rule, the 3rd law decides which type (color) of apple to pursue meaning which of the two rules will run in the case of conflict.

Figure 3-6. The Third (left) and Fourth (left) Law of Kodu

3.3.4 Fourth Law of Kodu- "An indented rule can run only of its parent can run"

The 4th Law introduces students to the concept of rule indentation and the condition when an indented rule will run. It explains that for an indented rule to run, its parent rule's condition must be true and its parent action should run. Thus, the 4th Law of Kodu highlights that the successful execution of the parent rule is necessary for an indented rule to evaluate its WHEN-condition and determine if its eligible to run.

3.4 Changes in Kodu Curriculum

The initial Kodu curriculum developed by Touretzky was based on self-discovery of the laws by observing kodu's behavior and observing it repeatedly without explicit labeling of the Laws as 1st, 2nd, 3rd, and 4th. Thus, students in our earlier Kodu studies were expected to recognize the laws of Kodu computation from observing repeated patterns of program execution behavior from running various programs throughout the curriculum. As a result, students passively discovered the laws on their own with little reinforcement from the curriculum. In this initial version of the curriculum, only the 1st Law of Kodu was the explicitly reinforced by curriculum activities that helped students to discover that "*Each rule picks a closest matching object*" but it was not explicitly named "the 1st Law of Kodu". The other three Laws of Kodu were not explicitly discussed or

reinforced with students. However, students' knowledge of the laws was implicitly assumed on some of the assessment questions.

In fall 2016, our analysis of students' performance on assessments led Touretzky to change the Kodu curriculum. This change was aimed to provide more support for students to help them understand the Laws of Kodu computation by explicitly naming and teaching the first four Laws of Kodu. Explicit teaching of the Laws of Kodu in the Kodu curriculum aims to build and improve students' reasoning ability because the self-discovery of laws alone was effective in teaching students the Laws of Kodu and had limited impact on their reasoning about program behavior. These first four Laws of Kodu discussed in this thesis, are printed on fridge magnets with a small example of a rule and kodu's movement based on the corresponding law (Figure 3-4, 3-6). The first two laws of Kodu also have YouTube videos which explain the laws and how they govern Kodu's execution of the rules. In addition, several Kodu worlds have been designed to help students apply the Laws of Kodu to reason about and explain the behavior of kodu programs. Currently there are more than 30 laws in Kodu, but the curriculum only covers first four laws [89].

After students are familiar with these laws explicitly and have had experience in demonstrating their use while creating and running simple Kodu programs, we expect them to reason about kodu's behavior based on the laws on assessment questions. We expect that on questions about how a program is being executed, students will refer to, state, and/or directly apply laws while reasoning about the program's execution. Assessment questions are asked at the end of the module which feature questions related to simple recognition and understanding of concepts. The questions assess

learning by asking students to trace, mentally simulate and predict the behavior of programs. In most questions, students are given a program, an image of the Kodu-world and four options to choose. Based on the responses on the assessments we can evaluate what students understand and identify where they have problems. Thus, by using well designed questions, we will be able to evaluate the extent to which students are reasoning computationally or not.

3.5 Computational Reasoning in Kodu

Computational Reasoning is the ability to read, write, trace, debug and predict program behavior. We believe that computational reasoning in Kodu is exhibited by students' use of design patterns and laws, to read and interpret programs and predict program behavior. The flashcards/design patterns/idioms tell kodu what to objects to focus on (e.g., pursuing and consuming apples). The laws govern the execution and interpretation of rules and determine in what order rules run and which objects in a world to focus on first.

As students progress through the Kodu curriculum, we expect them to successfully reason about programs by using their knowledge of the design patterns as demonstrated by their abilities to

- name the design pattern and recognize it when it appears in a Kodu program,
- recognizing syntactically correct design pattern (idiom) and interpret and understand their meaning and purpose,
- evaluate rules to determine whether they correctly implement a design pattern
- predict the program behavior based on design patterns, and
- instantiate a design pattern to generate rules if required.

In addition, we expect students to successfully reason about programs by using their knowledge of the laws to

- State the laws,
- Explain program behavior by referencing the laws,
- Trace programs using laws to correctly mentally simulate program behavior,
- Use the laws to predict future behavior from current program state, and
- Determine which laws are relevant to a given design pattern

This chapter described the Kodu curriculum, resources, and Laws of Kodu. It also outlined the expectations of student learning and reasoning about programs within the Kodu curriculum. Overall, we expect students who master the Laws of Kodu will be able to engage in pattern-matching to determine the objects in which each rule will match and the order in which objects will be visited (1st Law). We also expect that students are able to reason about the Kodu rules by evaluating the conditional value of the 'When' part of each Kodu rule and determine if the rule is eligible to run and, if eligible, what action the rules will take (2nd Law). We expect students to be able to recognize when rules are in a program conflict and how the Kodu compiler will resolve the conflict (3rd Law). Lastly, we expect students will be able to use their knowledge about the design patterns in Kodu and the first three Laws of Kodu to reason about and predict the behavior of two and three-rule Kodu programs on paper-based assessments.

CHAPTER 4
KODU PRIOR RESEARCH & OPEN RESEARCH QUESTIONS

Starting in summer of 2015, Dr. David S. Touretzky, a Research Professor at Carnegie Mellon University, Dr. Christina Gardner-McCune, an Assistant Professor at the University of Florida and I began collaborating on evaluating student learning as well as on testing and refining the Kodu curriculum and assessments. Over the last two years, we have conducted over ten Kodu after-school and summer-camp studies in Pittsburgh, PA, Greenville, SC, and Gainesville, FL. In total, our studies have enrolled more than 200 students.

Our overall analysis of student engagement in the Kodu curriculum tells us that students enjoyed engaging with Kodu Game Labs and curriculum activities. However, we also find that students have several common misconceptions that affect their reasoning about programs. Based on data collected from several of our Kodu studies, we have iteratively refined and redesigned the curriculum, instructional activities, and assessments. These changes have improved the quality of engagement and have minimized the inconsistency in instructions and learning expectations. While the curriculum has been refined over the two-year period, the basic framework of the curriculum (e.g., tiles, flashcards, major concepts) still remains the same from the original curriculum.

My role in the Kodu research project has been focused on analyzing the data collected from assessments, audio-video recordings and observations made across these Kodu studies. I have contributed to co-authored publications of the findings and results in the proceedings of ACM Special Interest Group on Computer Science Education (SIGCSE) 2016 [3, 89] and 2017 [2, 4, 90]. In this section, I will describe

several noteworthy findings from the prior Kodu research that have direct relevance to the work presented in this thesis.

4.1 Kodu Misconceptions and Fallacies

Based on analysis of some of the data from the ten Kodu studies conducted, we identified common incorrect responses to assessment questions which suggest that students have some misconceptions and fallacies that affect their ability to reason about Kodu program behavior.

4.1.1 Kodu Misconceptions:

We define misconceptions as assumptions based on prior knowledge or preconceived notions that negatively affect student's ability to reason about programs. We have seen consistent appearance of these misconceptions in student assessment responses. Presence of these misconceptions do not imply that students have not understood the laws or idioms, rather that prior student knowledge prevents students from successfully applying their knowledge of laws and idioms to new questions. For the current scope of work in the thesis, we discuss prior Kodu research on three types of misconceptions: Negative Transfer, Static Kodu, and One-Time-Rule Execution.

4.1.1.1 Negative transfer misconception:

We have found that students tend to analogically answer the assessment questions by drawing on their earlier experiences with Kodu program behavior and activities. Touretzky et al. discussed this phenomenon in their paper titled "Teaching Lawfulness with Kodu" [89], where students exhibited evidence of negative transfer on rule construction questions and state machine problems. Such negative transfer means that students often incorrectly refer to objects and scenarios in previous Kodu questions/activities not realizing that they are not in the current question and then use

this prior knowledge to reason about the current question. This type of reasoning means that students are using mental short-cuts that are not always applicable. This results in mistakes that prevent us from correctly evaluating their knowledge.

4.1.1.2 Static kodu misconception:

We have found that some students tend to ignore or not consider the dynamic positioning of a given kodu while simulating Kodu programs and predicting a given kodu's behavior. A kodu's dynamic repositioning during simulation directly impacts the path and order of objects that kodu pursues. Thus, student that hold a static kodu misconception will reason about a kodu's consumption of objects based on original fixed frame of reference presented in the question. This will result in an incorrect, unlawful, and often impossible path. We have observed the static kodu misconception in students' answers on many mental-simulation questions. This misconception will be discussed in more detail in Chapter 5.

4.1.1.3 One time rule execution misconception:

The rules in a Kodu program run in a continuous loop, similar to autonomous robot programs. Thus, each rule is checked 50-100 times per second evaluating and running eligible rules. We have found that some students think that rules only run once and they cannot run again. When students have the one-time-rule execution misconception, it negatively affects their ability to correctly simulating the program. For example, students might predict that the reverse pursue and consume idiom will result in a kodu only pursuing one apple and not consuming it because the rule could not run the first time the rules were evaluated.

4.1.2 Kodu Fallacies:

A fallacy is an incorrect reasoning pattern developed primarily as a result of earlier learning experiences or a misunderstood concept from instructions in the kodu session. Fallacies have an impact on how students reason and simulate programs. They can be identified by consistent use of an incorrect reasoning pattern drawn on an identifiable previous learning source. During our use of the Kodu curriculum with the students, we have found some common fallacies: Sequential Procedure Fallacy and Collective Choice Fallacy.

4.1.2.1 Sequential procedure fallacy:

We have often seen and heard students explain that rules in a program run sequentially. Students think that Kodu reads and executes the rules one by one starting from the first rule and proceeding to the last. Touretzky et al. discuss this fallacy and reasons why students develop this fallacy in their paper [90] explaining that the numbers on the rules in the kodu rule editor and assessment questions suggest to the students a logical ordering of program rules. This fallacy is harmful to students' reasoning because Kodu does not execute rules sequentially. We began teaching students the 2nd law of Kodu to help combat this misconception and to help them develop an understanding that kodu rules run in parallel and only run when they are eligible to run because their WHEN-conditions evaluate to true.

4.1.2.2 Collective choice fallacy:

After explicitly teaching students the three Laws of Kodu, we have noticed that instead of using the 3rd Law of Kodu to reason about programs where there is a conflict between two pursue rules. Students often use the 1st and 2nd Laws of Kodu to resolve the conflict by determining which object is closer in the world to the kodu character and

then assume that the rule corresponding to the closest object will run and not the earliest rule in the program. This suggests that students think that the kodu character collectively uses the two pursue rules to choose the closest object rather than resolving the conflict with the 3rd Law of Kodu. However, when students learn the 3rd law of Kodu, we try to help them to understand that when rules are in conflict (i.e., two rules have true WHEN-conditions but conflicting actions) the earlier wins. Touretzky et al. names this reasoning pattern as the Collective Choice Fallacy and discuss the findings of this fallacy in the paper titled “Semantics Reasoning in Young Programmers” [90].

4.2 Research Opportunities

For our research, we have worked on evaluating the effect of curriculum on student learning. We have been interested in students’ ability to understand Kodu rule syntax, to recognize and construct simple programs using the pursue-and-consume design pattern, and to understand, simulate, and predict the program’s behavior. We have used physical manipulatives like tiles and flashcards to scaffold student learning.

Our initial results suggest that students understand the pursue-and-consume idiom and can construct simple rules in the rule editor [86, 89, 90]. However, our conjectures on how students are interpreting and reasoning about simple and advanced mental simulation questions still needs to be studied explicitly to confirm our conjectures from a student perspective. Understanding mental simulation requires that we first have effective curricula and assessments which may accurately capture the students’ mental model of programs. Then we need to validate our conclusions from the perspective of students.

4.3 Thesis Research Question

The overall goal of this thesis is to understand to what extent elementary students are able to reason computationally**.** Specifically, this thesis aims to explore how elementary students develop this ability to reason computationally about programs, to validate misconceptions and fallacies that affect students' reasoning, and to design and use assessment questions to evaluate students' abilities to engage in computational reasoning.

In the study discussed in Chapter 5, we will examine my analysis of different groups of students' responses on the same mental simulation question with three different kinds of instructions. The goal of this study was to evaluate the extent to which our assessments successfully elicited students' ability to mentally simulate programs. This study helped in verifying if students were mentally simulating Kodu program behavior and to what extent we can capture it on paper-based assessments. This study also helped us to understand how students were comprehending instructions. In particular, this study helped us to remove aspects of the instructions that were misunderstood by students or that created confusion. Thus, the results of this study were valuable in designing and refining the mental simulation questions that we used in subsequent Kodu studies. The initial research on this study was presented as a poster [3] at ACM Special Interest Group for Computer Science Education, SIGCSE-2016.

In Chapter 6, we will discuss another study conducted which highlights the effectiveness of physical manipulatives like tiles and flashcards. The goal of this study was to evaluate the extent to which our curriculum and instructional strategies facilitate student learning. We were particularly interested in examining the role which the physical manipulatives play during the activities, and if they were helping the students to

correctly construct syntax (tiles) and recognize design patterns (flash cards). This study helped in understanding the critical affordances of the physical manipulatives and how to better adjust them for class activities and instructions. We were also interested in studying the best instructional techniques which assist students as they build mental simulation models. The findings of this study improved the curriculum and we were able to use assessments and curriculum activities using tiles and flashcards which fostered the understanding of rules and design patterns, and thus building the mental-simulation ability. The results of this study were presented as a paper [4] at ACM Special Interest Group for Computer Science Education, SIGCSE-2017. In order to study the extent to which elementary students were reasoning, it was important for us to use proven instructions first and then evaluate the progress of reasoning.

Based on the results of the ongoing Kodu project [89, 90], we were interested in studying cognitive development in the students. The curriculum earlier had the laws which were discovered by the students during the activities and demonstrations. This was changed and the new curriculum offered the student explicit laws to reason and interpret rules.

In the CS education literature, there is a lack of research focused on elementary school students' learning of CS concepts and computational reasoning. Thus, this research sought to explore the lack of research on how elementary students learn CS concepts and build computational reasoning ability. In keeping with this goal, we decided to design and conduct a think-aloud study where students first learn about the Kodu concepts like the design patterns, syntax and laws, and then they answer some questions. Students are then interviewed to collect data on how they are reasoning.

Laws were explicitly taught using the Kodu curriculum. Chapter 6 discusses this particular study and its findings. Our main research questions for the study were-

1. What kind of preconceived misconceptions and fallacies do students have about Kodu computation?
2. To what extent does explicit teaching of laws eliminate students' misconceptions and fallacies?
3. To what extent are students able to understand, apply and reason through laws?
4. In what ways do students misapply laws?

It should be noted that the three studies which are discussed in this thesis complement the ongoing research on the Kodu project. The Kodu project aims to study the development of formal reasoning ability in K-12 students using Kodu. Kodu supports features which allow students to reason about state machines fostering the development of formal reasoning. While we were studying students' learning, we became interested in understanding how specific parts of the Kodu curriculum and assessments were impacting student learning. This thesis parallels the iterative development of the curriculum. Chapter 5 and 6 discuss the changes made in the curriculum and assessment questions to support student learning. Once we refined the curriculum and assessments, we were able to study the misconceptions and challenges students were having in learning to reason about programs. Chapter 7 validates some of the misconceptions and fallacies found in our prior Kodu research; in addition, it discusses how explicit teaching of the laws helped to address students' misconceptions and fallacies. Chapter 7 also discusses how the laws provide the basis on which students are able to reason about programs. The findings across these studies help us answer the central question: to what extent are elementary students able to reason computationally?

CHAPTER 5
MENTAL SIMULATION STUDY

5.1 Introduction

Mental simulation is an important skill for program understanding and prediction of program behavior. Assessing students' ability to mentally simulate program execution can be challenging in graphical programming environments and on paper-based assessments. This study presents the iterative design and refinement process for assessing students' ability to mentally simulate and predict code behavior using a novel introductory computational thinking curriculum for Microsoft's Kodu Game Lab.

We present an analysis of question prompts and student responses from data collected from three rising 3rd - 6th graders where the curriculum was implemented. Analysis of student responses suggest that this type of question can be used to identify misconceptions and misinterpretation of instructions. Through this analysis, we also discuss patterns of students' mental simulation of simple Kodu programs. Finally, we present recommendations for question prompt design to foster better student simulation of program execution.

5.2 What is Mental Simulation?

Mental simulation and prediction of program behavior is an essential component of computational reasoning and students' ability to design and debug their programs. Thus, strengthening and assessing students' ability to engage in mental program simulation and prediction requires special attention while designing curriculum and assessments. As graphical programming language environments (e.g., Scratch, Agent Sheets, and Microsoft's Kodu) are increasingly used to teach elementary and middle

school students programming, there is a greater need to assess the skills students are learning with these environments.

5.3 Background

Figure 5-1 depicts Apple World, an introductory Kodu world. Figure 5-2, depicts the rule editor defining the rules for the Pursue and Consume idiom that student learn in the first module of the Kodu curriculum. This is a simple rule set that allows students to create autonomous agents in Kodu. When students run Apple World, they see the Kodu character navigate to the closest apple and eat it, repeating this process until there are no more apples.

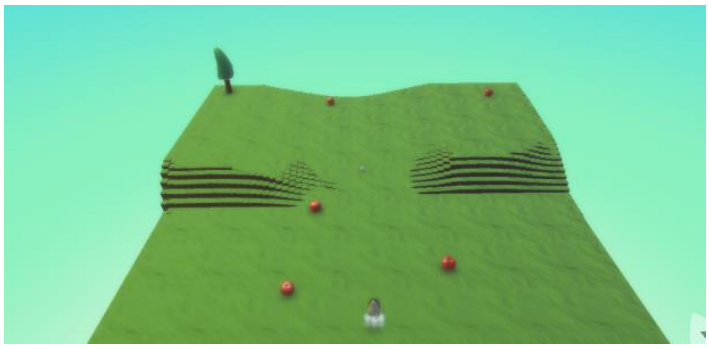


Figure 5-1. Apple World in Kodu

Figure 5-2. Pursue and Consume Rules

5.4 Study Design

Students were provided with the pursue and consume statements (Figure 5-2) and were asked to simulate and predict the sequence in which the apples will be eaten on a paper-based assessment where they were provided with the position of kodu and apples (Figure 5-3 left). Figure 5-3 (left) depicts a graphical map depicting scattered apples and a Kodu in which students record their answer to the question prompt. Three different kinds of instructions were iteratively piloted for the same simulation task (Figure 5-3) with different groups of students through three separate Kodu studies. This question assesses students' ability to mental simulate the behavior demonstrated by Kodu in Apple world (Figure 5-1) which students saw in their first Kodu session. Each group of students was given this question just after their first Kodu session when they had adequate knowledge of pursue and consume idiom and the 1st Law of Kodu.

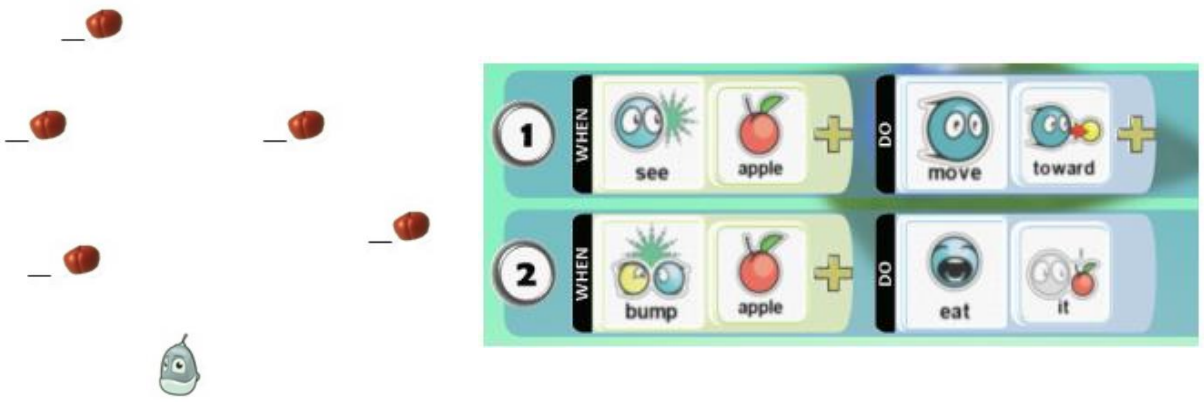


Figure 5-3. Graphical map depicting scattered apples and a kodu for the mental simulation question(left), pursue and consume rules (right)

5.5 Methodology

Data from students was collected through three studies: nine 3rd -5th graders, Thirty-seven 5th to 6th graders, and Thirty-four 5th to 6th graders. The responses were

qualitatively analyzed and question instruction prompts were iteratively improved with the expectation of eliciting better results from the students.

5.6 Findings

The three types of questions, type-I, type-II and type-III are compared and their respective results are following-

5.6.1 Mental Simulation Instruction Type-I

Mental Simulation Instruction Type-I (Figure 5-4) was designed to help students initially understand the dynamic positioning of Kodu and to help them think about how kodu's path changes based on the kodu's current position. This instruction prompted students to first identify the first apple, and then circle it and then further label the positions from 2 through 5. It had 3 bulleted points and the scaffolding technique to put "circle" was used to help students realize the dynamic position of kodu. We had 9 students who responded to Mental Simulation Instruction Type-I shown below (Figure 5-4). The text enclosed in the orange rectangle represents the scaffolding provided to assist students in communicating the dynamic positioning of the kodu character.

Here is a map showing the kodu and five apples.

- Which apple will it go and eat first? Write the number 1 to next to that apple.
- Draw a circle to show where the kodu will be *after* it eats its first apple.
- Write the numbers 2 through 5 next to the remaining apples to show the order in which the Kodu will go to and eat them.

Figure 5-4. Mental Simulation Instruction Type- I –Label Path and Circle Next Position & Label Path

Figure 5-5. Mental Simulation Instruction Type-1 Sample Responses: Correct response (left) and incorrect response (right) on type-I

5.6.1.1 Observations-

- 55% of students (n=5) correctly labeled kodu's path and only one student marked the circle correctly.
- 55% of students (n=5) mark the circle and out of those students only 40 % (n=2) mark circle correctly.
- 33%of students (n=3) did not mark circle still got the correct answer

5.6.1.2 Analysis-

Based on the results from Table 5-1, 2 out of 9 students correctly marked the circle and only 1 out the these 2 correctly answered the question. On the other hand, 1 out of 3 students who incorrectly marked the circle correctly answered the question. 3 out of 4 students who did not mark the circle correctly answered the question. This suggests that not many students were able to correctly use the scaffolding technique to mark a circle around the location of the kodu after it ate the first apple, and only one out of the two students (50%), correctly answered the question. 1 out of the 4 other students who answered correctly, did not mark the circle correctly and 3 students did not chose to mark the circle. Therefore, based on the observation, it can be said that the scaffolding technique of marking circle around kodu's location after eating the first apple was not useful to the students. Its use should have helped the students to mark the sequence correctly but that does not seem to be the case.

Table 5-1. Responses of 9 students who used Mental Simulation Instruction Type-I

n=9	Marked Circle Correctly	Marked Circle Incorrectly	Omitted Marking Circle	Total
Correct Labeling	1	1	3	5
Incorrect Labeling	1	2	1	4
Total	2	3	4	9

After reviewing student responses to the Mental Simulation Instruction Type-I, we decided to make the instructions more explicit and sequential as we thought that students were unable to understand what they were supposed to do. We expected the sequential and explicit instructions would provide clarity to students about what they were being asked to do and help to recognize the changes in kodu's position after pursuing each apple. The revised instructions resulted in Mental Simulation Instruction Type-II.

5.6.2 Mental Simulation Instruction Type-II

Students received Mental Simulation Instruction Type-II which had a list of four points sequentially instructing what to do in the response. Mental Simulation Instruction Type-II is shown below (Figure 5-6) and the text enclosed in the orange rectangle represents the scaffolding provided to assistant students in understanding the dynamic positioning of the kodu character.

Here is a map showing the kodu and five apples. Read the following instructions carefully:

a. Which apple will the kodu go to first? Write the number 1 next to that apple.
b. Draw an "X" to show where the kodu will be after it eats that first apple.
c. Once the kodu eats its first apple, which apple will it go to next? Write a 2 next to that apple, but don't draw any more "X" marks.
d. Write the numbers 3 through 5 next to the remaining apples to show the order in which the kodu will go to eat them.

Figure 5-6. Mental Simulation Instruction Type- II - Label Path & Mark "X" on Current Position & Label Path

This time, the instruction asked students to mark 'X' at the position where the kodu will be after it eats the first apple, and them prompted to write '2' next to the second apple which will be eaten without marking 'X'. In this way, we attempted to be extremely clear in what students were supposed to do.

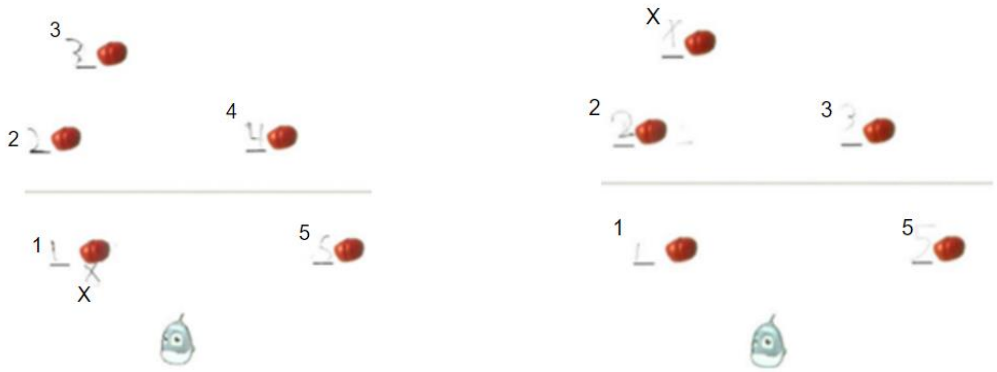


Figure 5-7. Mental Simulation Instruction Type-II Sample Responses: Correct response (left) and incorrect response (right)

5.6.2.1 Observations-

- 70% students (n=26) correctly labeled kodu's path and out of those students 38% (n=10) correctly marked "X" on current position.
- 70% of students (n=26) marked "X" on Kodu's current position and out of those only 46% (n=12) marked "X" correctly.

- 21% students (n=8) did not mark “X” but still correctly labeled Kodu’s path

5.6.2.2 Analysis-

Based on the results from Table 5-2, 12 out of the total 37 students marked the ‘X’ correctly and 10 answered the order correctly, 2 answered it incorrectly. 14 out of the 37 students marked the ‘X’ incorrectly and 8 out of these 14 students answered the sequence correctly. 11 out of the 37 students did not mark ‘X’, and 8 out of the 11 students answered correctly. This indicates that marking of ‘X’ does not help students in predicting the correct sequence.

Table 5-2. Responses of 37 students who used Mental Simulation Instruction Type-II

n=37	Marked “X” Correctly	Marked “X” Incorrectly	Omitted Marking	Total
Correct Labeling	10	8	8	26
Incorrect Labeling	2	6	3	11
Total	12	14	11	37

These results suggest that even explicit sequential instructions did not prove effective in helping students to understand the question. Thus, the use of scaffolding techniques to help students visualize the dynamic placement of the Kodu character after eating its first apple did not increase students’ awareness about the dynamic position of kodu. Therefore, these results led to the change in the instructions away from step-by-step instructions in the next goal.

5.6.3 Mental Simulation Instruction Type-III

After observing the results from Mental Simulation Instruction Type-I and Mental Simulation Instruction Type-II instructions, we decided to simplify the instructions and

simple them to trace the path of kodu which it will take to eat the apples. This instruction was meant remove any distractions that might have been caused by too many instructions and to simply require them to draw kodu's path. Mental Simulation Instruction Type-III is shown below (Figure 5-8) and the text enclosed in the orange rectangle represents the scaffolding provided to assistance students in communicating the dynamic positioning of the kodu character.

Here is a map showing the kodu and five apples. Based on your understanding of how the kodu moves, draw the path the kodu will take to eat all the apples. Start by drawing a line from the kodu to the first apple, then extend the line to the next apple, and so on. Finally, write the numbers 1 through 5 next to the apples to show the order in which they are eaten.

Figure 5-8. Mental Simulation Instruction Type- III - Trace Path & Label Path

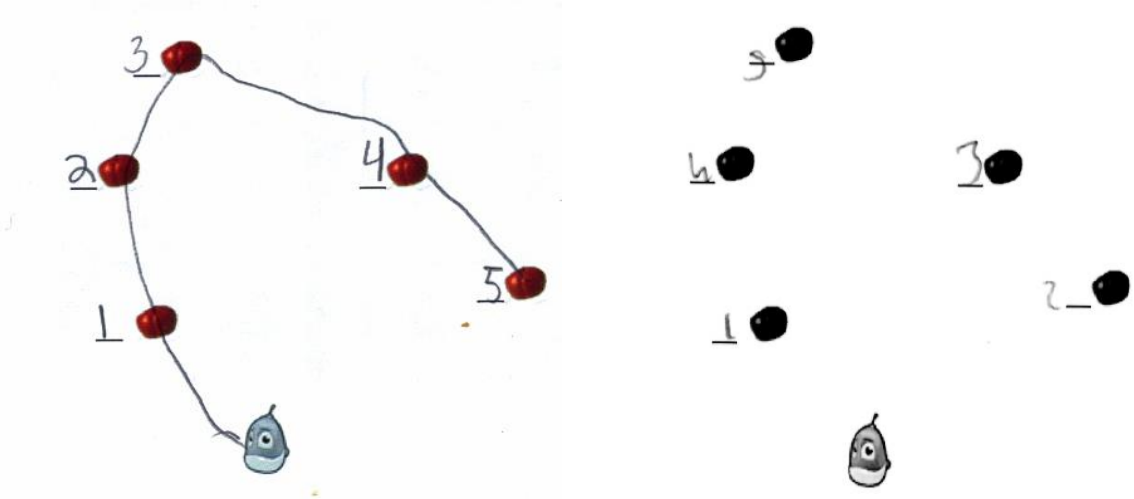


Figure 5-9. Mental Simulation Instruction Type-III Sample Responses: Correct response (left) and incorrect response (right)

5.6.3.1 Observations-

- 94% students (n=32) correctly labeled kodu's path and out of those 68% (n=22) traced the path correctly.
- 64% students(n=22) traced the path and 100% of those students traced the path correctly.

- 29% students did not trace path but still correctly labeled Kodu's path.

5.6.3.2 Analysis-

Based on the results of Table 5-3, 22 out of 34 students correctly traced and answered the question. There were no instances of incorrect tracing reported. 10 out of the 12 students who omitted any kind of tracing also answered correctly. These results provide support that tracing was an effective scaffolding technique as the students who did use it answered correctly. Thus, providing a high-level goal-based scaffolding technique like tracing helped students to comprehend in what they were really asked to do.

Table 5-3. Responses of 34 students who used Mental Simulation Instruction Type-III

n=34	Correct Tracing	Incorrect Tracing	Omitted Tracing	Total
Correct Labeling	22	0	10	32
Incorrect Labeling	0	0	2	2
Total	22	0	12	34

Thus, Mental Simulation Instruction Type-III which featured a brief goal-based instruction prompt, was found to be more effective as compared to earlier Mental Simulation Instruction Type- I and Type-II prompts. Based on the results of this study, we permanently integrated the Mental Simulation Instruction Type-III prompts into the curriculum.

5.7 Discussion

5.7.1 Mental Simulation Instructions

The mental simulation question which was used for this study was one of the first questions we asked students on their first Kodu assessment. We anticipated that students would have challenges in observing the dynamic positioning of kodu while making decisions about kodu's path. Thus, we wanted to provide a scaffolding strategy which helped students to overcome the potential assumption of static kodu. In the prompt for Mental Simulation Instruction Type-I, we had a sequence of instructions and asked students to mark circles when the kodu eats it first apple. Based on the results we can say that this scaffolding technique was not very helpful to students, and some of the wording might have confused students. The prompt for Mental Simulation Instruction Type-II provided more structure and sequencing through its step-by-step guidance on what students were expected to do. Again, based on the results we can say that this instruction prompt was not helpful and some of the instruction, especially point c) of Mental Simulation Instruction Type-II which explicitly asked students to determine the second apple which the kodu will eat, but don't draw any more "X" confused students. Moreover, the results suggest that this instruction was interpreted differently by different students. Some marked "X" on the second apple, some marked on others.

Finally, it seems that the tracing required by Mental Simulation Instruction Type-III's prompt, to draw the entire path of kodu was simpler and easier for students to understand. Especially, since it facilitated a continuous path drawing process throughout problem solving process. Unlike the piece-wise process facilitated by marking a circle or "X" on the first apple, students could use tracing technique from the very start till the last apple. We initially thought that exact sequential prompt structure

created by bullets or numbered list would help students understand and follow instructions. However, it seems like most students were not able to follow them. Most students found the structured instructions too challenging to comprehend.

5.7.2 Mental Simulation Instruction Recommendations

Excessive rule breakdown does not always scaffold student reasoning, rather it can create confusion about what students are supposed to do.

Suggestion:

- use clear and easy to comprehend instructions.
- Minimize ambiguity in instructions and scope of instruction interpretation
- For providing a scaffolding technique for mental simulation questions, choose a coherent and consistent technique (like tracing here), which encompasses the entire question and is not limited to only part of the task scope (mark “X” or circle on just the first one).

5.7.3 Misconception: Static Kodu

In analysis of the three mental simulation instruction types, we found that student responses demonstrated simulation errors. Often student responses incorrectly predicted the path of kodu based on the initial position of kodu. Each time kodu pursues and eats an apple, its position changes and the next closest apple is now dependent on its current position. For students, the earliest stage in the development of mental simulation ability is recognition of this dynamic positioning of kodu.

We also found this error in students’ responses to similar mental simulation questions asked to students in later modules of the curriculum.

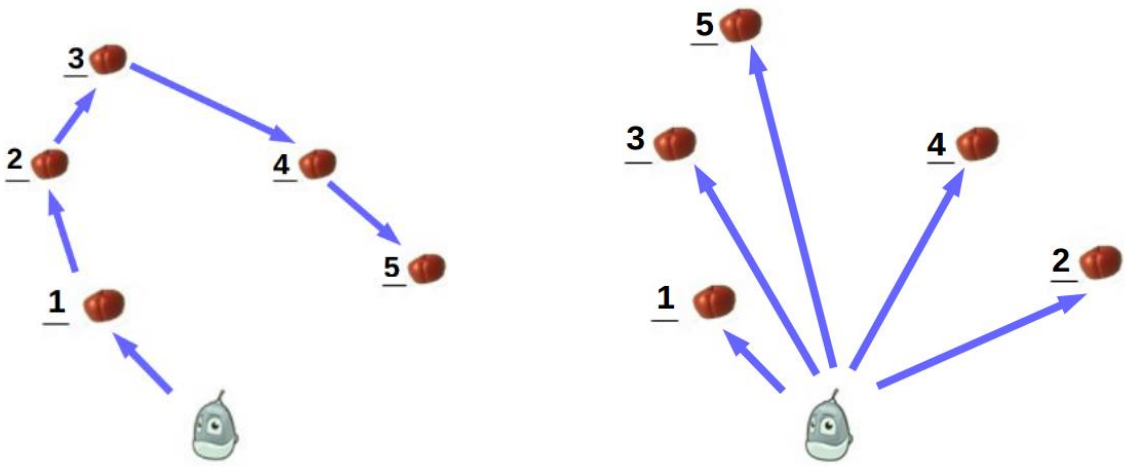


Figure 5-10. Correct simulation(left) and incorrect simulation based on static kodu (right) on type-III

5.8 Conclusion

For our research, we rely on data collected through paper-based assessments and we have learned that it is important to align the question instructions with the way students solve problems in the classroom. This is consistent with the suggestions of Shaffer et al. [76] which discusses the need for assessments to be aligned with what and how students learn content. The revisions we made to the Mental Simulation Instruction prompts, allow us to attribute any error in a simulation response to students' learning and not to their possible misunderstanding of the question's instructions.

This study helped us to understand that we can capture students' mental simulation models through responses on paper-based assessments. The results of this study confirm the static kodu model of reasoning and illustrates that it hinders students' development of successful computational reasoning ability. The results of this study also indicate that if a student knows the 1st law and the pursue and consume idiom, but does not recognize the dynamic positioning of kodu, then he/she will incorrectly simulate the program. Thus, correct mental simulation requires the ability to read rules

and interpret the behavior based on the laws while simultaneously recognizing that kodu changes its position. This study suggests that the ability to mentally simulate programs plays a significant role in determining how students reason about programs.

CHAPTER 6
PHYSICAL MANIPULATIVE STUDY

6.1 Introduction

The goal of this study is to evaluate the extent to which our Kodu curriculum supports learning and instructional strategies facilitate the development of computational reasoning. We particularly want to explore the use of tiles and flashcards by 3rd – 5th grade students within a Kodu curriculum [87]. The results of this study will help in further understanding and assessing the development of computational reasoning ability by using verified instructions and use of physical manipulatives. This chapter on based on the paper published at the ACM Special Interest Group Computer Science Education, SIGCSE-2017 conference [4].

6.2 Background

The development of curricula and resources to help K-12 students learn computer science fundamentals include activities, assessments, projects, and teaching tips. In many classrooms, teachers use CS Unplugged activities that commonly feature kinesthetic and interactive elements in order to help students understand the concepts [15]. Several curricula have developed quick reference guides [68, 70, 83] and flashcards [8, 72, 88] to help students quickly identify key concepts or algorithms for implementing common programming design patterns (e.g., save data to a variable, program an autonomous sprite, or draw a shape). These resources help students to develop their understanding of computational principles away from the distractions of the programming environment and give them tangible ideas for creating meaningful artifacts [46, 86].

While the use and development of physical manipulatives in CS is growing, physical manipulatives have been around for years in other disciplines. Physical manipulatives like Cuisenaire rods and algebra rods have long been recommended and used in K-12 Mathematics. These tools help students to learn mathematical concepts by making the leap from “intuitive to logical thinking, from the concrete to the abstract” [43]. The affordances of these physical manipulatives, such as size and shape, influence how students use these tools and how they develop conceptual understanding and chunking strategies [58]. In literacy education, “physical manipulatives are physical objects that aid understanding of concepts or processes by allowing students to physically demonstrate and see the concept or process. The use of manipulatives provides a way for students to learn concepts in a developmentally-appropriate, hands-on, experiential way” [24].

As CS educators continue to design and refine physical manipulatives to improve students’ CS learning, it is important to develop an understanding of how students interact with these resources, how to best use these resources to support student learning, and how to measure the impact that these resources have on student learning.

We aim to measure the impact of tiles and flashcards [88] on students’ ability to understand, recognize, construct, and use game programming design patterns. Based on our analysis, we make recommendations for the optimal use of these resources to support CS learning and skill development.

6.3 Experiment

We hypothesized that the use of physical manipulatives such as tiles and flashcards improves students’ performance in understanding and recognizing design patterns and properly constructing rules in Kodu Game Lab.

Our goal for this study was to explore the effectiveness and usefulness of tiles and flashcards in the Kodu curriculum relative to paper-based alternatives. The paper-based alternatives remove the manipulative nature of these resources while keeping the curricular content intact in order to isolate the impact of the use of the manipulatives on students' ability to understand, recognize, construct, and use Kodu design patterns.

This intervention was designed to model the recommended usage of the curriculum in the classroom by recreating the usage conditions and preserving the learning from these resources. In this condition, printed versions of flashcards and WHEN-DO tile templates were provided to the students.

The work presented in this paper was a mixed methods research study. We used a between-subject study design to isolate the use of physical manipulatives versus paper-based alternatives. Our independent variable was the use or non-use of physical manipulatives. Our dependent variable was student performance on the Module 1 Assessment. We controlled for the instructional time, location, curriculum activities, and instructor. Random variables that we could not control for were the students' states of mind and prior programming experiences. The students were randomly divided into two groups and assigned the following conditions:

The Group A students (with physical manipulatives): were given the Kodu tiles and flashcards and were provided with verbal instruction on how to use them as described in the curriculum (Figure 6-1 left).

The Group B students (without physical manipulatives): were given 8.5 x 11 sheets of paper with color prints of the first two design patterns that were relevant for the learning activities. They were also provided with visual representations of the plastic

tiles that were printed on black and white 8.5 x 11 sheets of paper (Figure 6-1 right). In both groups, each student was also given a laptop and an Xbox controller for the experiment.

Figure 6-1. Test Conditions -Tiles and Flashcards -Group A (left) & Paper Constructs - Group B (right)

6.4 Participant's and Procedure

The students were recruited from the same elementary school and were randomly divided into two groups, A and B. Group A had five students: 3 third graders, 1 fourth grader, and 1 fifth grader; 2 girls and 3 boys. Group B had 6 students: 1 second grader, 2 third graders, 2 fourth graders, and 1 fifth grader; 2 girls and 4 boys. None of the students indicated any prior programming experience, except for one student in Group A who had used Minecraft.

Each group participated in two 90-minute sessions conducted after school on a Tuesday and Thursday in a given week. The goal was to complete the two curriculum modules: Module 1: Pursue and Consume and Module 2: Color Filters. Prior research suggests that students benefit most from the tiles and flashcards during these introductory modules [86]. Thus, we limited this study to the first two modules of the Kodu curriculum.

In the first session, the students were introduced to Kodu and to the first module of the curriculum, which focused on the Pursue and Consume design pattern. In the second session, they were given a refresher activity to remind them of the concepts learned in the previous session. Then they were given an assessment to evaluate their knowledge of Module 1. Finally, they were introduced to Module 2 and asked to complete an assessment for that module. All sessions were led by the first author, aided by two teaching assistants who helped the students as needed.

6.5 Data Collection and Analysis

We collected student pre- and post-surveys, student end-of-module assessments, student artifacts, and researcher field notes. The surveys and assessments were paper-based. The researcher field notes included time spent on session activities, students' overall engagement, use of tiles, flashcards, and alternatives, and interaction with Kodu.

We analyzed the data using quantitative and qualitative techniques. The students were given two assessments during the study. However, due to timing, Group A was unable to complete the Module 2 assessment before the end of the second session. Thus, we only present the results from Module 1, which had 13 questions.

These 13 questions focused on understanding, recognition, and construction of the Pursue and Consume design pattern. We analyzed the data from Module 1 in two ways. First, we used Bloom's Taxonomy [39] to measure concept understanding and skills targeted by the Kodu curriculum and assessments. This taxonomy was used because it considers all the levels of cognitive understanding of the materials covered in Module 1. For the quantitative analysis of the Module 1 assessment, each of the 13 questions were categorized by the authors according to the six levels of Bloom's

Taxonomy: Remembering (1 question), Understanding (4 questions), Applying (1 question), Analyzing (3 questions), Evaluating (2 questions), and Creating (2 questions).

In the second analysis, we focused on evaluating the impact of the use of manipulatives on skill and knowledge development. In particular, we qualitatively explored questions that provided evidence of the students' ability to recognize design patterns, construct proper rules for design patterns, and demonstrate concept understanding by use of the flashcards and visualization of program execution in 3D Kodu worlds. This analysis helped provide an additional perspective on our data that was not available from Bloom's Taxonomy analysis. We found this level of analysis to aid us in understanding the conditions and affordances of manipulative use that were beneficial or non-optimal.

6.6 Findings

The students in both groups demonstrated varying levels of mastery of Module 1 concepts. However, overall, the students answered roughly the same number of assessment questions correctly. In Group A (with physical manipulatives), 4 out of the 5 students individually answered 6 to 9 questions out of 13 correctly. The fifth student in this group answered only 2 questions correctly. This resulted in an overall total of 50% correct answers for Group A. All 6 students in Group B (without physical manipulatives) individually answered 6 to 10 questions out of 13 correctly with an overall total of 64% correct answers for the group. When looking at the levels of Bloom's Taxonomy, however, we can see variations in students' learning across groups.

6.6.1 Similar Performance Between Groups:

In both groups, most students answered the Remembering question correctly: 80% (4 of 5) in Group A (with manipulatives), and 83% (5 of 6) in Group B (without

manipulatives). This suggests that the students did learn the Pursue and Consume design pattern irrespective of whether they used physical manipulatives or not.

6.6.2 Slightly Differing Performance Between Groups:

In this category, the differences between the groups were worth exploring. For the four Understanding questions, students in Group A (with manipulatives) produced in aggregate 8 correct responses out of 20 (40%), while students in Group B (with manipulatives) produced 15 correct responses out of 24 (63%). Similarly, for the three Analyzing questions, Group A produced 8 correct responses out of 15 (53%) while Group B produced 12 correct responses out of 18 (67%). In the Evaluating questions, Group A answered 40% (4 out of 10) correctly, and Group B answered 58% (7 out of 12) correctly. The differences between the groups indicate that Group A did not perform as well as Group B on the Understanding, Analyzing, and Evaluating questions

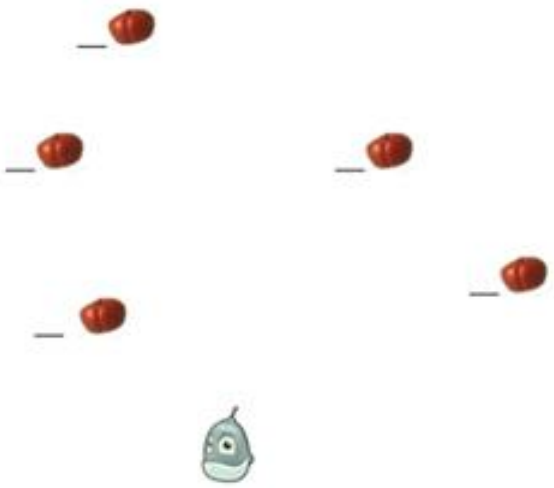
6.6.3 Drastically Differing Performance Between Groups:

We also found that Group A (with manipulatives) significantly underperformed Group B (without manipulatives) on the Applying question, but the reverse was true for Creating questions. For the Applying question, Group A produced 1 correct response out of 5 (20%), while everyone in Group B answered the question correctly (100%). For the Creating questions, Group A produced 7 correct responses out of 10 (70%) while Group B produced only 5 correct out of 12 (42%).

In summary, using Bloom’s Taxonomy, we found that Group A performed much better on the Creating questions, while Group B performed much better on the Applying question and slightly better on the Understanding, Analyzing, and Evaluating questions.

Here are the rules for eating apples:

Here is a map showing the kodu and five apples. **Draw the path the kodu will take** to eat all the apples. Start by drawing a line from the kodu to the first apple, then extend the line to the next apple, and so on.



Now write the numbers 1 through 5 next to the apples to show the order in which they are eaten.

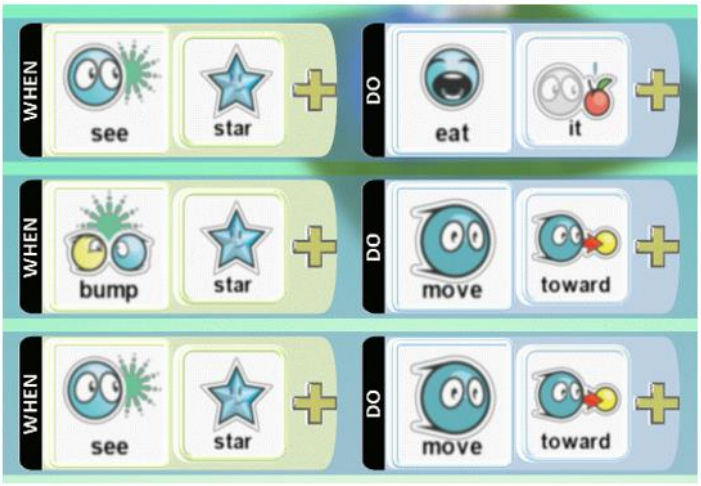
Figure 6-2. Q2, which was categorized in the 'Applying' category of the Bloom's Taxonomy.

Finding stars

Put an "X" next to the rule that tells the kodu to go to the nearest star:

Q3.

1. _____
2. _____
3. _____



Q4. Is this a pursue rule or a consume rule? _____

Figure 6-3. Q3 and Q4, which were categorized in the 'Understanding' category of the Bloom's Taxonomy.

Eating Candy Hearts

Suppose we want the Kodu to visit and eat all the candy hearts. Choose words from the list below to fill in the blanks to make the two rules we need to visit and eat the hearts:

see	**bump**	
apple	**heart**	**star**
eat	**move**	
toward	**it**	

[1] WHEN _____ _____ DO _____ _____

[2] WHEN _____ _____ DO _____ _____

Figure 6-4. Q7, which was categorized in the 'Creating' category of the Bloom's Taxonomy.

6.7 Pursue and Consume Understanding

To better understand the differences between these two groups, we analyzed the concepts and skills that students learn in the Kodu curriculum: recognition of design patterns, proper rule syntax construction, concept understanding, and simulation. We expected the use of physical manipulatives to have a direct impact on the students' learning of these concepts and skills.

The primary concept in Module 1 was the P&C design pattern, which instructs the Kodu character to move toward the nearest apple and eat it, then repeat that action until all of the apples are eaten. In order to test the students' understanding of this concept in the assessment, they were given three separate questions about the P&C rules to independently examine their understanding. Q3 and Q4 (see Figure 6-3) were based on Pursue. Q3 asked the students to select the correct rule out of the three possible rules by which the Kodu character can move toward an apple. Q4 asked the students to identify the name of the movement, choosing between 'pursue' and 'consume'. Similarly, Q5 and Q6 were based on Consume. Q5 asked the students to select the correct rule that would make the Kodu character eat an apple once it bumped into an apple. Q6 asked the students to identify the name of the action with the same three options available in Q4. Q7 asked the students to write the rules for a Kodu character to Pursue and Consume candy hearts using a fill-in-the-blank format (Figure 6-4). The students were given a word bank of Kodu tile labels to fill into the respective WHEN-DO rules. The first statement tested students' Pursue understanding, and the second tested their Consume understanding.

The results from Q3 through Q7 revealed that the two groups were similar in concept understanding: 50% of Group A (with manipulatives) responses, and 55% of

Group B (without manipulatives) responses were correct. This suggests that the students gained similar P&C design pattern understanding, regardless of the use of physical manipulatives.

6.8 Proper Rule Recognition and Construction

One of the expectations for the students' use of tile manipulatives in the Kodu curriculum is for them to be able to recognize and construct design pattern rules correctly. While the students in this study gained similar levels of understanding of the Pursue and Consume design pattern, an analysis of the questions that were focused on proper rule construction revealed differences between the groups. We used Q3 and Q5 to measure the students' ability to recognize proper syntax of the Pursue and Consume rules, since the students' ability to construct proper rules was influenced by their use of tiles (with manipulatives) and the rule editor (with and without manipulatives). The questions related to recognizing proper rule syntax also represent a subset of questions from the Understanding level of our Bloom's analysis. We used Q7 Parts 1 and 2 (see Figure 6-4) to measure the students' ability to construct Pursue and Consume rules. These questions were categorized as Creating questions in the Bloom's analysis.

While both groups were able to recognize rules equally (50% each), they differed in rule construction. The Group A (with manipulatives) students correctly answered 70% of the questions, while the Group B (without manipulatives) students correctly answered 42% (see Table 6-1). This suggests that Group A performed better than Group B on rule construction, which we believe was caused by the Group A students' use of tiles.

Table 6-1. Analysis of Proper Rule Syntax

Skill	Question Number	Group A (n=5)	Group B (n=6)
Recognition (Tiles & Flashcards)	Q3	2/5	2/6
	Q5	3/5	4/6
Overall Recognition	Q3 & Q5	50%	50%
Construction (Tiles)	Q7 Part 1	4/5	3/6
	Q7 Part 2	3/5	2/6
Overall Construction	Q7 Parts 1 & 2	70%	42%

6.9 Concept Understanding with Flashcards

The Kodu curriculum was designed for students to use flashcards in order to learn design patterns and refer back to them while completing learning activities. We observed that the Group A students actively used the flashcards during the activities. For this analysis, we selected questions that evaluated students on their conceptual understanding of Pursue and Consume (P&C) as described on the flashcards and paper-based equivalent. In this way, the selected assessment questions allowed us to measure the impact of flashcard use on concept understanding.

Q1 tested the students' ability to recognize the direction in which the Kodu character would move based on the Pursue and Consume (P&C) rules provided. Q4 (Figure 6-3) and Q6 (not shown due to space constraints), by contrast, tested the students' ability to correctly identify the name of each rule associated with Kodu character's action as described in the question.

Group B (without manipulatives) performed better than Group A (with manipulatives) on these three recognition of Pursue and Consume questions. The

Group A students correctly answered 46% of the questions, while the Group B students correctly answered 77% (see Table 6-2). This suggests that interaction with Kodu's 3D worlds may have helped the Group B students to develop their ability to recognize the actions associated with the P&C concepts.

Table 6-2. Concept Understanding Based on Flashcards

Skills	Question Number	Group A (n=5)	Group B (n=6)
Recognizing Pursue and Consume	Q1	4/5	5/6
	Q4	1/5	5/6
	Q6	2/5	4/6
Overall P&C Recognition	Q1, Q4, Q6	46%	77%

6.10 Simulation

Another expectation of the Kodu curriculum is to develop students' ability to mentally simulate and predict program behavior [89]. In this study, mental simulation of Kodu rules was gauged using two types of questions. These assessed the students' ability to mentally simulate (1) the basic P&C design pattern, which required only knowledge of the pursue and consume rules; and (2) intermediate P&C programs using multiple pursue or consume rules.

On the basic simulation questions (Figure 6-2), we found that the Group A students (with manipulatives) correctly answered 50% of the questions, while the Group B students (without manipulatives) correctly answered 91% of questions. On the intermediate simulation questions which required the program prediction ability, Group A (with manipulatives) students correctly answered 43% of the questions, and Group B (without manipulatives) students correctly answered 69% of the questions (Table 6-3).

This suggests that the development of mental simulation abilities require more interaction with the rule editor and the 3D visualizations of the rules, as experienced by Group B.

Table 6-3. Analysis of Simulation

Skill	Question Number	Group A (n=5)	Group B (n=6)
Basic P&C Simulation. Knowledge	Q2, Q8	5/10	11/12
Intermediate P&C Sim. Knowledge	Q9-12	8/20	14/24
Overall	Q2, Q8-12	43%	69%

6.11 Observation

We hypothesized that the use of physical manipulatives such as tiles and flashcards improves student performance. However, we found that while students who used manipulatives did better on rule construction, those who did not use the manipulatives did better on simulation and overall understanding of the concepts.

We turned to our observation and field notes to better understand the differences between the two learning methods in the classroom.

The students of Group A (with manipulatives): extensively used tiles before every Kodu activity that they were asked to complete. They constructed rules using the tiles before constructing them in the rule editor. Some of the students referred back to the flashcards while completing other learning activities, primarily focusing on the back-side of the flashcards, which had the Pursue and Consume rule syntax. Whenever the students were in doubt, they checked the syntax of the tiles using the flashcards and also consulted the instructors before finally putting the rules into the rule editor. Discussions between the instructors and the students were often lengthy, as the

instructors helped the students to understand why their rules were not logical, or were inconsistent with the type of rule construction that was needed to complete the activity.

The students of Group B (without manipulatives): initially used the paper constructs of tiles and flashcards, but they were reluctant to continue using them beyond the initial group activity. After interacting with the rule editor, the paper-printed tile rules were not used by the students; rather, the students used the Kodu rule editor to directly construct the rules. Students would make their solutions for the activity in the rule editor, run their worlds and make observations, then iteratively change their solutions until they accomplished the required task. After a couple of iterations, the students often called over the instructors without referring back to any material available to them (i.e., the paper based alternative to flashcards and tiles). But while the students were explaining their problems to the instructors, they often identified their own issues before an instructor could assist them.

6.12 Discussion

The results of this study indicate both the benefits and the drawbacks of using physical manipulatives in different learning situations. The students who used physical manipulatives were better at rule construction than the students who did not use physical manipulatives. This might have been because the students who used tiles before completing the activities in the rule editor developed a more refined understanding of the proper rule syntax of Kodu design patterns resulting in better reading and writing ability of program. We believe that the students who did not have manipulatives focused more on completing the activities iteratively, as they received more dynamic feedback from the programming environment. In contrast, the students with manipulatives focused on constructing the syntax with tiles without getting the

same dynamic feedback that the students without manipulatives received. Thus, this difference in feedback altered the direction of focus for the two groups while they completed the learning task. Group B was primarily focused on completing the activity task, and Group A was focused on constructing rules for that activity using the tiles and flashcards. This suggests that extended use of tiles may have diminishing returns, as it can be time-consuming to construct the correct syntax without dynamic feedback from the programming environment. These findings also suggest that the Group A students' intense focus on constructing rules also limited their usage of the flashcards to understand the general concept of Pursue and Consume. Therefore, we recommend that the use of tiles should be limited to introducing students to proper rule syntax and construction and to explain more complex syntax configuration, such as indentation.

The students who did not use tiles (Group B) may have acquired a more nuanced understanding of how the rules are executed through completing the activities through trial and error interactions in the programming environment. We believe that the reinforcement provided by dynamic feedback and the visualization of rule execution in the programming environment helped these students to develop the ability to mental simulate rule execution. The students with manipulatives had limited interaction with the rule editor due to time constraints, resulted in (1) limited exposure to dynamic visualization of rule execution and (2) limited recognition of behavior caused by errors in syntax.

Thus, the findings of this study suggest that iterative development in the programming environment helps students to observe and visualize Kodu's dynamic

behavior, which impacts students' ability to simulate and predict Kodu's behavior on paper-based assessments.

6.13 Takeaways from the Study

This study helps in establishing the relationship between the skills fostered by manipulatives and the different components of computational reasoning. Advance reasoning can be cultivated with proper use of physical manipulatives and the real-time interaction with the Kodu rule editor. Physical manipulatives scaffold the learning and assist in introducing the computational framework to the students who have had to significant or any kind of experience in programming.

- Tiles and flashcards help students develop rule construction and idioms/design pattern recognition abilities. This helps in developing the reading and writing component of computational reasoning.
- The experience with the Kodu rule editor is on trial-error basis which dynamically simulate rules provides a mental model for dynamic movement of kodu.
- Students' experiences, entering the kodu idioms into the Kodu rule editor provides opportunities for them to see dynamically simulated rules and to develop mental model for dynamic movement of kodu.
- This iterative process creates awareness of dependency of the syntax and causality of WHEN-DO conditions which help in tracing and program predating abilities.

6.14 Implications

In this study, we discover how different kinds of activities help in building specific aspects of computational reasoning. For example, the use of tiles help in fostering syntax reading and writing part of computational reasoning, and engaging with the virtual rule editor helps in development of mental simulation ability.

In addition to helping us gain a better understanding about physical manipulatives, the results of this study helped us to revise the curriculum activities in the

classroom. This study has provided us with instructions which have been proven to be effective in facilitating the development of computational reasoning and mental simulation skills.

6.15 Limitations

This study has the following limitations: Number of Students: Both, Group A and Group B had a small number of students because of the qualitative nature of the evaluation and other logistical constraints. This limits our ability to generalize these results over a large population using similar materials. Length of Study: The study had only two 90-minute sessions with each group. This provided students with a relatively short amount of time to learn and explore these concepts. In addition, the results presented in this study were based on only the first module of the curriculum. For more nuanced results, longer instructional time is needed to allow students to practice and learn the concepts and to see if these results can be replicated.

6.16 Conclusion

This study has implications for researchers and practitioners who are working on developing K-12 CS educational curricula and resources. Our results show that students make active use of flashcards when they are learning new concepts. Our results also show that selective and strategic use of physical manipulatives such as tiles can foster the development of rule construction. However, if the use of these manipulatives is not monitored, students can spend their time unproductively at the expense of reinforcing their conceptual understanding, which is fostered more deeply through students' iterative interaction with the rule editor.

CHAPTER 7
THINK-ALOUD STUDY

7.1 Introduction

Our prior Kodu work identified several consistent patterns of student misconceptions and fallacies, which we derived from students' performance on post-assessments [89, 90]. However, in these studies, we had no way to verify and validate the causes of the common errors. Thus, we were interested in understanding the source of these errors. We surmised that there were two plausible causes: faulty instruction (curriculum & teacher error) and/or faulty mental models or preconceived notions on the part of the student. We knew we needed to talk to students to better understand their experiences in our curriculum, interpretation of instructions, and reasoning behind answers they game to assessment questions.

In addition, we were interested in verifying our earlier work on mental simulation [3]. Thus, we decided to conduct a think-aloud study, where we planned to observe a student's reasoning and validate our earlier results. In particular, we wanted to capture and observe a student's reasoning and understand how and with what knowledge they were reasoning. We were also interested in better understanding how students were reading the instructions, interpreting Kodu laws and design patterns, simulating the rules and predicting the behavior of kodu.

7.2 Study Design

We aimed to explore the two plausible causes for poor performance on post assessments and the underlying causes of student misconceptions and fallacies: faulty instructions (curriculum & teacher error), and erroneous student mental models or preconceived notions. Thus, we designed our study to have two main components: (1)

an instructional intervention where we taught students the Kodu curriculum and observed their reasoning during class sessions and (2) a think-aloud interview where students either retrospectively explained their reasoning or verbally walked through their reasoning as they solved problems with the interviewer. The results of this study will allow us (1) to evaluate the influence of explicit teaching of laws in reducing faulty reasoning and improving the correct reasoning acumen and (2) gain a better understanding of student reasoning

7.2.1 Intervention: Instructional Approach & Curriculum

By the time we conducted this study, the overall instructional approach of the Kodu curriculum had shifted away from a dominant focus on teaching the design patterns (e.g., pursue and consume) using the flashcards and toward explicit teaching of laws to promote students' ability to understand and predict Kodu program behavior. As result, additional instructional resources such as Kodu law magnets, videos, and activity worlds had been developed and incorporated in the curriculum to help student explore and reason with Kodu laws. In addition, slight changes were made to the curriculum by incorporating results from our findings from study #1 in which question instructions on mental simulation questions were revised to allow students to better demonstrate their ability to simulate the path of kodu when executing a simple program.

Using results from Study #2, we shifted our instructional approach to emphasize student use of the Kodu programming environment to enter and run rules to see the resulting behavior. In addition, we have added kinesthetic activities in which students are presented with a set of rules and they make predictions about kodu behavior by pretending to be kodu characters in the physical world. In this way, students demonstrate their understanding of the rule execution. They then were asked to verbally

explain why kodu behaved in the way they have acted out in the physical world. Our goal with this activity is to help students make their biases and misconceptions explicit so that we can inspect them and help them overcome them. In addition, we wanted them to have plenty of practice simulating Kodu rules and making predictions about kodu behavior. Moreover, kinesthetic activities were introduced to help students gain awareness of the dynamic nature of Kodu during the execution of rules.

7.2.2 Session Overview

In this study, we used a 4-session curriculum adapted from the original Kodu curriculum in which students completed activities which primarily focused on variations of the pursue and consume idiom using 2-3 rule programs. The activities were also selected because this is where we saw most of the students' misconceptions and fallacies on the post-assessments. Using the instructional approach described previously, we focused each session on successively introducing one of the first three Laws of Kodu, and testing and evaluating students' reasoning based on the focal law and its application with combination of previous laws. We then, used the last session as an opportunity to explore students' use of the laws and pursue and consume to build and debug their own games. Table 7-1 summarizes the learning objectives for each of the three sessions.

Table 7-1. Learning objectives for each Session/Week

Session / Week	Learning Objective
1	• Introduction to Kodu Game Lab • Understand WHEN-DO semantics of Kodu • Introduction to Pursue and Consume Idiom and constructing simple program using tiles • Scaffolding students' ability to use 1st law- "Each rule picks the closest matching object" to simulate and predict program behavior.
2	• Understanding the behavior of individual Pursue and Consume rules • Introduction to the 2nd law- "Any rule that can run, will run" and its use in interpreting which rules were running and the effects on program behavior • Ability to construct and mentally simulate Pursue and Consume programs using 1st law and 2nd law
3	• Introduction to the 3rd law- "When actions conflict, the earlier wins" and its use to identify conflicting rules and the resulting program behavior • Ability to reason and mentally simulate programs using all three laws
4	• Apply Pursue and Consume and a new design pattern from flashcards to build a game of own choice

By the end of the third session, we expect students to be able to understand the Kodu semantics, know how to use laws in reasoning about the execution of a program, recognize and be able to use the pursue and consume design pattern, mentally simulate 2-rule and 3-rule programs, and predict the behavior of kodu. This is what essentially would exhibit the presence of advanced computational reasoning ability after this 4-session curriculum.

7.2.3 Session Structure

Each session started with a general activity which helped students to discover the Kodu law of the day. After the activity, the law was taught and a fridge magnet

referring to the law was given to every student for them to refer back to throughout the session. Students were also shown a video for the 1st and 2nd laws of Kodu. The video shows students, animations of different combinations of the pursue and consume idiom (e.g., reversed rules, pursue only and consume only) which was aimed at helping students associate program behavior with the laws. The videos were paused at different intervals to ask students to predict the program behavior before resuming the video. Afterwards, students were given the opportunity to apply the law using different two and three line programs either using the Kodu programming environment or through kinesthetic activities.

Lastly, students were engaged in a kinesthetic activity where they were shown a program and asked to predict the behavior of kodu using the laws. Then students were asked to explain their predictions and reasoning. Subsequently the program was run on a projected screen where students saw the execution of the program, and the behavior was discussed and explained by the instructor. This was expected to help students build correct reasoning, understand how laws are applied, and eliminate any misconceptions which the students might have had.

7.3 Methodology

We conducted a four-week study to explore student reasoning about programs and assessment questions and the usefulness of explicitly teaching laws. We conducted four ninety-minute session after school with three small groups of participants. Participants first participated in the instructional intervention session. Then they completed a paper-based pre-& post assessments based on the content covered in the session. The assessments where focused on recognition of laws and idioms and students understanding and simulation of the rules, prediction of program behavior.

After the completing their assessment, students were asked to participate in a think-aloud interview.

In the think-aloud interviews, students were asked to explain their reasoning on the mental simulation questions from the assessment and on Session 2 and 3, they were asked to explain their interpretation and simulation of the program on a set of new single correct multiple choice questions. This was done to capture the reasoning of the students in real time.

In each of the think-aloud interviews, students were asked to read and explain the question and then reason about it while speaking simultaneously. The interviewer asked students to explain why they chose an answer option or rejected the other options. The think-aloud interviewers were also facilitators in the instructional intervention, so students were comfortable in sharing their ideas without fear of judgement.

7.4 Participants

7.4.1 Participant Recruitment

Eighteen participants were randomly recruited from an elementary school. Students were given flyers and interest forms to sign up for the study. An information session was conducted for parents where they were provided with all the relevant details about the study and Kodu. We received the necessary parental consent and student assent for each student to participate prior to the start of study. These eighteen students were divided into one of the three groups offering based on their schedule availability.

7.4.2 Demographics

By chance, every group ended up having six students with four boys and two girls each. Seven students were from fourth grade and eleven students were from fifth grade.

7.4.3 Prior Programming Experience

16 out of 18 students indicated that they had prior programming experience while the remaining two did not. All eighteen students indicated that they had done Hour of Code Activities, while varying numbers of students indicated using other programming environments such as Code-Monkey (n = 1), HTML and JavaScript (n = 5), Kodu (n = 1), Mine-Craft (n = 3), Python (n = 1), Scratch (n = 3), Vex Robotics or Lego (n = 3), Other (n = 3). On asking the number of prior computing workshops and activities experienced, seven students indicated that they had experienced one while another seven students indicated participation in two to three computing activities/workshops. One student indicated participation in four to five computing activities/workshops; and three students indicated participation in six or more computing activities/workshops. Most of the students (n = 13) indicated that they have explored programming as a part of class activity.

7.5 Data Collection

Data was collected on pre-and post-assessments and think-aloud interviews. Extra mental-simulation questions were added in the assessment to test mental simulation reasoning. Besides being a part of the assessment, these mental simulation questions were also used in the think-aloud interviews which were aimed at helping researchers collect data on students' reasoning patterns and styles.

Data collection and think-aloud interviews were taken on the first three sessions. In sessions two and three, students were given pre-assessments where they were asked the questions on the law which they were about to learn in the same class. This was done to collect data to understand how students reason before they are taught the required law. These pre-assessment responses help in understanding the default reasoning patterns.

7.6 Data Analysis

Based on the responses on both pre-and post-assessments, and think-aloud interviews, we analyzed student responses to questions primarily related to mental simulation and understanding of laws. This analysis is qualitative and descriptive in nature as we explored the nuances in reasoning ability of students.

7.7 Findings

Our first research goal was to verify and validate various misconceptions and fallacies which have been observed in the past, and understand the role which the laws play in either correcting or supporting the misconceptions in the process of reasoning. Our analysis, based on data collected from pre-and post-assessment responses and think-aloud interviews, results in four claims about students' understanding, reasoning and role of laws:

Claim 1: Students have preconceived notions of the sequential execution of rules (sequential procedure fallacy) and learning of laws is effective in removing this fallacy

Claim 2: Students can refer to, state and apply the laws correctly when reasoning about programs

Claim 3: Laws can be misapplied when students reason about 3-rule programs

Claim 4: Validation of Negative Transfer and the role of laws to correct them

7.7.1 Claim 1: Students have preconceived notions of the sequential execution of rules (sequential procedure fallacy) and learning of laws is effective in removing this fallacy

On Session-1 students were introduced to the pursue and consume (P&C) rules, and were taught the 1st Law of Kodu, “Each rule picks the closest matching object”. Each time the students saw the rules in the rule editor or on video, they were in order, rule 1 was Pursue and rule 2 was Consume. However, at that point, they had not been introduced to how Kodu executes rules and that any rule that can run, will run (2nd and 3rd Law of Kodu). Students were asked questions on Session-2 and Session-3 pre-assessment which evaluated their reasoning on an inverse pursue and consume rule prior to being taught 2nd and 3rd law to verify the existence of sequential fallacy. In the following discussion, a comparison of pre-and post-assessment responses from Session 2 and 3 is presented to establish that students did have the sequential procedure fallacy and that the use of explicit laws helped in removing the fallacy.

7.7.1.1 Session-2, Pre-Assessment, Q2.

Q2 (Figure 7-1 Q2-right) was asked to identify students’ reasoning about rule-ordering using inverted P&C rules, prior to explicit instruction on 2nd law. Q2 asked students what the kodu would do in the given world (Figure 7-1 Q2-right). An image of the Kodu world was provided which had the kodu and three coins and asked students what would the kodu do in the given world (Figure 7-1 Q2-right).

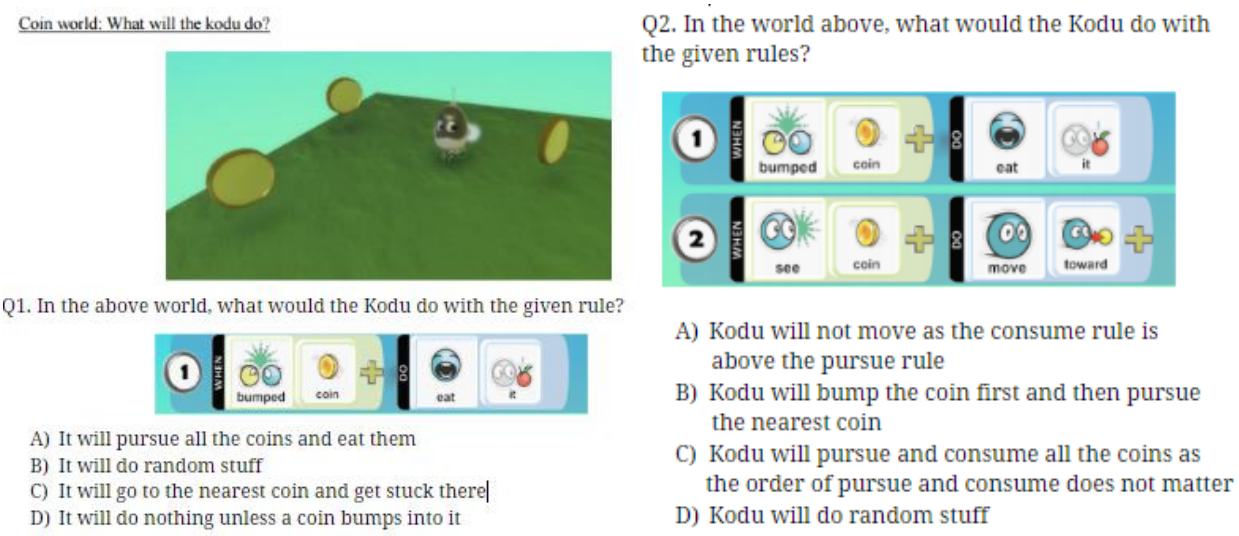


Figure 7-1. The pre-assessment used on Session-2: Q1 (left), Q2 (right)

The answer to Q2 is option C – "Kodu will pursue and consume all the coins as the order of pursue and consume does not matter". Q2 has two answer options that suggest that students may have sequential rule execution fallacy: Options A & B. The following were students' responses on Q2:

Q2. Option A- ("Kodu will not move as the consume rule is above the pursue rule"):11 out of 16 students incorrectly marked option A which suggests that they think that kodu will not move because of the reverse ordering of the rule. Students selection of this answer option suggests that they thought that these rules will not make kodu do anything because sequential execution of the reverse ordered rules do not make sense. Selection of this option also suggests that students were drawing on a preconceived notion that rules execute sequentially.

Q2. Option B- ("Kodu will bump the coin first and then pursue the nearest coin"): 3 out of 16 students incorrectly marked option B which meant that consume rule will be executed first and then the pursue rule. This is a special case of the sequential procedure fallacy, which suggests that students thought that by default the first rule will

run first. They failed to test the feasibility of kodu bumping an apple even before it is pursued.

Q2. Option C- ("Kodu will pursue all the coins as the order of pursue and consume does not matter"): 2 out of 16 students correctly marked this option which meant that the order of pursue and consume rules did not matter in this case.

Q2. Option D- ("Kodu will do random stuff"): None of the students marked the option D (Kodu will do random stuff).

Analysis of Q2 demonstrates that the majority of students, 14 out of 16 students (87.5%) held the sequential execution fallacy prior to explicit teaching of 2nd law.

7.7.1.2 Session-2, Post-Assessment, Q5.

At the end of Session-2, students were asked to reason about an inverse pursue and consume rule question Q5 (Figure 7-2) during the think-aloud which was similar to Q2 on the pre-assessment (Figure 7-1 Q2-right.) By this time in the curriculum, the students were introduced to 2nd Law of Kodu which states that "Any rule that can run, will run" which is essential for understanding how to reason about inverse pursue and consume rules. They also were exposed to different cases of inverse pursue and consume rules and had experience in running such programs using the Kodu rule editor. In addition, they also had experienced reasoning about such programs as part of the kinesthetic activities where they executed the rules like Kodu would and predicted program behavior.

Q5. 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.

Figure 7-2. Q5 on the Session-2 post-assessment question

For this Q5, the correct option was C which stated, "Eat all the coins; it doesn't matter that the consume rule comes first". Options A and B indicate the presence of sequential fallacy.

The following were students' responses on Q5:

Q5. Option A- ("Sit around waiting for a coin to bump into it"): None of the students marked the incorrect option A, which suggests that the rules will not produce any action. If a student marked this option, it would be attributed to sequential procedure fallacy because it implies that the pursue rule will not run as it is below pursue and the kodu will sit wherever it is and wait for the coin to bump so that the consume rule can run. Bumping of the coin will make the first rule true, hence the coin will be eaten by the kodu.

Q5. Option B- ("Eat one coin and then stop"):-None of the students marked the incorrect option B which also suggests the sequential fallacy coupled with incorrect simulation because a student can sequentially interpret the first rule (consume rule) to

"eat one coin" first and then interprets the second rule (pursue rule) to not run as it is below a consume rule, so stopping the kodu.

Q5. Option C- ("Eat all the coins; it doesn't matter that the consume rule comes first"): 15 out of 16 students answered correctly by marking this option as the order of pursue and consume rules does not matter. This shows that students who incorrectly reasoned sequentially (n=14) on Q2 on the pre-assessment were able to correctly reason about inverted pursue and consume rules after being introduced the 2nd law.

Q5. Option D- ("Go to the first coin and get stuck there"): 1 out of 16 students incorrectly answered option D. This same student had marked the option which said that the kodu will not move as the consume rule is above pursue rule which suggested sequential procedure fallacy on Q2 of the pre-assessment.

7.7.1.3 Reflection on Session-2

Table 7-2 compares the pre-and post-assessments of the students on a same question of inverse pursue-and-consume rules respectively from Session-2. It should be noted that while majority of the student (n=14/16) marked options which represent the sequential fallacy on pre-assessment, after the session, almost all (n=15/16) marked the correct option as they were introduced explicitly to 2nd law during the session.

Table 7-2. Pre- and post-assessment results on Session-2: correct responses shown in blue; sequential procedure fallacy in red.

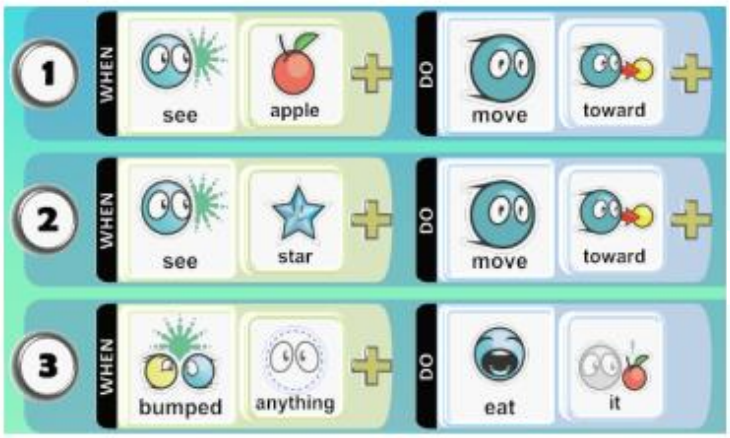
	A	B	C	D	Total
Pre	11	3	**2**	0	16
Post	0	0	**15**	1	16

This suggests that explicit teaching of the laws is helpful in mitigating students' initial sequential execution fallacies.

7.7.1.4 Session-3, Pre-Assessment, Q2.

In this section, we discuss a similar pre-and post-assessment comparison from Session-3. Like Session-2, students' responses were taken on pre-and post-assessment on Session-3. Students were asked to answer 4 questions on a pre-assessment that asked students to reason about 3-rule programs. The first two questions were based on the 2-pursue 1-consume program and the second two questions were based on 1-pursue 2-consume programs (Figure 7-2). This was the first time that the students were exposed to the 3-rule programs because earlier in Session 1 and 2, they only saw and ran 2-rule simple pursue and consume programs. In the following, we discuss the second question of the 2-pursue and 1-consume program which asked when will the kodu eat a fish (Figure 7-3 Q2-below). Students were provided an image of Kodu world which had kodu, 2 apples and 2 stars. Q2 demonstrates a 3-rule problem with rule conflict requiring students to apply the 3rd Law of Kodu. It asks, when will the kodu eat its first star?

Answer the next two questions using the same apple/star world, but with these different rules:



1. 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.

2. 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.

Figure 7-3. The first two questions of the Session-3 pre-assessment based on 2-pursue and 1-consume rule. Q2 (highlighted in the colored box)

Based on the 3rd law, option A ("When there are no apples left") is the correct answer as the first pursue rule (pursue apple) will have to be exhausted for the second pursue rule (pursue star) to run. Here, option B ("Right after it eats its first apple") refers to the sequential procedure fallacy as a student who is reasoning sequentially will think that since the pursue star rule is after pursue apple rule, kodu will eat its first star after it eats its first apple.

The following were students' responses on Q2:

Q2. Option A ("When there are no apples left"): None of the students marked the correct option A. This shows that no student could correctly reason about the execution of the rules without the understanding of the 3rd law.

Q2. Option B ("Right after it eats its first apple"): 9 out of 17 students incorrectly answered by marking this option. We believe that students marked this option because they could see a pursue star rule just below the pursue apple rule. This implies that students did not know how to reason about the conflicting rules, and instead chose to reason sequentially resulting in choosing this option. Thus, based on this understanding of option B, it can be said that students had sequential procedure fallacy in their reasoning.

Q2. Option C ("It will never eat a star; it will keep **looking for apples forever"):** 1 out of 17 students incorrectly marked option C which suggests that only the first rule runs in the entire program irrespective of whether apples are present or not. Here student's reasoning does not take into account the conditional execution of the rule, which means that if there are no apples, then the pursue apple rule is not true and hence, it will not run.

Q2. Option D ("it will only eat a star if it bumps into one by accident"): 1 out of 17 students incorrectly marked option D which suggests that kodu will not pursue the star. This indicate that student is unable to recognize the pursue and consume idiom with respect to stars.

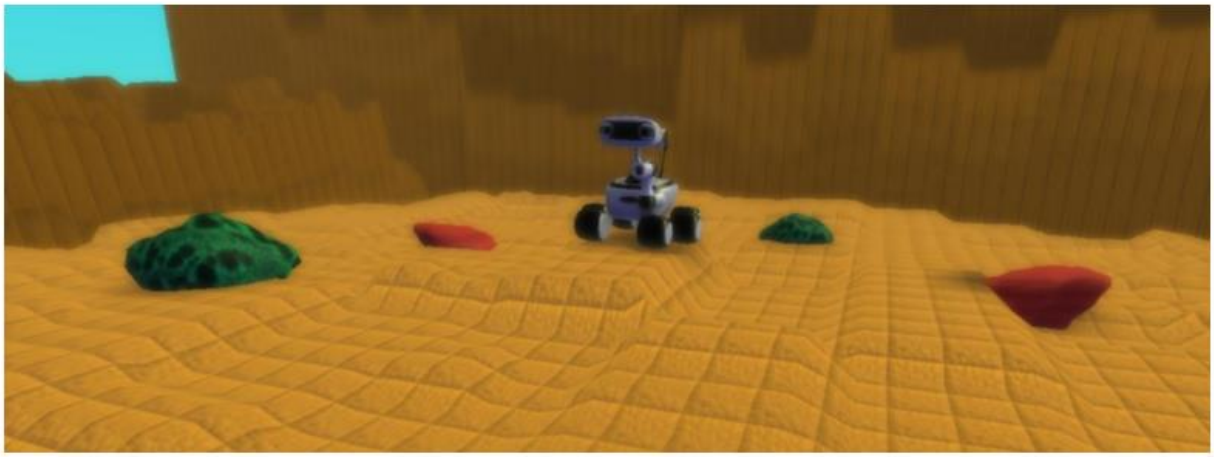
Q2. Blank Responses: 6 out of 17 students did not answer the question.

Students' responses to this question suggests that 53% (n=9) defaulted to using sequential reasoning to answer this question, 6% (n=1) executed the rules incorrectly, 6% (n=1) did not comprehend the pursue-and-consume-star relationship while reasoning this question, and 35%(n=6) didn't provide an answer to this question which suggests that they didn't know how to answer this question.

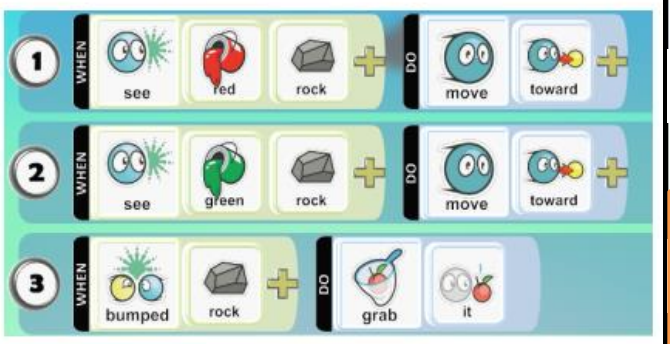
7.7.1.5 Session-3, Post-Assessment, Q11, Part-2

After Session-3, students were asked question Q11, part-2 which was similar to Q2 on the Session-3 pre-assessment (Figure 7-3: Q2), however Q11, part-2 was set in a different world with different objects (Figure 7-4: Part-2). At this point in the curriculum, students learned 3rd Law of Kodu which stated that "When actions conflict, the earliest win". During Session-3, students worked on creating a program where kodu eats all the coins first and then the hearts. They also simulated many Kodu programs during kinesthetic activities for this session which required students to practice reasoning with the 3rd law to simulate Kodu programs and predict program behavior.

Grabbing Rocks on Mars



11. Answer these questions using the Mars world shown above.



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

1. **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?

2. **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 7-4. Session-3 post-assessment 3-rule question. Part-2 (highlighted in the colored box)

The question (Figure-7-4: Part 2) had a pursue red rock rule, then pursue green rock rule followed by a consume rock rule. Students were asked to tell when will the rover grab its first green rock (Figure-7-4: Part 2). The correct answer was option A- "When the red rocks are gone" based on the application of the 3rd law. The sequential fallacy is represented by option B- "Right after it grabs a red rock".

The following were students' responses on Q11, Part-2:

Q11, Part-2. Option A ("When the red rocks are gone"): 10 out of 17 students correctly marked this option A means that these students directly applied the 3rd law to answer this question. 5 out of these 10 students, in pre-assessment had earlier marked the option which represented the sequential procedure fallacy. It can be said that the 3rd law did affect their reasoning on 3-rule questions, and it was successful in removing the sequential fallacy.

Q11, Part-2. Option B ("Right after it grabs a red rock"): 4 out of 17 students incorrectly marked this option B which indicates the sequential interpretation of the rules by the students. 3 out of these 4 students did mark the same option (Figure 7-3 Q2 below option B, "Right after it eats its first apple") when asked in the pre-assessment questions. The remaining 1 student out of these 4 students marked the option C (Figure 7-3: Q2, option C, "It will never eat a star; it will keep looking for apples forever") on his pre-assessment. This suggests that there are still students who hold on the sequential procedure fallacy or are unable to correctly reason using 3rd law.

Q11, Part-2. Option C ("It will never garb a green rock; it will keep looking for red rocks"): 1 out of 17 students incorrectly marked the option C. This same student had also answered the pre-assessment question (Figure 7-3 Q2-below) incorrectly by answering option D which states that "it will only eat a star if it bumps into one by accident". This suggests that student misapplied the 3rd law as he/she thought that only the earlier or lower numbered rule will run. The 3rd law states that in conflict, the earlier rule wins, however if it's WHEN condition becomes false then it will cease to run. This student did not understand how to apply the 3rd law while reasoning

Q11, Part-2. Option D (It will only grab a green rock if it bumps into one by accident): 1 out of 17 students incorrectly marked this option and he/she had a blank response in the pre-assessment.

Q11, Part-2. Blank Response

Overall, there was 1 blank response out of the 17 students.

7.7.1.6 Reflection on Session-3

Table 7-3 compares the pre-and post-assessments of the students on a same question with 2-pursue rule and 1-consume rule respectively from Session-3. It should be noted that while majority of the student (n=9/17) marked the option which represent the sequential fallacy on pre-assessment, after the session, majority of students (n=10/16) marked the correct option as they were introduced explicitly to 3rd law during the session.

Table 7-3. Pre- and post-assessment results on Session-3: correct responses shown in blue; sequential procedure fallacy in red. * Includes blank responses.

	A	B	C	D	Total
Pre	0	9	1	1	17*
Post	10	4	1	1	17*

This suggests that explicit teaching of the laws is again helpful in mitigating students' initial sequential execution fallacies.

7.7.1.7 Discussion on Claim 1:

It can be said that students in both, Session 2 and 3 pre-assessment questions were sequentially reasoning through the rules. It can be said that by default, students assume that rules are executed in the order they are written. We observe that even after removing the sequential fallacy in Session-2, it reappears in Session-3 which had longer programs and were more complex than the program students encountered during

Session-2. This indicates that when students were unaware about the 3rd law, which is essential to reason about conflicting rules, they engaged in sequential interpretation of programs. This suggests that sequential rule execution is central to students' mental model of programs execution prior to explicit instruction. This pattern of reasoning can be attributed to students' general reading and writing ability which has been cultivated by students over their early elementary school learning.

In reflecting on the effectiveness of explicit teaching laws, it can be said that in both Session 2 and 3 post-assessments, students have been able alter their reasoning after being introduced to laws that explain how to reason about rule execution when they appear out of order (2nd law) and/or conflict with other rules (3rd law) which is required for reasoning about more complex programs.

7.7.2 Claim 2: Students can refer to, state and apply the laws correctly when reasoning about programs.

In the last Claim 1 (section 7.7.1), we discussed how teaching of explicit laws helped in removing the sequential procedure fallacy. In this section, we will discuss the examples from the think-aloud interviews of Session-2 where we observed students directly referred and stated laws while explaining their reasons.

7.7.2.1 Session-2, Think-Aloud, Q3.

During the Session-2 think-aloud interviews, students were first given inverse pursue and consume rules to test the understanding of the rules (Figure-7-5 Q3. left), and then they were asked to predict program behavior based on it in context of the given world (Figure 7-5 Q5. right).

In Q3 (Figure-7-5 Q3. left), inverse pursue and consume rules were shown and students were asked what will these rules do? The correct option was A which stated

that “they can pursue and consume all the hearts” as the order of pursue and consume will not matter. Option B which stated, “they cannot do anything as the pursue rule is below the consume rule” refers to the sequential procedure fallacy.

The following were students’ responses on think-aloud Q3:

Q3. Option A (“They can pursue and consume all the hearts”): 14 out of the 16 students, marked the correct option A.11 out of these 14 students who correctly answered the question, directly stated or referred to the 2nd law of Kodu while explaining the reason for marking this choice. The remaining 3 did not refer the 2nd law, but explained their reasoning by applying that law in a correct way.

Q3. Option B (“They cannot do anything as the pursue rule is below the consume rule”): 2 out of 16 students incorrectly marked the option B. During the think-aloud interview, both the students confirmed the sequential procedure fallacy.

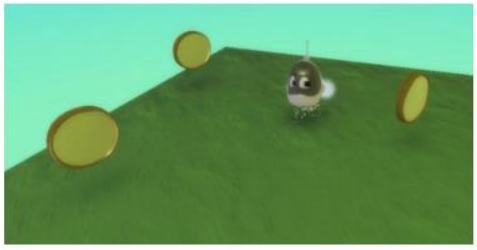
Q3. Option C (“These rules make no sense”): None of the students marked option C, and during think-aloud interviews rejected this option with correct reasoning by saying that these rules do make sense to them.

Q3. Option D (“They can do random stuff”): None of the students marked option D, and during think-aloud interviews rejected this option with correct reasoning by saying that kodu will not do random stuff.

Q3. What can these rules do?

A) They can pursue and consume all the hearts
B) They cannot do anything as the pursue rule is below the consume rule
C) These rules make no sense
D) They can do random stuff

Coin world: What will the kodu do?



Q5. In the world above, what would the Kodu do with the given rules?

A) Kodu will not move as the consume rule is above the pursue rule
B) Kodu will bump the coin first and then pursue the nearest coin
C) Kodu will pursue and consume all the coins as the order of pursue and consume does not matter
D) Kodu will do random stuff

Figure 7-5. Q3 Inverse pursue and consume think-aloud question (left); Q5 Inverse pursue and consume to be applied in the Coin World (right)

Think-aloud Interviews: The three interview excerpts below provide examples from three different students' think-aloud interviews of their explanations for choosing option A and show students directly referencing and stating the 2nd Law of Kodu while explaining their selection of option A.

Interview Transcript 1: Student directly referred to the 2nd Law of Kodu

Interviewer: Why did you mark option A?

Student A: "because even though this is below (pursue rule) the top one (consume rule) it can still follow that rule (pursue rule) because that's the second rule of kodu"

This transcript shows that Student A is able to point to the pursue and consume rules individually, comment on their current order in the program, and attribute the

working program to the 2nd Law of Kodu. This suggests that Student A knows that even though the order of the rules is switched in the program, it will still work as intended.

Interview Transcript 2: Student indirectly referenced the 2nd Law of Kodu

Interviewer: Why doesn't the rule order matter?

Student B: “Yeah, because if the order did matter and it was like that [indicating the rules in the question], it would not work because this one is in front of it (consume rule), but since the second rule."

Interviewer: [interrupt student speech for clarity] second rule or second law?

Student B: “the second law is that it does not matter what order they are in, they would still do it [function normally.]”

Transcript 3 shows that Student B is able to explain the implications of the 2nd Law of Kodu on the program while explaining their selection of option A which suggests that the student understands that rule order doesn't matter and that they are attributing this knowledge to the 2nd Law of Kodu.

Interview Transcript 3: Student directly referred the 2nd Law of Kodu

Context: Student read the rules aloud

Student C: “…and I know from the 2nd Law of Kodu that it does not matter what order they are in, so they can pursue and consume all the hearts.

Overall, these think-aloud responses indicate that students understood how to apply the 2nd Law of Kodu and students’ reasoning were positively influenced by the explicit teaching of the laws.

2 out of 16 students incorrectly marked the option B. The interview transcript #5 highlights the reasoning of one of these students and confirms that option B indicates evidence of the sequential procedure fallacy.

Interview Transcript 4: Student indicated sequential procedure fallacy

Interviewer: Why did you choose B?

Student D: “B, as it cannot get the hearts” [this means that kodu cannot pursue the hearts]

Interviewer: You cannot get the heart because the bump rule is first?

Student D: "Yes"

Confirmation of Sequential Procedure Fallacy: One of the students, Student E, confirmed that earlier he/she used to think that the rules execute sequentially. Transcript #5 is an excerpt from Student E's interview when asked Q3 (Figure 7-5).

Interview Transcript 5: Student confirms that before learning the 2nd Law of Kodu, Student E used to have the sequential procedure fallacy

Context: Student is deciding the answer after reading the question and the rules

Student E: "I would say the first one (option A)"

Interviewer: Why?

Student E: “Because even though this (pursue rule) is below the top one (consume rule), it (kodu) can still follow that rule (pursue rule), because that’s second rule of Kodu. And the rest of these (options), [reads option B] 'they cannot do anything as the pursue rule is below the consume rule’, I used to think that but that’s not the 2nd Law of Kodu"

This interview illustrates that Student E used to think that rules executed sequentially, which confirms the presence of sequential procedure fallacy.

None of the students marked options C and D during think-aloud interviews. All of the students rejected these options and correctly reasoned that "these rules do make sense" because kodu will not do random stuff.

Students' responses to this question suggest that 87% (n=14) correctly answered the question during think-aloud and 69% (n=11) stated and referred to the second law. 12% (n=2) confirmed the sequential procedure fallacy during think-aloud. This suggests that students used 2nd law to reason about kodu's behavior.

7.7.2.2 Session-2, Think-Aloud, Q5.

In a subsequent question Q5. (Figure 7-5 Q5-right) during the think-aloud, where the inverse pursue and consume rules were given in context of a Kodu world, students were asked to predict the kodu's behavior. Here, option C which stated that "Kodu will pursue and consume all the coins as the order of pursue and consume does not matter" was the correct option as the order of pursue and consume rule did not matter and option A which stated that "Kodu will not move as the consume rule is above the pursue rule" referred to the sequential procedure fallacy.

The following were students' responses on think-aloud Q5:

Q5. Option A ("Kodu will not move as the consume rule is above the pursue rule"): 1 out of 16 students, who incorrectly marked option A confirmed the sequential execution of rules (sequential fallacy) during think-aloud interview.

Q5. Option B ("Kodu will bump the coin first and then pursue the nearest coin"): None of the students marked option B, and during think-aloud interviews

rejected this option with correct reasoning by saying that kodu cannot bump the coin first.

Q5. Option C ("Kodu will pursue and consume all the coins as the order of pursue and consume does not matter"): 15 out of the 16 students, answered correctly by marking the option C. 9 out of these 15 students who answered correctly, directly referred or stated 2nd law while explaining the reason of choosing this option in the think-aloud. These 9 students also referred or stated 2nd law in the previously discussed question Q3 (Figure-7-5 Q3. left), where only the inverse pursue and consume rules were given. Other students said that rule ordering will not matter which implied a correct application of 2nd law.

Q5. Option D ("Kodu will do random stuff"): None of the students marked option D, and during think-aloud interviews rejected this option with correct reasoning by saying that kodu will not do random stuff as it will follow the rules.

Think-aloud Interviews: The interview excerpts below provide examples of four different students' direct reference to or stating 2nd law while explaining correct option C.

Interview Transcript 6: Student indirectly referred to the 2nd Law of Kodu

Interviewer: Why do you think it's C?

Student F: "I think it is C because it does not matter what order the code's in, it's just gonna (going to) run what can run"

Interview Transcript 7: Student directly referred to the 2nd Law of Kodu

Interviewer: Why is option A not correct?

Student G: "because that is basically saying almost the opposite of the 2nd Law of Kodu…."

Interviewer Transcript 8: Student directly referred to the 2nd Law of Kodu

Interviewer: Why did you choose option C?

Student H: “because of the second law of kodu”

Interviewer Transcript 9: Student directly referred to the 2nd Law of Kodu and explained it

Context: At the beginning of one of the interview

Interviewer: What did you learn today?

Student I: “I learned today that it doesn’t matter which rule is first, it matter which rule works, so if the first rule is WHEN bump apple eat it and the second rule is WHEN see apple move toward, the second one will work instead of the first one”

Interviewer on Q5: What is the correct answer?

Student I: “I think it is C…because it is the law of kodu number 2, the 2nd Law of Kodu is whatever can run will run so it would go when it would see coin and when it sees coin it will move toward it and then the second law will go in and then when it bumps the coin it will eat it”

This shows how students did understand the laws and that they were able to apply them in correct context. 1 out of 16 students, who incorrectly marked option A confirmed the sequential execution of rules (sequential procedure fallacy) during think-aloud interview. The excerpts of this students’ interview confirming the sequential procedure fallacy is below.

Interviewer Transcript 10: Student confirmed that rule order does matter implying sequential procedure fallacy

Context: The student indicated that after the second rule, the first rule cannot be executed

Interviewer: Do the order of the rules matter?

Student J: “Yes, it does matter…”

None of the students marked options B and D during the think-aloud interviews and rejected these options with correct reasoning by saying that kodu cannot bump the coin first and kodu will not do random stuff as it will follow the rules respectively.

7.7.2.3 Discussion on Claim 2:

Through this data, we can say that student’s reasoning ability is affected by the laws, and they can refer to the laws while reasoning. It can also be noted that students who consistently reason correctly also consistently refer and use the laws correctly.

7.7.3 Claim 3: Laws can be misapplied when students reason about 3-rule programs

While analyzing data from pre-and post-assessment responses and think aloud interviews, we found that students were using laws incorrectly in some questions and in some cases, they justified their incorrect responses by stating laws during think-aloud. This suggests that students were having challenges in identifying which law would determine the interpretation of rules in a particular context. Students were also having challenges in reasoning about programs with 3-rules because it required them to reason with multiple rules. A major fallacy we observed in students’ reasoning was collective choice fallacy, where students, while reasoning about the conflict between 2-pursue rules applied 1st law and reasoned that the two pursue rules jointly choose the closest object. We discuss some questions to explain this claim in detail.

7.7.3.1 Session-3, Pre-Assessment, Q1.

In Session-3 pre-assessment, Q1 asked the students to identify what will the kodu eat first, given a 3-rule program with 2-pursue and 1-consume rule (Figure 7-6). Students were not aware of the 3rd law which states "When actions conflict, the earliest wins". Knowledge of law-3 was required to correctly answer this question.

1. 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.

Figure 7-6. Session-3, Pre-Assessment Q1

Based on the 3rd law, the correct option is B "Apples" as it is the earlier or lower numbered rule. Options A refers to the sequential procedure fallacy and option C refers to the collective choice fallacy.

The following were students' responses on Q1:

Q1. Option A ("Stars"): 2 out of 17 students incorrectly answered this option which suggests that they were reasoning sequentially indicating the sequential procedure fallacy.

Q1. Option B ("Apples"): 5 out of 17 students correctly answered this question by marking this option.

Q1. Option C ("Whichever thing is the closest, no matter what"): 9 out of 17 students incorrectly marked this option which stated that kodu will eat the closest object, no matter what. At the time when the students responded to this question, they were

familiar with 1st and 2nd law. This answer option resonated with the 1st law as it states that “each rule pick the closest matching object”. A majority of students picked this rule because this option resembled 1st law in which they have used in the past for choosing where will the kodu go.

Q1. Option D (“It will choose randomly”): 1 out of the 17 students incorrectly marked this option. It indicates that the student thinks that kodu can choose any object as both pursue rules are true. However, it may also indicate that student does not reason at all and thus, does not apply 1st law which was discussed in the last session while reasoning.

7.7.3.2 Reflection on Session-3, Pre-Assessment, Q1.

In Q1 of Session-3 pre-assessment (Figure 7-6), we observe how students used 1st law as they had prior knowledge about it. This shows the effect of reinforcing the law and how students may misapply it. This was an example of a pre-assessment question so students did not have the knowledge of the correct law to be applied. In the following, we discuss some post-assessment results and think aloud responses from mental simulation questions which give us insights on the use of laws during mental simulation of programs.

7.7.3.3 Session-3, Post-Assessment & Think-Aloud, Q13.

After Session-3, students were asked to trace and simulate the path of kodu in two questions (Figure 7-7, Q13 & Q14). Later during the think-aloud interviews, they were also asked to explain their responses.

Q13. Here is another Kodu program for eating apples. Draw the path and write a number next to each apple to show the order in which the apples will be eaten.

Q14. Here is another Kodu program for eating apples. Draw the path and write a number next to each apple to show the order in which the apples will be eaten.

Figure 7-7. Session-3, Post-Assessment Mental Simulation Question: Q13(left), Q14(right)

Q13 (Figure 7-7: Q13-left) asked the students to simulate a 2-pursue 1-consume program on a paper based map, positioning kodu, 2 red apples and 2 blue apples. In this case, kodu will first pursue all the red apples (3rd law) starting from the closest one (1st law), and then closest blue apples (1st and 2nd law). The correct order in which the given apples will be eaten, from left to right is 1,4,3,2.

The following were students' responses on Q13:

Q13. Answered correctly as 1 ,4, 3, 2: 7 out of 16 students answered correctly by labelling the order as 1, 4, 3, 2 as all the red apples are eaten first and then the blue apples.

Q13. Answered incorrectly as 1 ,3, 4, 2: 1 out of 16 students answered incorrectly by marking the order as 1, 3, 4, 2. During think-aloud, the student explained

the he/she applied 3rd law correctly and decided that the red apples will be eaten first and then the blue apples. However, the student's point of reference was always from the original position of kodu which indicates a **static kodu misconception**. The student was not able to see that when the kodu eats its second red apple, its closest blue apple will be different from the closest blue apple from its initial position.

Q13. Answered incorrectly as 1, 2, 3, 4: 6 out of 16 students answered incorrectly by marking the order as 1, 2, 3, 4. 3 out of these 6 students, changed their answers during think-aloud interviews as they recognized that kodu will pursue only red apples first and then the blue apples (3rd law application). This sequence refers to the collective choice fallacy because students think that both the pursue rules will jointly choose to pursue the closest object. Students prioritize the use of 1st law in their reasoning to choose the path of kodu. 2 students explained that the kodu will go to the nearest object which suggests a collective choice fallacy. The remaining 1 student referred and stated the 2nd law, and then made decision according to the closest object which also suggests the collective choice fallacy.

Q13. Answered incorrectly as 1,2,0,0: 1 out of 16 students incorrectly answered 1,2,0,0 and during think-aloud interview referred to the 1st law which suggests the collective choice fallacy. This student only executed the rules once, and he/she was unable to recognize that the rules run continuously suggesting the one-time rule execution misconception, else his/her answer would have been 1, 2, 4, 3. The student interpreted the rules sequentially and thought that kodu will alter the decision between eating red apples and blue apples.

Q13. Answered incorrectly as 1,0,0,0: 1 out of 16 students incorrectly answered 1, 0, 0, 0 but changed her answer during the think-aloud to the correct pattern 1, 4, 3, 4 as he/she applied the 3rd law and recognized that first rule will have the priority with the second rule.

7.7.3.4 Reflection on Session-3, Post-Assessment & Think-Aloud, Q13

We observe that 4 students exhibited collective choice fallacy and used 1st law while reasoning. 1 out of these four students referred to 2nd law "any rule that can run, will run" and justified that kodu can go to any apple but since it is closer to red first, it will go to the red apple. This indicates a type of reasoning pattern where students are unable to see the applicability of 3rd law and how 3rd law applies on top of 2nd law in decision making process. It should be also noted that during the think-aloud interview, 4 students changed their answers and recognized the correct reasoning pattern using 3rd law. This suggests that they did not pay adequate attention while reasoning before think-aloud interview was conducted. This may also indicate that verbally reading the question and reflecting on their answers during the think aloud changed their perspective on what the question was asking and/or helped them to recognize inconsistencies in their reasoning. Thus, the think-aloud helped us in distinguishing students who reasoned according to collective choice fallacy and students who were able to reason correctly but answered incorrectly due to other reasons like inadequate attention to the question instructions or unawareness of inconsistencies in their reasoning.

We find similar results in Q14 (Figure 7-7 right), where students exhibit identical patterns of reasoning.

7.7.3.5 Discussion on Claim 3:

In sections, i: e, 7.7.3.1 (pre-assessment-Q1) and 7.7.3.3 (post-assessment-Q13), we discuss two questions which require students to reason with the 3rd law. We observed that students have difficulty in recognizing the application of laws in 3-rule problems with two conflicting pursue rules as a result of reasoning based on collective choice fallacy. The collective choice fallacy refers to students' reasoning that two pursue rules jointly choose the closest matching object. It is a misapplication of 1st law, which states that "Each rule picks the closest matching object". Students' use of 1st law despite learning 3rd law to resolve the conflicting pursue rules demonstrates how laws can be misapplied.

Students have difficulty in correctly using the laws in the right context, and when they are given a multiple-rule program they tend to use the law they have experienced the most. This suggests that multiple-rule program reasoning is challenging and that use of collective-choice fallacy or one-time rule execution misconception are indicators that students may need additional practice and better instructions to overcome these reasoning challenges.

7.7.4 Claim 4: Validation of Negative Transfer and the role of laws to correct it

In the earlier studies, we found that students marked incorrect options which could be traced back to what they saw in the class or remembered in the context of that question [89]. We also observed the same behavior on some of the questions in our study.

7.7.4.1 Session-2, Pre-Assessment, Q1.

In Session-1, students saw demonstrations of pursue and consume programs but did not develop a nuanced understanding of how the individual rules behaved. At the

beginning of the next session (Session-2), they were provided with a pre-assessment and were given a similar world as they had experienced on Session-1 and asked that in the given world what would the kodu do with the given rule? (Figure-7-8: Q1). This was a relatively easy rule to interpret as students were introduced to pursue and consume rules in Session-1. However, they did not have experience in executing or using a single rule like this one.

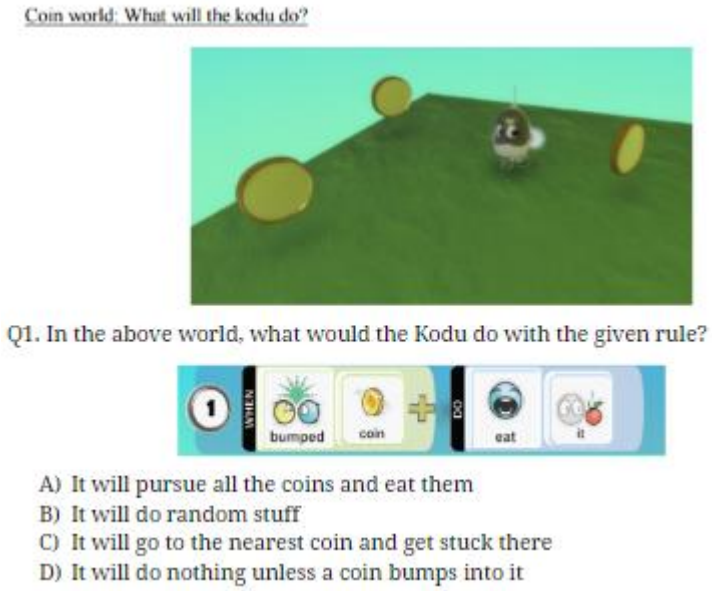


Figure 7-8. Session-2, Pre-Assessment Q1

In Q1(Figure-7-8 Q1), as there is no pursue rule, the kodu will not move and thus, option D which states that “It will do nothing unless a coin bumps into it” is the correct answer. Option A which states that “It will pursue all the coins and eat them” represent “negative transfer misconception” which means that students reason analogically based on what they experienced in the last session.

The following were student’s responses on Q1:

Q1. Option A (“It will pursue all the coins and eat them”): 7 out of 16 students, incorrectly marked option A, which means that they thought that kodu will

pursue and consume all the coins even if the pursue rule is not present. This is an evidence of an incorrect transfer of what students learned in the earlier session (pursue and consume). Students mentally referred back to what they saw during session one and thus answered that kodu will eat all coins. This reference which is analogical in nature and ignores the context, is an example of negative transfer.

Q1. Option B ("It will do random stuff"): None of the students marked this option. A student will pick this option if he/she is not reasoning or do not know if this is a consume rule.

Q1. Option C ("It will go to the nearest coin and get stuck there"): 2 out of 16 students incorrectly marked this option. This potentially shows a misapplication of 1st law and a negative transfer also as students were taught 1st law in the prior session, where they were explained that kodu goes to the closest matching object.

Q1. Option D ("It will do nothing unless a coin bumps into it"): 7 out of 16 students correctly marked this option and exhibited correct understanding of reasoning on an individual consume rule.

In this question, 7 students who marked option A clearly indicate the phenomenon of negative transfer. We can also infer that these 7 students have incorrect understanding of pursue and consume.

7.7.4.2 Session-2, Post-Assessment, Q4.

After Session-2, the students were given Q4 (Figure 7-9) which was similar to Q1 on Session-2 pre-assessment and featured single consume rule (Figure-7-8, Q1). Q4 (Figure 7-9) asked that what would the kodu do with the given consume rule?

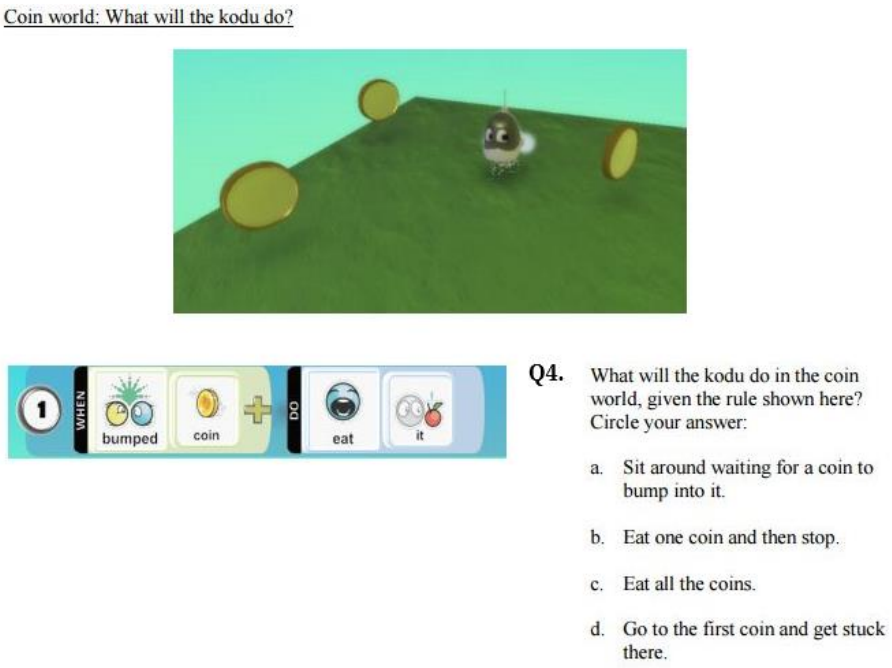


Figure 7-9. Session-2, Post-Assessment Q4

Based on the 2nd law, the correct answer of Q4 (Figure 7-9) is option A ("Sit around waiting for a coin to bump into it") as the kodu will not move. Option B ("Eat one coin and then stop") represents the sequential procedure fallacy and Option C ("Eat all the coins") represent the negative transfer misconception option.

The following were student's responses on Q4:

Q4. Option A ("Sit around waiting for a coin to bump into it"): 14 out of 16 students answered correctly by marking the option 'A' ("Sit around waiting for a coin to bump into it"). This indicates that the understanding of the 2nd law helped students in recognizing if the given rule can run or not. Earlier in the pre-assessment question, students were reasoning based on the conflated (joint) understanding of pursue and consume as experienced in the first session, but now they were reasoning and checking to determine if this rule can run or not. Such understanding helped in shaping their reasoning ability resulting in answering the correct option.

Q4. Option B ("Eat one coin and then stop"): 2 out of 16 students answered option 'B' ("Eat one coin and then stop") which suggests that students directly executed the rule. This indicates the sequential procedure fallacy.

Q4. Option C ("Eat all the coins"): None of the students marked this option, which suggests that they were all reasoning while answering the question. This option represents the negative transfer misconception option.

Q4. Option D ("Go to the first coin and get stuck there"): None of the students marked this option, which suggests that every student recognized that the given rule was a consume rule and not a pursue rule.

7.7.4.3 Think-Aloud Observation

In an in-class Apple world activity, students were exposed to the blue apples which were poisonous. When the blue apples were again given as a part of the post-assessment questions, 4 out 16 students marked their answers based on the blue apples being poisonous and not directly reasoning through the rules. This incorrect transfer of the understanding of characteristics of the objects resulted in incorrect reasoning by these 4 students. They decided their decision based on the apples but not on the rules in the think-aloud interviews. 1 student out of the 4 recognized the fault while reasoning during think-aloud. This confirms the phenomenon of negative transfer and its effect on students' reasoning ability.

7.7.4.4 Discussion on Claim 4:

We observed that students often use the understanding they gained from previous experiences and previously encountered questions when answering new questions. This analogical application of what a student has previously observed, impacts a student's current ability to correctly read and reason through the rules or the

laws to determine the correct behavior. Table 7-4 compares the pre-and post-assessment of the question that has a single consume rule (Figure 7-8-pre-assessment and Figure 7-9-post-assessment). The responses on the pre-assessment shows how reasoning based on negative transfer plays a negative role in students' ability to correctly answer this question.

Table 7-4. Pre- and post-assessment results on Session-2: correct responses shown in blue; negative transfer shown in green.

	A	B	C	D	Total
Pre	**7**	0	2	**7**	16
Post	**14**	2	**0**	0	16

The responses on the post-assessment also indicate how explicit teaching of laws help in removing negative transfer successfully.

7.8 Takeaways from the Study

The findings described in this section suggest that:

- A student's understanding of simultaneous rule execution of Kodu programs determines their ability to correctly mentally simulate programs and predict program behavior.
- Mental simulation can be fostered by kinesthetic activities in the classroom.
- Students often have preconceived notions that negatively affects their reasoning about the program behavior such as negative transfer misconception, static kodu misconception and one-time rule execution misconception.
- Students often develop incorrect reasoning patterns about how the rules are interpreted in Kodu such as sequential procedure fallacy and collective choice fallacy which affect their ability to reason about programs.
- Learning laws is helpful in eliminating basic fallacies such as sequential procedure fallacy.
- Even if the students understand the laws and idioms, they may still incorrectly predict program's behavior because of misconceptions.

- On 2-rule programs, laws successfully scaffold students' reasoning about how rules were interpreted in Kodu and help students correctly simulate and predict program behavior. Thus, laws are helpful in supporting students' development of simple reasoning after brief learning experiences.

- On 3-rule programs, students tended to misapply laws because these programs required students to use one or more laws to simulate and predict the program behavior. This suggests that 3-rule programs are more complex to reason than 2-rule programs.

7.9 Limitations

Conclusions drawn for this study are based on students' limited exposure to Kodu content through three 90-min sessions. If students could have had more time to engage in and complete more activities on their own in Kodu, with more time for personal reflection, they might have resolved some of these issues. We believe this to be true because participants often changed their answers during the think-aloud when they had time to hear and to reflect on the inconsistencies in their thinking.

7.10 Future Work

The findings from this study can be used to understand the characteristics of different reasoners as mentioned in Neo-Piagetian classifications. Such a work would help in understanding how advance reasoners reason about programs, and perhaps what mental model should be fostered by instructions in a curriculum like Kodu.

7.11 Conclusion

This study verifies the presence of misconceptions and fallacies and discovers the negative effect caused by them in the students' reasoning process. We also find the successful use of laws in fostering computational reasoning and a subsequent decrease in misconceptions and fallacies.

During class activities and think aloud sessions, students were observed referring to and stating specific laws in their reasoning about and prediction of program

behavior. We also observed that laws were effective at helping students reason about simple two rule programs where only one law was applicable. However, laws were less effective in supporting students' reasoning about more complex programs (3-4 rules) because students had not developed the ability to effectively and correctly apply multiple laws when reasoning about program behavior. This is expected, given the limited amount of time students were given to practice and master concepts [97.

CHAPTER 8
HOW DO STUDENTS BECOME COMPUTATIONAL REASONERS?

The three studies discussed in this thesis provide evidence of the reasoning patterns elementary students engage in when reasoning about programs. The understanding of an elementary student is different from a high-school or an undergraduate student. Thus, to effectively tell if an elementary student is a computational reasoner, we must first understand their default reasoning pattern.

The findings from Chapter 5 suggest that question instructions on assessments, which may seem to be well structured, may be interpreted by students in different ways than those intended by the designer, confuse the students, and mask their true reasoning ability.

The findings from Chapter 6 suggest that when introducing students to programming, physical manipulative such as tiles and flashcards can help students to understand the syntax of a language and learn how to properly construct rules/statements. However, in order to scaffold the development of students' reasoning about program behavior, students need to tinker with constructing rules and programs in a programming environment to understand the dependency between rules and their behavior when they are run. Thus, interactions with visual programming environments provide students with dynamic feedback which help to develop their mental models of simulated programs and program behavior. Students' interactions with visual programming environments and activities that expose them to different initial program conditions and allow them to observe the resulting program behavior help students to develop more a contextualized understanding of rule behavior and execution. As a

result, these students are better able to reason about and predict program behavior on paper-based assessments.

The findings of Chapter 5 and 6 reaffirm the claim of Tew [85] who said that computing lacks valid and reliable assessments. Had we used the initial assessment instructions and the same classroom activities with the physical manipulatives, we would have drawn incorrect conclusions on students understanding.

Findings from Chapter 7 verified that students' naive misconceptions about how programs execute often persist even after explicit instruction and hinder their ability to reason about and predict program behavior. Findings from Chapter 8 also verify that during computing instructions students develop different fallacies which again negatively affect their reasoning ability. These findings are aligned with the findings from McCraken's 2001 ITiCSE working group [55], which found that novices have fragile knowledge, which negatively affects their reasoning ability.

The existence of sequential procedure fallacy and phenomena of negative transfer confirm that novices reason analogically, which has been earlier discussed by Du Boulay [22], Pea [63] and Halasz et al. [29]. Use of explicit laws and its success in removing sequential fallacy verifies the suggestions of Meerbaum-Salant et al. [56] where they suggested that difficulties in teaching CS concepts in Scratch can be overcome by explicitly teaching the language constructs. Thus, we found some of the earlier findings, which resonated with our observation and conclusions.

Implications: Curriculum designers and instructors need to understand misconceptions that students may have about program behavior based on prior programming experiences or logical and intuitive understandings of programs and their

behavior. In addition, curriculum designers and instructors need to consider how to help students understand new concepts given these naive misconceptions. It should be a goal for any curriculum designer to minimize the negative effects caused by the preconceived notions or understanding.

LIST OF REFERENCES

1. Adelson, B., & Soloway, E. (1985). The role of domain expenence in software design. *IEEE Transactions on Software Engineering*, (11), 1351-1360.

2. Aggarwal, A. (2017, March). Neo-Piagetian Classification of Reasoning Ability and Mental Simulation in Microsoft's Kodu Game Lab. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 745-746). ACM.

3. Aggarwal, A., Gardner-McCune, C., & Touretzky, D. S. (2016, February). Designing and Refining of Questions to Assess Students' Ability to Mentally Simulate Programs and Predict Program Behavior. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (pp. 696-696). ACM.

4. Aggarwal, A., Gardner-McCune, C., & Touretzky, D. S. (2017). Evaluating the Effect of Using Physical Manipulatives to Foster Computational Thinking in Elementary School.

5. Almstrum, V. L. (1999). The propositional logic test as a diagnostic tool for misconceptions about logical operations. *Journal of Computers in Mathematics and Science Teaching*, *18*, 205-224.

6. Anderson, J. R. (2005). *Cognitive psychology and its implications*. Macmillan.

7. Basawapatna, A. R., Koh, K. H., & Repenning, A. (2010, June). Using scalable game design to teach computer science from middle school to graduate school. In *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education* (pp. 224-228). ACM.

8. Bhagi, A. (n.d.). App Inventor Concept Cards. Retrieved August 26, 2016, from http://appinventor.mit.edu/explore/sites/all/files/ConceptCards/ai 2/AI2_ConceptCards.pdf

9. Biddle, R., & Tempero, E. (1998). Java pitfalls for beginners. *ACM SIGCSE Bulletin*, *30*(2), 48-52.

10. Brooks, R. (1983). Towards a theory of the comprehension of computer programs. *International journal of man-machine studies*, *18*(6), 543-554.

11. Chmiel, R., & Loui, M. C. (2004). Debugging: from novice to expert. *ACM SIGCSE Bulletin*, *36*(1), 17-21.

12. Clancy, M. (2004). Misconceptions and attitudes that interfere with learning to program. *Computer science education research*, 85-100.

13. Cooper, S., Pérez, L. C., & Rainey, D. (2010). K--12 computational learning. *Communications of the ACM*, *53*(11), 27-29.

14. Corney, M., Lister, R., & Teague, D. (2011, January). Early relational reasoning and the novice programmer: swapping as the hello world of relational reasoning. In *Proceedings of the Thirteenth Australasian Computing Education Conference-Volume 114* (pp. 95-104). Australian Computer Society, Inc..

15. CS Unplugged; http://csunplugged.org. Accessed 2016 June 21.

16. CSTA. http://csta.acm.org/, 2017.

17. Dancik, G., & Kumar, A. (2003, November). A tutor for counter-controlled loop concepts and its evaluation. In *Frontiers in Education Conference* (Vol. 1, pp. T3C-7). STIPES.

18. Dann, W. P., Cooper, S., & Pausch, R. (2011). Learning to Program with Alice (w/CD ROM). Prentice Hall Press.

19. De Raadt, M., Watson, R., & Toleman, M. (2002, June). Language trends in introductory programming courses. In *Proceedings of Informing Science and IT Education Conference* (pp. 329-337).

20. Deimel Jr, L. E. (1985). The uses of program reading. *ACM SIGCSE Bulletin*, *17*(2), 5-14.

21. Détienne, F., & Soloway, E. (1990). An empirically-derived control structure for the process of program understanding. *International Journal of Man-Machine Studies*, *33*(3), 323-342.

22. Du Boulay, B. (1986). Some difficulties of learning to program. *Journal of Educational Computing Research*, *2*(1), 57-73.

23. Du Boulay, B., O'Shea, T., & Monk, J. (1981). The black box inside the glass box: presenting computing concepts to novices. *International Journal of Man-Machine Studies*, *14*(3), 237-249.

24. Enome, Inc. (n.d.). Literacy Manipulatives. Retrieved August 26, 2016, from https://goalbookapp.com/toolkit/strategy/literacymanipulatives

25. Fincher, S., & Petre, M. (Eds.). (2004). *Computer science education research*. CRC Press.

26. Fitzgerald, S., Simon, B., & Thomas, L. (2005, October). Strategies that students use to trace code: an analysis based in grounded theory. In *Proceedings of the first international workshop on Computing education research* (pp. 69-80). ACM.

27. Guzdial, M. (2015). Learner-centered design of computing education: Research on computing for everyone. *Synthesis Lectures on Human-Centered Informatics*, *8*(6), 1-165.

28. Hadjerrouit, S. (1998). Java as first programming language: a critical evaluation. *ACM SIGCSE Bulletin*, *30*(2), 43-47.

29. Halasz, F., & Moran, T. P. (1982, March). Analogy considered harmful. In *Proceedings of the 1982 conference on Human factors in computing systems* (pp. 383-386). ACM.

30. Herman, G. L., Loui, M. C., & Zilles, C. (2010, March). Creating the digital logic concept inventory. In *Proceedings of the 41st ACM technical symposium on Computer science education* (pp. 102-106). ACM.

31. Hourcade, J. P. (2015). Child-computer interaction. *Self*.

32. ISTE. http://www.iste.org/, 2017.

33. Keilman, J. (2016, January 2). Coding education rare in K-12 schools but starting to catch on. *Chicagotribune.com*. Retrieved March 3, 2017, from http://www.chicagotribune.com/news/ct-coding-high-school-met-20160101-story.html

34. Kelleher, C., Pausch, R., & Kiesler, S. (2007, April). Storytelling alice motivates middle school girls to learn computer programming. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 1455-1464). ACM.

35. Kessler, C. M., & Anderson, J. R. (1986). Learning flow of control: Recursive and iterative procedures. *Human-Computer Interaction*, *2*(2), 135-166.

36. Kolikant, Y. B. D. (2001). Gardeners and cinema tickets: High school students' preconceptions of concurrency. *Computer Science Education*, *11*(3), 221-245.

37. Kolikant, Y. B. D. (2004). Learning concurrency: evolution of students' understanding of synchronization. *International Journal of Human-Computer Studies*, *60*(2), 243-268.

38. Kollmansberger, S. (2010, June). Helping students build a mental model of computation. In *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education* (pp. 128-131). ACM.

39. Krathwohl, D. R. (2002). A revision of Bloom's taxonomy: An overview. Theory into practice, 41(4), 212-218.

40. Kuittinen, M., & Sajaniemi, J. (2004). Teaching roles of variables in elementary programming courses. *ACM SIGCSE Bulletin*, *36*(3), 57-61.

41. Kumar, A. N. (2013, July). A study of the influence of code-tracing problems on code-writing skills. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education* (pp. 183-188). ACM.

42. Le, P. (2015, December 2). Growing push to expose more students to computer science. *Phys.org*. Retrieved March 3, 2017, from https://phys.org/news/2015-12-expose-students-science.html

43. Learning Resources, “Research on the benefits of manipulatives;” http://www.learningresources.com/text/pdf/Mathresearch.pdf Accessed 2016 June 23.

44. Lewis, C. M. (2010, March). How programming environment shapes perception, learning and goals: logo vs. scratch. In *Proceedings of the 41st ACM technical symposium on Computer science education* (pp. 346-350). ACM.

45. Lewis, C. M. (2012, September). The importance of students' attention to program state: a case study of debugging behavior. In *Proceedings of the ninth annual international conference on International computing education research* (pp. 127-134). ACM.

46. Lifelong Kindergarten Group (MIT), “Blocks;” http://wiki.scratch.mit.edu/wiki/Blocks. Accessed 2016 June 18.

47. Lister, R. (2011, January). Concrete and other neo-Piagetian forms of reasoning in the novice programmer. In *Proceedings of the Thirteenth Australasian Computing Education Conference-Volume 114* (pp. 9-18). Australian Computer Society, Inc..

48. Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., ... & Simon, B. (2004, June). A multi-national study of reading and tracing skills in novice programmers. In *ACM SIGCSE Bulletin* (Vol. 36, No. 4, pp. 119-150). ACM.

49. Lister, R., Fidge, C., & Teague, D. (2009, July). Further evidence of a relationship between explaining, tracing and writing skills in introductory programming. In *ACM SIGCSE Bulletin* (Vol. 41, No. 3, pp. 161-165). ACM.

50. Littman, D. C., Pinto, J., Letovsky, S., & Soloway, E. (1987). Mental models and software maintenance. *Journal of Systems and Software*, *7*(4), 341-355.

51. Lopez, M., Whalley, J., Robbins, P., & Lister, R. (2008, September). Relationships between reading, tracing and writing skills in introductory programming. In *Proceedings of the fourth international workshop on computing education research* (pp. 101-112). ACM.

52. Malan, D. J., & Leitner, H. H. (2007). Scratch for budding computer scientists. *ACM SIGCSE Bulletin*, *39*(1), 223-227.

53. Maloney, J. H., Peppler, K., Kafai, Y., Resnick, M., & Rusk, N. (2008). *Programming by choice: urban youth learning programming with scratch* (Vol. 40, No. 1, pp. 367-371). ACM.

54. Mannila, L., Peltomäki, M., & Salakoski, T. (2006). What about a simple language? Analyzing the difficulties in learning to program. *Computer Science Education*, *16*(3), 211-227.

55. McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B. D., ... & Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin*, *33*(4), 125-180.

56. Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2013). Learning computer science concepts with scratch. *Computer Science Education*, *23*(3), 239-264.

57. Milbrandt, G. (1993). Using problem solving to teach a programming language in computer studies. *Journal of Computer Science Education*, *8*(2), 14-19.

58. Moore, S. D. (n.d.). Why Teach Mathematics with Manipulatives? Retrieved August 26, 2016, from ETA hand2mind; http://www.hand2mind.com/pdf/Benefits_of_Manipulatives.pdf. Accessed 2016 June 18.

59. Nanja, M., & Cook, C. R. (1987, December). An analysis of the on-line debugging process. In *Empirical studies of programmers: second workshop* (pp. 172-184). Ablex Publishing Corp.

60. Papert, S. (1996). An exploration in the space of mathematics educations. *International Journal of Computers for Mathematical Learning*, *1*(1), 95-123.

61. Parsons, D., & Haden, P. (2007, July). Programming osmosis: Knowledge transfer from imperative to visual programming environments. In *Conference of the National Advisory Committee on Computing Qualifications. Citeseer*.

62. Paul, J. L., Kouril, M., & Berman, K. A. (2006, March). A template library to facilitate teaching message passing parallel computing. In *ACM SIGCSE Bulletin* (Vol. 38, No. 1, pp. 464-468). ACM.

63. Pea, R. D. (1986). Language-independent conceptual “bugs” in novice programming. *Journal of Educational Computing Research*, *2*(1), 25-36.

64. Pennington, N. (1987, December). Comprehension strategies in programming. In *Empirical studies of programmers: second workshop* (pp. 100-113). Ablex Publishing Corp.

65. 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.

66. Repenning, A. (2000). AgentSheets®: An interactive simulation environment with end-user programmable agents. *Interaction*.

67. Repenning, A., Basawapatna, A., & Escherle, N. (2016, September). Computational thinking tools. In *Visual Languages and Human-Centric Computing (VL/HCC), 2016 IEEE Symposium on* (pp. 218-222). IEEE.

68. Repenning, A., Webb, D., & Ioannidou, A. (2010, March). Scalable game design and the development of a checklist for getting computational thinking into public schools. In Proceedings of the 41st ACM technical symposium on Computer science education (pp. 265-269). ACM.

69. Resnick, M., Bruckman, A., & Martin, F. (1996). Pianos not stereos: Creating computational construction kits. *interactions*, *3*(5), 40-50.

70. Resnick, M., et al. “Scratch: Programming for everyone.” Communications of the ACM 52, 11 (2009): 60-67.

71. Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer science education*, *13*(2), 137-172.

72. Rusk, N. “Scratch cards;” http://scratched.gse.harvard.edu/resources/scratch-cards-0. Accessed 2016 July 21

73. Samurcay, R. (1989). The concept of variable in programming: Its meaning and use in problem-solving by novice programmers. *Studying the novice programmer*, *9*, 161-178.

74. Schollmeyer, M. (1996, March). Computer programming in high school vs. college. In *ACM SIGCSE Bulletin* (Vol. 28, No. 1, pp. 378-382). ACM.

75. Shaffer, D. (1986). The use of Logo in an introductory computer science course. *ACM SIGCSE Bulletin*, *18*(4), 28-31.

76. Shaffer, D. W., & Resnick, M. (1999). " Thick" Authenticity: New Media and Authentic Learning. *Journal of interactive learning research*, *10*(2), 195.

77. Sheard, J., Carbone, A., Lister, R., Simon, B., Thompson, E., & Whalley, J. L. (2008, June). Going SOLO to assess novice programmers. In *ACM SIGCSE Bulletin* (Vol. 40, No. 3, pp. 209-213). ACM.

78. Shneidennan, B. (1986). Empirical studies of programmers: The territory, paths, and destinations. In *Empirical Studies of Programmers: First Workshop* (Vol. 1, p. 1). Intellect Books.

79. Soloway, E. (1986). Learning to program= learning to construct mechanisms and explanations. *Communications of the ACM*, *29*(9), 850-858

80. Spohrer, J. C., & Soloway, E. (1986). Novice mistakes: Are the folk wisdoms correct?. *Communications of the ACM*, *29*(7), 624-632.

81. Spohrer, J. G., & Soloway, E. (1986). Analyzing the high frequency bugs in novice programs.

82. Stephenson, C., & West, T. (1998). Language choice and key concepts in introductory computer science courses. *Journal of Research on Computing in Education*, *31*(1), 89-95.

83. Stolee, K. T., & Fristoe, T. (2011, March). Expressing computer science concepts through Kodu game lab. In Proceedings of the 42nd ACM technical symposium on Computer science education (pp. 99-104).

84. Teague, D., & Lister, R. (2014, January). Longitudinal think aloud study of a novice programmer. In *Proceedings of the Sixteenth Australasian Computing Education Conference-Volume 148* (pp. 41-50). Australian Computer Society, Inc..

85. Tew, A. E. (2010). *Assessing fundamental introductory computing concept knowledge in a language independent manner* (Doctoral dissertation, Georgia Institute of Technology).

86. Touretzky, D. S. (2014). Teaching Kodu with physical manipulatives. ACM Inroads, 5(4), 44-51.

87. Touretzky, D. S. (n.d.). Kodu Curriculum Modules. Retrieved July 26, 2016, from https://www.cs.cmu.edu/~dst/Kodu/Curriculum/

88. Touretzky, D. S. (n.d.). Kodu Idiom Flash Cards & Tiles. Retrieved July 26, 2016, from https://www.cs.cmu.edu/~dst/Kodu/

89. Touretzky, D. S., Gardner-McCune, C., & Aggarwal, A. (2016, February). Teaching Lawfulness With Kodu. In Proceedings of the 47th ACM Technical Symposium on Computing Science Education (pp. 621-626). ACM.

90. Touretzky, D. S., Gardner-McCune, C., and Aggarwal, A. (2017) Semantic reasoning in young programmers. *Proceedings of SIGCSE '17*, Seattle, WA. Association for Computing Machinery.

91. Touretzky, D. S., Marghitu, D., Ludi, S., Bernstein, D., & Ni, L. (2013, March). Accelerating K-12 computational thinking using scaffolding, staging, and abstraction. In Proceeding of the 44th ACM technical symposium on Computer science education (pp. 609-614). ACM.

92. Vainio, V., & Sajaniemi, J. (2007, June). Factors in novice programmers' poor tracing skills. In *ACM SIGCSE Bulletin* (Vol. 39, No. 3, pp. 236-240). ACM.

93. Van Rossum, G. (1999). Computer programming for everybody. *Proposal to the Corporation for National Research Initiatives*.

94. Vee, A. (2013). Understanding computer programming as a literacy. *Literacy in Composition Studies*, *1*(2), 42-64.

95. Westervelt, E. (2014, February 14). A Push To Boost Computer Science Learning, Even At An Early Age. *Npr.org*. Retrieved March 3, 2017, from http://www.npr.org/sections/alltechconsidered/2014/02/17/271151462/a-push-to-boost-computer-science-learning-even-at-an-early-age

96. Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33-35.

97. Wing, J. M. (2011, March). Computational thinking. In *VL/HCC* (p. 3).

98. Winslow, L. E. (1996). Programming pedagogy—a psychological overview. *ACM Sigcse Bulletin*, *28*(3), 17-22.

BIOGRAPHICAL SKETCH

Ashish Aggarwal completed his Bachelor of Technology degree in Computer Science and Engineering from Jaypee University of Information Technology, India in 2015.

He received Master of Science degree in Computer Science and Management from the University of Florida, Gainesville in 2017. His research interest was in studying Computational Thinking and K-12 Computer Science Education where his work focused on understanding how elementary students reason about programs.