The Fairy Performance Assessment: Measuring Computational Thinking in Middle School

Linda Werner
University of California
Santa Cruz, CA
831-459-1017
linda@soe.ucsc.edu

Jill Denner
ETR Associates
4 Carbonero Way
Scotts Valley, CA 95066
831-438-4060
jilld@etr.org

Shannon Campe
ETR Associates
shannonc@etr.org

Damon Chizuru Kawamoto
Brown University
chchchez@aol.com

ABSTRACT
Computational thinking (CT) has been described as an essential capacity to prepare students for computer science, as well as to be productive members of society. But efforts to engage K-12 students in CT are hampered by a lack of definition and assessment tools. In this paper, we describe the first results of a newly created performance assessment tool for measuring CT in middle school. We briefly describe the context for the performance assessment (game-programming courses), the aspects of CT that are measured, the results, and the factors that are associated with performance. We see the development of assessment tools as a critical step in efforts to bring CT to K-12, and to strengthen the use of game programming in middle school. We discuss problems and implications of our results.

Categories and Subject Descriptors
K.3.2 [Computers and Education]: Computer and Information Science Education – *Computer science education.*

General Terms
Measurement.

Keywords
Assessment, Computational Thinking, Game Programming, Middle School, Pair Programming, Alice.

1. INTRODUCTION
Computational thinking (CT) was first described by Papert [10], and then pioneered by Wing [16]. At its core, it is closely related to scientific reasoning defined as the specification of a problem, use of resources for inquiry and hypothesis generation, model building and experimentation to test hypotheses, and the evaluation of evidence [4] [17]. Additional features of CT include creating artifacts and automation. Little is known about the development of CT in K-12, although recent articles begin to describe what it looks like [3][6].

One factor that limits the uptake of CT into K-12 is the lack of assessments. The ACM and CSTA report titled "Running on Empty: Failure to Teach K-12 Computer Science in the Digital Age" [15] states "Assessments for computer science education are virtually non-existent." To our knowledge, the only existing measures of pre-college CT were created by Repenning et al [11] yet there have been numerous recent reports of measures of computer science conceptual understanding thought to be useful in the acquisition of CT including [5][12][13].

The purpose of this study was to develop and test a CT performance assessment for middle school students, and to understand why there is variation across students, in order to develop and strengthen efforts to engage K-12 students in CT. The study was conducted in computer game programming classes held after school and during electives. Classes were randomly assigned to either solo or pair programming conditions, based on research that shows programming with a partner is beneficial in terms of retention and performance to both male and female university students learning to program [17]. We predicted that some of these benefits would be observed from pair programming over solo programming for middle school students too.

Prior work provides the basis for both the type of assessment we designed and for the choice of attributes we used in our search for which factors explain variance in students' CT. Most relevant to our choice for assessment style is the work by Webb [14] who reports success with a post-test assessment used in after-school classes of 24 middle school students creating 3D games with AgentCubes, a programming environment designed for youth. Webb used five faulty scenarios to assess his students' skill in CT. He observes that the administration of equivalent pre- and post-test assessments "for software-dependent instruction is problematic as student fluency requires some degree of familiarity with the software." Problems arise because students could take hours to become familiar enough with the programming environment so that working on the assessment would make sense. Webb's faulty scenarios are based on programming patterns the students learn in class but are interspersed with faults. Most important for our choice for attribute analysis is the work by Linn [9] who is an early investigator in the area of assessment of problem solving skills among novice programmers. Linn described the Chain of Cognitive Accomplishments [9], which includes three links: Comprehension: understand programs and be able to make small changes to single program instructions; Design: build programs from collections of patterns and use procedural skills to combine these patterns to solve problems; and Problem Solving: learn transferable problem-solving skills and use them in new and different formal systems. In this paper, we use Linn's three-link chain metaphor to describe the different cognitive demands in our assessment, and to interpret variation in our results.

2. METHOD

2.1 Participants

The data described in this paper were collected over two years as part of a study of how game creation and pair programming can promote CT. A total of 325 students with parental consent participated in classes at seven public schools along the central California coast. Participation in all parts of the study was voluntary and we are reporting on 311 students who submitted a Fairy assessment. In year 1, 78 students participated in after-school classes, and in year 2, 37 participated in after-school and 196 in elective technology classes. More than half of the students (57%) were in the pair condition; however, due to absences, only 21% of the total were partnered all the time, and 23% most of the time.

Of the 311 students, 36% were female; they ranged in age from 10-14 years (mean=12), 52% were white, 37% were Latino/a, and 75% spoke English or primarily English at home. Twenty five percent had mothers with educational levels of high school or lower, and 39% of mothers had completed a university degree.

2.2 Procedure

Students used one of the two programming environments in the Alice [2] series developed at Carnegie Mellon. In year 1, students used Storytelling Alice (SA), and in year 2 they used Alice 2.2 because SA is only available on PCs, which were limited at our partner schools. In the rest of this paper, we will refer to both SA and Alice 2.2 as Alice unless it is necessary to distinguish between the two. Alice allows users to control characters in 3D environments using drag-and-drop programming, a language that is closely related to Java, and many other modern imperative programming languages. Most code is written in the methods of objects that have properties that store state and functions that return values. Each property, method, and function is attached to an object, with World being the global object. The event system in Alice is primarily used to handle user interactions, such as mouse clicks, although it can also handle in-World events, such as when the value of a variable changes.

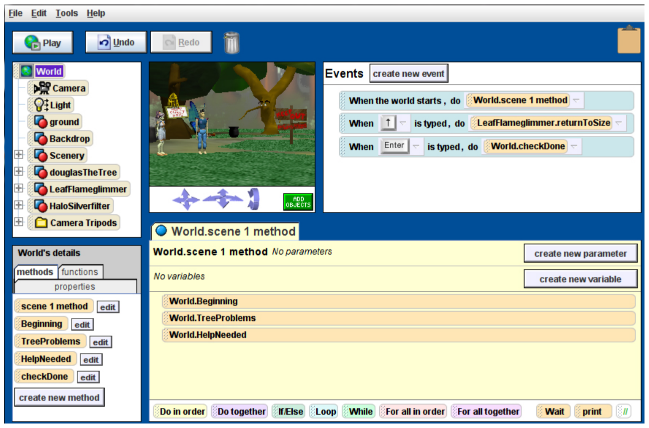
Students engaged with CT in a three-stage progression called Use-Modify-Create[6] over approximately 20 hours during a semester. In the first half of the semester, students worked through a series of self-paced instructional exercises built to provide scaffolding, which we call "challenges." During the last half of the course, the students freely designed and developed their own games. Most students completed 8 to 10 of 11 required challenges, though some completed up to six additional optional challenges. In our courses where pair programming was used, two students shared one computer, with one driving (controlling the mouse and keyboard) and the other navigating (checking for bugs, consulting resources, and providing input). We asked students to reverse roles approximately every 20 minutes. At the end of the semester, students were given up to 30 minutes to *individually* complete the Fairy Assessment described below. The data collectors and the instructions did not refer to it as a test or assessment, but we told students we designed it to help us know what students learned in the classes. While working on the assessment, students could ask for clarification of instructions only.

2.3 Measures

Students completed an on-line survey at the beginning and end of the semester. The items include demographic questions about age, gender, race/ethnicity, language spoken at home, mother's and father's use of computers at work, grades on last report card, and favorite subject at school. Additional questions measure confidence with computers (e.g., I feel confident about my ability to use computers), attitude toward computers (e.g., I would like to learn more about computers), frequency and type of computer use, and simple Alice content knowledge (8 multiple-choice items with screen shots of images and sample code). Classroom teachers collected attendance data that we used to compute total number of hours attended, as well as total number of hours that students in the paired condition worked with a partner. The parent completing the consent form reported parent education levels.

3. THE FAIRY ASSESSMENT

We created the Fairy Assessment as an Alice program to analyze two of the three parts of CT identified by the Carnegie Mellon Center for Computational Thinking (CMCCT): thinking algorithmically, and making effective use of abstraction and modeling. We did not include an assessment task to measure considering and understanding scale, the third part of CT as identified by the CMCCT, since we did not include scale topics in our set of required challenges and few students experimented with aspects of scale in their games. In Year 1, we gave some of the directions on paper; we gave other directions via game character dialog. In Year 2, in an effort to standardize the instructions, all directions were communicated both via game character dialog and repeated as itemized text instructions displayed at the bottom of the program screen throughout the exercise. The top level of the program is a sequence of method calls each containing a series of Alice primitive control statements and built-in method calls. See Figure 1.



```
When the world starts , do  World.scene 1 method
When  ↑  is typed, do  LeafFlameglimmer.returnToSize
When  Enter  is typed, do  World.checkDone
```

```
World.scene 1 method  No parameters
No variables
  World.Beginning
  World.TreeProblems
  World.HelpNeeded
```

Figure 1: Initial screen layout.

We believe that in order for students to perform well on the assessment, they have to understand the narrative framework of the story underlying the program and to understand the existing program instructions that create that framework. The assessment features two fairies: HaloSilver (female) and LeafFlame (male). When the student plays the program, they see HaloSilver talking, and LeafFlame talking about walking into a forbidden forest where he shrinks to one-half his original size because of a magic spell. He asks for help because he's stuck in the forest and can't fly (also the result of a magic spell). Figure 2 is a screen shot showing LeafFlame walking into the forbidden forest. We designed three separate and independent assessment tasks with varying levels of difficulty so that failure in any one task did not dictate failure in any other. We designed the tasks to measure aspects of algorithmic thinking and abstraction and modeling. The following is an overview of each task.

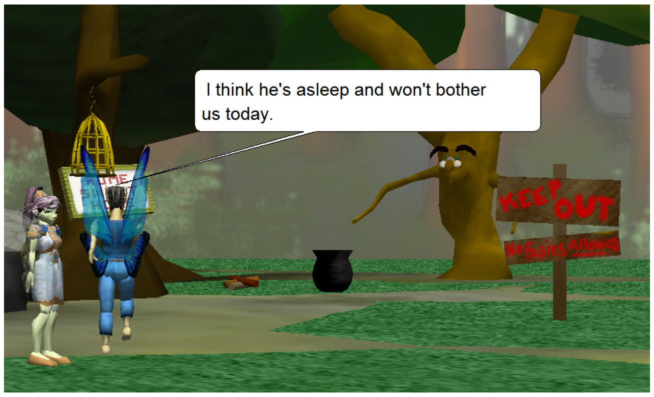
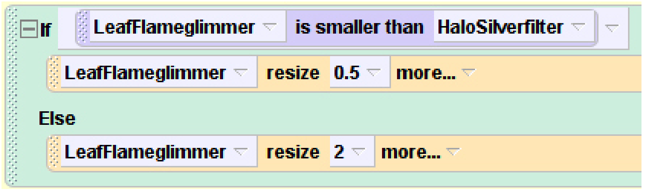


Figure 2: LeafFlame walking into forbidden forest.

Task 1 instructions say to program HaloSilver so that she turns to watch LeafFlame as he walks into the forbidden forest; the turning should take the same amount of time that it takes LeafFlame to talk. Students that do this correctly understand the basic narrative framework of the story and can correctly place instructions within a given finite sequence of instructions (algorithm). They also recognize the need to modify a method parameter: the length of time of execution of an instruction (abstraction and modeling). We assigned full credit for this task (10 points) if students recognized the need for concurrent execution of this new turn method call along with an existing LeafFlame walk method call into the forest and if the instruction length was modified appropriately. We assigned students 4 points (partial credit) if HaloSilver turned somewhere in the vicinity of LeafFlame's walk. Both of these solutions suggest the student has achieved the first link in Linn's chain of cognitive accomplishments because he/she understands someone else's program code enough to make simple changes or additions.

In Task 2, HaloSilver asks the student to repair the program; HaloSilver says, "Make pressing the up arrow work right and return LeafFlame to his original size." We expected Task 2 to be the most difficult. To solve task 2, students need to think algorithmically and understand that event handlers alter a program's default execution sequence. They need to recognize the presence of an event handler for the up arrow and verify it is not working correctly. Correcting the method called in the 'up arrow event handler' (LeafFlame's returnToSize method) shows algorithmic thinking because it shows the student understands how to read, interpret, and correct or replace the conditional logic in this method. See Figure 3. We assigned credit of from 2 to 8 points for partial solutions. For example, in one solution attempt, a student added another event handler for the up arrow that called one of LeafFlame's built-in methods: resize with parameter set to '2'. If this code was present **and** the initial handler removed, this solution would have earned the student 10 points because it would clearly demonstrate a cognitive accomplishment including the second link in Linn's chain of cognitive accomplishments: build programs from collections of patterns and use procedural skills to combine these patterns to solve problems. Unfortunately, the student left both up arrow event handlers in the program; indicative of movement into the second link of Linn's chain.



```
If  LeafFlameglimmer is smaller than HaloSilverfilter
    LeafFlameglimmer resize 0.5 more...
Else
    LeafFlameglimmer resize 2 more...
```

Figure 3: Faulty logic of returnToSize method.

For Task 3, HaloSilver asks students to help LeafFlame fly out of the forest toward her; she says, "Make clicking on LeafFlame bring him to me and FLY out of the forest!" See Figure 4.

Make clicking on LeafFlame bring him to me and FLY out of the forest!
KEEP OUT
Figure 4: HaloSilver providing instructions.

To solve task 3, again students need to think algorithmically and understand that events alter the program's default execution sequence. This required students to recognize the need for, and then add another event handler. The complexity of Task 3 was high because the intended method call for this event handler ('flyAway') uses two parameters: 'to whom to fly' and 'number of times to fly.' Students need to think on another level of abstraction and navigate the object tree to locate LeafFlame and his built-in methods. The students must also have an understanding of abstraction to use the method 'flyAway towardWho numberOfTimes,' and analyze it in a detailed level to determine values for the parameters. Within-program feedback told the student if he or she was successful in helping LeafFlame: LeafFlame's size and position are checked. Partial credit of 8 points was assigned to students who created a 'mouse-click-on-LeafFlame' event handler calling one of LeafFlame's built-in methods: 'move toward' with parameter 'toward-who' set to HaloSilver. This solution makes LeafFlame move, but not 'fly,' out of the forest toward HaloSilver. Full credit of 10 points was assigned to a very creative solution: the student created a new method local to LeafFlame where LeafFlame's wings' flapped as he hopped up and down which made him appear to fly toward HaloSilver. The student called this method in a 'mouse-click-on-LeafFlame' event handler.

Successful solutions to Tasks 2 and 3 use knowledge of programming patterns, or segments of non-contiguous code that together form a complex computational construct. The use of a programming pattern is an example of a design skill that Linn [9] calls templates. She writes, "Templates are stereotypic patterns of code that use more than a single language feature. Templates perform complex functions such as sorting names alphabetically or summing a set of numbers. Templates can be used each time a given task is encountered. A large repertoire of templates enables the programmer to solve many problems without creating new code." The patterns we expected from our students include event handling, complex conditional use, and use of methods with multiple parameters. Partial scores of 4 in either of these two tasks constitute movement into the second link in Linn's chain of cognitive accomplishments.

We argue that non-zero performance in Tasks 2 or 3 represents some movement toward the second link in the chain of cognitive accomplishments because it shows that the student has a partial understanding of someone else's code and can modify portions of programming patterns to accomplish the tasks. The second link involves the design of a program using templates and skill in combining these templates. Completely understanding Task 2 requires code understanding, fault identification, code modification to remove the fault and places the student at the beginning of the third link in the chain.

4. DATA ANALYSIS

Two of the paper's authors scored the students' solutions to the Fairy Assessment. We graded each task on a scale from zero to ten, with partial credit possible, resulting in a maximum score of 30. The scorers discussed discrepancies and came to an agreement on all students' scores. The scoring rubric is available upon request from the first author.

The tasks were summarized separately, and although the average score on each task varied, the three task scores were significantly correlated (range is from r=.30 to r=.45). In this paper, we combined the task scores into a single total Fairy score for some analyses. We used bivariate analyses using t-tests and correlations to identify key factors from the pre- and post-survey data that were related to total Fairy scores.

5. RESULTS AND DISCUSSION

Mean (and standard deviation - σ) scores for the Fairy Assessment are given in Table 1.

Table 1. Fairy Assessment Scores, n=311

Task	Mean (σ)
1 (comprehension)	6.03 (3.87)
2 (comprehension, design, problem solving)	4.54 (4.73)
3 (comprehension, design)	5.93 (4.34)
Total score	16.50 (9.95)

Student performance was highest on Task 1, which measures comprehension, and lowest on Task 2, which measures comprehension, design, and complex problem solving skills using debugging. There was great variation in scores: 30 students submitted an unchanged program and were assigned a score of zero on all three tasks, while 40 students had perfect scores on all three tasks. There was not a significant difference between average scores for students from year 1 using SA (mean=16.13, standard deviation (sd)=9.38, n=78) and year 2 using Alice 2.2 (mean=16.63, sd=10.15, n=233).

Pair programming was found to be significantly positively related to assessment scores at the p<0.01 level: the more time a student spent with a partner while learning and programming a game, the higher their individual score on the assessment (r=.17). Since some students missed some classes, we maintained records of amount of time in a paired state while learning and programming their games. Several pre- and post-survey factors were significant at the p < 0.05 level. Several parental factors were positively related to total score: parent education (r=.31), and having a mother (yes mean=19.01, sd= 9.10; no mean=14.96, sd=10.18) and/or father (yes mean=19.79, sd=8.60; no mean=14.08, sd=10.31) that works with computers. In addition, students scored higher if they also reported speaking more English at home (r=.30). Several student factors at post-survey were positively correlated with Fairy Assessment scores: interest in taking a computer science class in high school (r=.14), grades in school (r=.25), frequency of computer-use across locations (r=.13), confidence with computers (r=.26), attitude toward computers (r=.18), and Alice content knowledge measured at post-survey time (r=.54). Total scores were not related to gender, student age, hours of attendance in the class, or their frequency of creating things using the computer outside this class.

The findings suggest that the Fairy assessment is a promising strategy for assessing CT during middle school. First, it was motivating: all but 30 of the students that worked on it submitted a modified program. Fourteen students were absent on the day of the assessment; there were no refusals. Webb [14] found similar results with the use of his assessment requiring trouble-shooting skills to debug faulty scenarios. He found middle school-aged students to be engaged for up to 45 minutes on his assessment.

Our results also suggest that the assessment was successful in picking up a range of CT across students, and a variety of types of CT across the three tasks. Mean scores for Task 2 were lowest, because it required students to debug the conditional logic in the method in the 'up arrow' event handler. Not all students completed challenges that taught conditional logic; this may have contributed to the lower scores on this task. Our findings are similar to what Linn [9] found in her sample of middle school students using BASIC: more progress was made in comprehension than in design. As a comparison, her assessment also did not address the third link in the chain of cognitive accomplishments.

The data provide some explanation of why student performance varied. For example, the scores for students in elective classes were higher (mean=17.94, sd=9.64) than those taking the class after school (mean=14.05, sd=10.02). The data confirm that this was not due to differences in SA and Alice 2.2. However, it may be related to the improved instructions for year 2 when all the in-school classes were held or to the higher levels of parent education among students in year 2. Another possible explanation is that students working in an in-school environment may be more attentive, more concerned about performance and grading, and therefore, learn more or try harder on an assessment.

Students that worked using pair programming scored higher than those that programmed alone. Additionally, we found the more time with a partner was related to higher CT performance. This is important because it tells us that pair programming was an effective technique to use for engaging middle school students in CT. Some educators express concern about whether both students benefit or whether the weaker student lets the stronger student work and reap the educational benefits. We found that both students, regardless of initial ability, scored higher than students that worked alone. This finding is consistent with what has been reported by researchers of pair programming at the university level [17]. Our results differ from those of Lewis [8] who gave daily quizzes to students entering sixth grade enrolled in a 3-week summer camp. She found statistically significantly greater variation in quiz scores between the students that shared one computer over the variation between students that worked on their own computer with help from an assigned collaborator to discuss problems. However, Lewis' study was of a very small number of students (40) and partners switched roles every 5 minutes instead of the longer time period we used. More research needs to be done to determine the circumstances under which pair programming benefits learning in middle school, however it appears to be useful for improved performance on the Fairy Assessment.

Not surprisingly, parent capacity was also related to students' scores on the assessment. The two measures, parent education and use of computers at work, indicate socioeconomic resources as well as exposure to computing outside of school were positively associated with assessment score. There were some students who did well on the assessment, even though their parents had low education and did not use computers at work. More detailed case studies are needed to understand the factors that helped some students with lower parent capacity to score high on the assessment.

Several behavioral factors were related to scores. More frequent computer use and higher grades on their last report card were related to higher performance on the Fairy Assessment. This suggests that students with greater access to computers, and those that do better academically, also engage in higher levels of CT. Similarly, students who are interested in taking a computer science class in high school score higher. These findings are not surprising, and unfortunately without a pre-test of the Fairy Assessment we cannot tell if the more advanced and interested students improved more over time, or started out at a higher level of CT.

Similar to results found by Levine and Donitsa-Schmidt [7] in a study of grade 7-12 students, we found confidence with computers to be an important predictor of performance. This may be due to a willingness to persist in the face of a challenging task, or to just having more experience. In our sample, confidence was also significantly correlated with frequency of computer use, and further analyses are needed to determine the relative importance of each of these factors. Levine and Donitsa-Schmidt suggest that confidence also predicts attitudes, which was another significant predictor of students' scores.

Some limitations of our study are that we are reporting bivariate associations that do not let us make conclusions about the relative importance of each factor. Many of these variables (significant and not significant) are highly correlated, and multi-level modeling is necessary to tease out the interrelationships. Another limitation is that the data collected from pairs is not independent, which violates certain statistical assumptions. Similarly, mean scores varied greatly across the seven school groups, and the interdependence of these scores is not taken into account. These types of data require more sophisticated dyadic data analysis, and multi-level modeling. Our next step is to conduct these analyses.

Another limitation of this study is the lack of a test of construct validity—whether the Fairy Assessment really measures the aspects of CT that we claim. To this end, we compared students' Alice content knowledge measured at post-survey and found it was positively related to performance on the Fairy Assessment. These knowledge questions assessed a student's understanding of someone else's program code, which is the first link in Linn's chain of cognitive accomplishments, and a key aspect of each of the tasks of the Fairy Assessment.

While the Fairy Assessment itself may not be generalizable to learning environments not using Alice, others who are trying to measure CT can easily adapt this assessment. Each task should be designed to measure certain aspects of CT (e.g., in our case we measured algorithmic thinking and abstraction and modeling), and certain links in the chain of cognitive accomplishments. Care must be taken to describe the assessment tasks in language that is consistent with the style used by the game characters. The tasks should be described to students without suggesting the use of specific program constructs (that would test programming syntax knowledge but not higher levels of knowledge needed for CT) but instead by describing requirements the solution must meet using the context of narrative of the game's virtual world. We believe we have done some of this with our Fairy Assessment.

It is possible that, with the current drag-and-drop programming interfaces that prevent syntactic errors, middle school students could complete a course using one environment (for example, Alice), and then work on an assessment using another environment (for example, Scratch). This would allow us to more accurately assess whether students accomplished the third link in the chain of cognitive accomplishments.

6. CONCLUSIONS AND FUTURE WORK

Elective technology and computer game courses for middle school students are a promising strategy to introduce CT to a broad population. This is an important step towards engaging groups that are currently underrepresented in the computer science community. However, without the ability to assess whether and how such programs can engage students in CT, they will likely never see widespread adoption.

In this paper, we continue the work pioneered by others in addressing the issue of CT by middle school students. We still have quite a few things to learn about the definition and assessment of CT in middle school youth. We are currently analyzing the 'challenges,' self-paced instructional materials, to identify aspects of CT that are taught in each of the challenges. We are also working on analysis of the log files to identify steps in students' solutions, which includes trying to understand what they tried before reaching a final solution, and their 'intent'. The Fairy Assessment logs can show us whether students place the correct instruction at the correct program location in one attempt or if they make multiple attempts. We want to look closer at the students' Fairy Assessments to determine if mistakes are indicative of CT misunderstandings or reflect specific issues in the mechanics of Alice programming. We are also experimenting with a translation of the Fairy Assessment into a Scratch version, which was recently implemented in two different Scratch courses. The Scratch assessment has three tasks that are very similar to those in the Fairy Assessment. We hope to report on these experiments soon. Also, we are looking at the results of other assessment methods used in this study. These include looking for evidence of CT in the students' games (identifying programming constructs and CT patterns similar to those described in [5] [12][11] and in the logs captured during game creation. Finally, we are coding videotapes of the pairs working together to see if the quality of interaction while students were learning Alice and making their games affects how well individual students do on the assessment.

7. ACKNOWLEDGMENTS

Our thanks go to the teachers and administrators at our seven schools, specifically Anne Guerrero, Shelly Laschkewitsch, Don Jacobs, Sue Seibolt, Karen Snedeker, Susan Rivas, and Katie Ziparo. Thanks also to teaching assistants, Will Park and Joanne Sanchez; and to Eloy Ortiz and Pat Rex, whose vital support was necessary to run this program. Thanks to all of the students who participated. Finally, thanks to Barb Ericson, Irene Lee, Clayton Lewis, Fred Martin, Alex Repenning, David Webb, Teale Fristoe, and Caitlin Sadowski for help on early drafts of this paper. This research is funded by a grant from NSF 0909733 "The Development of Computational Thinking among Middle School Students Creating Computer Games." Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

8. REFERENCES

[1] ACM K-12 Task Force Curriculum Committee. A Model Curriculum for K-12 Computer Science: Final Report of the ACM K-12 Task Force Curriculum Committee, http://www.csta.acm.org/Curriculum/sub/ACMK12CSModel.html.

[2] Alice website, September 1, 2011. www.alice.org.

[3] Barr, V. and Stephenson, C., 2011. Bringing computational thinking to K-12: what is Involved and what is the role of the computer science education community? ACM Inroads 2, 1, 48-54.

[4] Bransford, J.D. and Donovan, M.S., 2005. Scientific inquiry and how people learn. In National Research Council, How students learn: History, mathematics, and science in the classroom. Washington DC: The National Academies Press.

[5] Denner, J., Werner, L. and Ortiz, E., 2012. Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts? Computers & Education, 58(1), 240-249.

[6] Lee, L., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J., and Werner, L., 2011. Computational thinking for youth in practice. ACM Inroads 2, 1, 32-37.

[7] Levine, T. and Donitsa-Schmidt, S., 1998. Computer use, confidence, attitudes, and knowledge: A causal analysis, Computers in Human Behavior, Volume 14, Issue 1, 125-146.

[8] Lewis, C., 2011. Is pair programming more effective than other forms of collaboration for young students? Computer Science Education Vol. 21, No. 2, 105-134.

[9] Linn, M. C., 1985. The Cognitive Consequences of Programming Instruction in Classrooms. Educational Researcher, Vol. 14, No. 5, pp. 14-16+25-29 at http://www.jstor.org/stable/1174202.

[10] Papert. S., 1993. Mindstorms: Children, Computers, and Powerful Ideas. 2nd ed. NY: Basic Books.

[11] Repenning, A., Webb, D., and Ioannidou, A., 2010. Scalable Game Design and the Development of a Checklist for Getting Computational Thinking into Public Schools. SIGCSE '10, 265-269.

[12] Rodger, S.H., Hayes, J., Lezin, G., Qin, H., Nelson, D., Tucker, R., Lopez, M., Cooper, S., Dann, W., and Slater, D., 2009. Engaging middle school teachers and students with Alice in a diverse set of subjects. SIGCSE Bull. 41, 1, 271-275.

[13] Stolee, K.T. and Fristoe, T., 2011. Expressing computer science concepts through Kodu game lab. In Proceedings of the 42nd ACM technical symposium on Computer science education (SIGCSE). ACM, New York, NY, USA, 99-104.

[14] Webb, D.C., 2010 Troubleshooting assessment: an authentic problem solving activity for it education, Procedia - Social and Behavioral Sciences, Volume 9, World Conference on Learning, Teaching and Administration Papers, pp. 903-907.

[15] Wilson, C., Sudol, L., Stephenson, C., and Stehlik, M., 2010. Running on Empty: The Failure to Teach K-12 Computer Science in the Digital Age. ACM Available as: http://www.acm.org/runningonempty/fullreport.pdf.

[16] Wing, J., 2006. Computational Thinking. CACM vol. 49, no, 3. March 2006, pp. 33-36.

[17] Werner, L., Hanks, B., & McDowell, C., 2004. Pair programming helps female computer science students. ACM Journal of Educational Resources in Computing, 4(1).

[18] Zimmerman, C., 2005. The development of scientific reasoning skills. Accessed on Sept. 1, 2011 from www7.nationalacademies.org/bose/Corinne_Zimmerman_Final_Paper.pdf.