Neo-Piagetian Classification of Reasoning Ability and Mental Simulation in Microsoft's Kodu Game Lab

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

ABSTRACT

Over the past five years, there has been a major push to develop the computational thinking skills of K-12 students. Tools such as Scratch, Alice, and Kodu have been developed to engage students in learning to program through the creation of computational artifacts (e.g., games, animations, and stories). However, less is known about how elementary and middle school children reason about program behavior. Such skills are useful for reading and adapting others programs, locating possible sources of bugs, and predicting program behavior given code snippets (i.e., mental-simulation). The goal of this poster is to measure and track the development of students' ability to reason about programs using Teague & Lister's Neo-Piagetian classification of novice programmers: Sensorimotor, Preoperational Thinkers, and Concrete Operational Thinkers. We operationalize Teague and Lister's category descriptions by creating a criterion for each category. This classification has helped us characterize students' mastery of strategies for reasoning about the lawful behavior of programs using a Kodu curriculum. In particular, this categorization was used to differentiate students' reasoning styles using data from two studies having 20 and 19 students each. We found strong consistency in the results across both studies. Through analysis and categorization of student responses, most students fall into the preoperational thinker category. Within this category, we found a diversity of mastery patterns that help us understand where students face challenges in reasoning about programs.

Keywords

Computational Thinking; Neo-Piagetian; Kodu; Mental Simulation; Reasoning Ability; Lawfulness; K-12;

1. PROBLEM AND MOTIVATION

Many researchers have studied how novice programmers learn to program using text-based programming languages. However, the process by which students transition from various stages of program understanding, reasoning, and mastery of concepts has not been studied at length. Using Microsoft Kodu's Game Lab, we aim to study how students reason about programs. Kodu is an ideal environment to explore how students reason about programs because the simple rule structure allows students to create programs with a wide range of behaviors quickly. In addition, there are simple laws that govern how "[program rules] are evaluated, actions are sequenced, and state is updated" [6]. Thus, using Kodu we expect novice students will be able to interpret Kodu syntax and computational rules and predict program behavior. This will allow us to explore the developmental stages students undergo while learning to reason about programs.

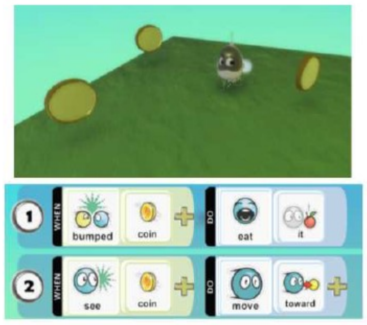
SIGCSE '17, March 8–11, 2017, Seattle, WA, USA.
ACM 978-1-4503-4698-6/17/03.
DOI: http://dx.doi.org/10.1145/3017680.3022469

2. BACKGROUND AND RELATED WORK

Microsoft Kodu Game Lab is a tile-based visual programming language built specifically to develop 3D game development. It uses WHEN-DO conditional rules to construct valid syntax. Dr. David S. Touretzky at Carnegie Mellon University has developed a Kodu curriculum [3] focused on developing reasoning about programs based on lawfulness [6]. Over 100 students have gone through the program. The curriculum teaches students about lawful reasoning about program behavior by teaching them a set of simple laws that govern rule evaluation and actions in Kodu worlds. To help students learn to reason about programs, the curriculum uses physical manipulatives such as tiles and flashcards [4] to help students learn basic computational rules and design patterns. The flashcards have basic design patterns or idioms which give initial actions that can be programmed in Kodu. Tiles are meant to scaffold students' rule construction and rule recognition ability.

At the end of each module, students are given a set of assessment questions to gauge their mastery of content covered in the module and development of their reasoning skills. Figure 1 demonstrates a type of question students are given to evaluate their program reasoning skills. We expect students to lawfully and logically interpret Kodu's syntax after participating in the Kodu sessions. For example, in Kodu's Apple 1 World (Figure 2), students see 5 apples scattered throughout the world and a student is expected write a program to pursue and consume the apples. When asked to predict the order in which the Kodu character will eat the apples the students are expected to reason using Law 1 which states "Each rule picks the closest matching object". Thus, a student who is able to lawfully reason about the program would say that the rules state the Kodu character will pick the closest matching apple then eat it. Then it will continue on to the next closest matching apple and eat it, repeating this sequence until all the apples are gone.

In previous research, we have analyzed the end of module assessments to map out students' performance on these questions and to identify the kind of errors students make [6]. In our research to date, we have identified several common mistakes and misconceptions students have that affect their ability to correctly reason about the behavior of Kodu programs. Our next step in this research is to understand the relationship between these misconceptions and errors and students' development of program reasoning skills.



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 1: The Question asked to mentally simulate the syntax.

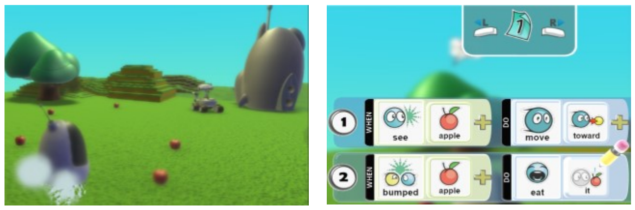


Figure 2: Kodu apple 1 world (left) and rule editor (right).

Teague and Lister [1, 5] have earlier categorized novice programmers and have researched the development of novice students' ability to program. They characterize novice programmers into one of three categories: sensorimotor, pre-operational or concrete operational thinker. Sensorimotor novice programmers represent the "least mature stage" of cognitive development. They possess "fragile domain knowledge as disjointed snippets which they find difficult to piece together in any satisfactory manner" [1]. At the next stage, preoperational novice programmers are able to "more reliably trace code" but do not really understand the "relationships between different parts of the code" [1]. Concrete operational novice programmers are able to reason "at a more abstract level" They have developed the "ability to see the whole and its parts at the same time" [1]. We use this classification as a reference to understand and characterize the different states of reasoning students are able to demonstrate.

3. APPROACH AND UNIQUENESS

The research presented in this poster is focused on characterizing and understanding the mastery and reasoning abilities of elementary school students. We use Neo-Piagetian classifications [2] as it captures the development of reasoning skills which is important for understanding the extent to which students are able to lawfully reasoning about Kodu programs. In particular, we separate and classify the assessment questions based on the required reasoning ability such as mental simulation/program prediction ability, rule recognition, or simple understanding of the concepts. This helps us in categorizing the reasoning ability of novice programmer using the Neo-Piagetian classifications. For example, a student who can successfully mental simulate and predict the program behavior demonstrates a more abstract form of reasoning and thus exhibiting the quality of a concrete operational thinker. While a student who analogically recognize a behavior or concept without regard to the laws, exhibits the qualities of a preoperational thinkers.

We internally validated the classification of questions and students reasoning ability using the overall score of students on the assessments. Thus, if students scored low on the assessments and only correctly answered simple understanding questions, they were classified as sensorimotor novice programmers. Simple understanding questions included those asking students to state the laws or recognize the basic functionality of rules. Often, the sensorimotor students had difficulty in the understanding the application of the rules and laws. We then further classify pre-operational thinkers as students who made random or consistent errors in applying or interpreting the laws and predicting the program behavior. In this poster, we evaluate the effectiveness of this classification on student mastery of the first three modules of the Kodu curriculum with 39 students, having 20 and 19 students each in two different studies to evaluate and classify students' reasoning ability as sensorimotor, pre-operational or concrete operational thinking.

4. RESULTS AND CONTRIBUTION

Our observations and results suggest that most of the students in our study were pre-operational reasoners after the first three modules and this is consistent across the two studies reported in this poster. 15 out 20 students were classified as pre-operational in the first study and 15 out of 19 in the second study. 9 students in each study demonstrated a pattern of incorrect responses which could be attributed to simple misconceptions. This helped in identification of common misconceptions among students. These results show a snapshot of students' reasoning ability and an approach for operationalizing the Neo-Piagetian Novice Programmer Classifications developed by Teague & Listers [1,5]. This work will help CS educators to reflect on how to assess students computational understanding and their reasoning abilities. In addition, it can help us to reflect on what supports are needed to develop students in concrete operational thinkers and reasoners. Additional studies of novice programmers and their program comprehension ability will help the field understanding the common misunderstandings the students are developing and how can we improve their reasoning abilities.

5. ACKNOWLEDGMENTS

Funded by a gift from Microsoft. Thanks to Dr. Christina Gardner-McCune for advising the research and Dr. David S. Touretzky for his guidance and support.

6. REFERENCES

[1] Teague, D., & Lister, R. (2014, June). Programming: reading, writing and reversing. In Proceedings of the 2014 conference on Innovation & technology in computer science education (pp. 285-290). ACM.

[2] 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.

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

[4] Touretzky, D. S. (2014). Teaching Kodu with physical manipulatives. ACM Inroads, 5(4), 4451.

[5] Teague, D., Corney, M., Ahadi, A., & Lister, R. (2013, January). A qualitative think aloud study of the early neo-piagetian stages of reasoning in novice programmers. In Proceedings of the Fifteenth Australasian Computing Education Conference-Volume 136 (pp. 87-95). Australian Computer Society, Inc.

[6] 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