Assessment of Computer Science Learning in a Scratch-Based Outreach Program

Diana Franklin†, Phillip Conrad†, Bryce Boe†, Katy Nilsen‡, Charlotte Hill†, Michelle Len†, Greg Dreschler†, Gerardo Aldana§, Paulo Almeida-Tanaka†, Brynn Kiefer†, Chelsea Laird♯, Felicia Lopez§, Christine Pham†, Jessica Suarez§, Robert Waite†

† Department of Computer Science
‡ Gevirtz School of Education
§Chicana/o Studies Department
♯Biology Department
UC Santa Barbara

ABSTRACT

Many institutions have created and deployed outreach programs for middle school students with the goal of increasing the number and diversity of students who later pursue careers in computer science. While these programs have been shown to *increase interest* in computer science, there has been less work on showing whether participants *learn computer science content*.

We address two questions, one specific, and the other more general: (1) "What computer science did our middle school students learn in our interdisciplinary two-week summer camp?" (2) "How can computer science concepts be assessed in the context of Scratch-based outreach programs"? We address both questions by presenting the design of our summer camp, an overview of our curriculum, our assessment methodology, and our assessment results.

Though the sample size is not statistically significant, the results show that a two-week, interdisciplinary, non-academic summer camp can be effective not only for engaging students, but also for imparting CS content. In just two weeks, with a curriculum not entirely focused on computer science, students displayed competence with event-driven programming, initialization of state, message passing, and say/sound synchronization[1]. We have employed assessment methodologies that avoid written exams, an approach both outreach and classroom-based programs may find useful.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education; K.4.m [**Computers and Society**]: Miscellaneous—*Diversity and Outreach*

General Terms

Design, Human Factors

[1]A timing issue between say bubbles and sound in Scratch

SIGCSE'13, March 6–9, 2013, Denver, Colorado, USA.

Keywords

diversity, K-12 education, outreach, assessment, Scratch

1. INTRODUCTION

The desire to expose more students to computer science has led to the development of many educational activities[15, 9, 12, 5] and outreach programs to broaden participation in computer science [4, 3, 10] . Unfortunately, these outreach programs can reach only a small segment of the population. To truly broaden participation in computing, we must help computer science find a place at the table in K-12 schools—and that requires the development of standards, curricula, and assessments.

Previous work shows that Scratch-based outreach can be effective at increasing students' *interest* in computing [19, 10] . With this paper, we contribute to a small but growing body of work assessing the effectiveness of Scratch for teaching CS *content*—in our case, in the context of a two-week interdisciplinary summer camp for middle school students, based on Scratch programming [15] presented along with culturally-relevant non-CS themes aimed at broadening participation among underrepresented groups.

We present this work both to (1) address the question of whether outreach activities can really teach computer science ("Sure, they had fun, but did they learn anything?") and (2) provide a starting point for further development of assessment for Scratch programming; techniques that can be adapted not only to assessing outreach activities, but also in any computing curriculum based on Scratch programming.

We found that in just two weeks of our interdisciplinary camp, students displayed competence in several areas, including event-driven programming, initializing state, synchronization between audio and visual elements as well as between objects, and some proficiency in creating complex animations requiring the integration of several concepts including loops. Our assessment techniques resulted in fairly low-overhead note-taking and program analysis, with very little reliance on written exams.

The rest of the paper is organized as follows. We provide a brief summary of related work in Section 2. Section 3 describes our camp and curriculum. Sections 4 and 5 describe our assessment methodology, and the results of our assessment. Finally, Section 6 describes future work.

2. RELATED WORK

We are adding to a small but growing body of work assessing student learning with Scratch programming—work that spans a va-

riety of approaches, settings, and contexts. A general framework for such assessments is described by Brennan and Resnick [7].

Adams and Webster[1] conducted a thorough analysis of programs from nine years of CS-centric summer camps for middle and high school students. By inspecting which and how many blocks of each type were used, they answered interesting questions about how the project types, gender, etc. influenced the CS constructs used.

A series of papers [16, 17, 11] assessed learning of CS concepts through Scratch in the school setting, examining, among other things, whether the programming approaches learned through Scratch were beneficial or detrimental to later success in CS. Their methodology used traditional classroom tests.

Burke and Kafai [8] found that students following a similar curriculum to ours (digital story-telling) in a school setting (vs. our summer-camp setting) over seven weeks demonstrate competence in several key CS concepts (event-driven programming, synchronization/coordination of multiple threads, and loops). Like us, they had a small sample size (10).

Lewis and Shah [14] describe an assessment of Scratch programming quizzes in a summer enrichment course focused primarily on math and computer science and containing high-achieving students. They administered paper-based quizzes on the 2nd through 10th day of instruction in a 12 day course.

Building on this prior work, our work has two major differences. First, we adapt the assessment methodology to a non-academic camp setting with a broad spectrum of students (academically and culturally). We cross-reference help received with the programs to more accurately reflect the competence, and we analyze the implementation of the Scratch programs deeply to rate the competence displayed. Second, we show that with just a two-week, interdisciplinary, non-academic camp program, students demonstrate competence in many CS areas.

3. CAMP AND CURRICULUM

Animal Tlatoque, our two-week summer camp, was initially designed with two specific goals: (1) to attract a target audience of middle school students from underrepresented groups with non-CS themes[2, 13, 18] that appeal to both parents and children (at which we have succeeded [10]), and (2) to engage participants in interdisciplinary activities that allow them to learn about computer science and develop skills for computational thinking in the context of those non-CS themes. This past year, we took on a third goal: (3) to assess the learning of CS content that took place during the camp.

The Summer 2012 instance of our camp had 35 campers. Ten of these were enrolling in the camp for a second or third year; these campers progressed through the Scratch curriculum on an accelerated track. Our assessment efforts are focused on students participating in the camp for the first time; of these we had IRB permission and informed consent to report data for 22 campers.

3.1 Curriculum Design Constraints

The curriculum design was heavily influenced by two constraints:

(1) The interdisciplinary themes influenced the computer science *content* of the lessons because of the culminating project—a digital story-telling project.

Week 1 lessons consisted entirely—and only—of lessons that taught skills and concepts needed to complete that final project. Concepts not strictly needed for that project—such as user-defined variables and conditional control flow—were deferred to Week 2, and students had less opportunity to practice them.

(2) Assessment influenced the *structure* of the lessons. Each lesson has two parts: a warm-up exercise (not assessed) in which students learn one or more new CS concepts with substantial scaffolding and support, and a small project (assessed) in which students apply their new knowledge to a similar, but new, problem.

Lesson	event-driven prog.	Initial State: size	Initial State: orientation	Initial State: background	Initial State: visibility	Initial State: costume	Initial State: position	synchronization	broadcast/receive	complex animation
1) Scratch Basics	X	X								
2) Conversations	X	X	X					X	X	
3) Scene Changes	X	X	X	X	X			X	X	
4) Complex Anim.	X	X	X			X	X			X
Final project	X	X	X	X	X	X	X	X	X	X

X: item required by the project's checklist.

Table 1: Computer science concepts assessed through Scratch projects.

Our core Scratch curriculum is summarized in Table 1. The columns to the right of the table show the major CS concepts that formed the learning objectives, and the targets of assessment, namely: Event-Driven Programming, Initial State, Synchronization, Broadcast/Receive, and Complex Animation. In addition, we offered an additional lesson covering interactive input, user-defined variables and conditional control flow (if/then/else) and an optional project involving a side-scroller game. The remainder of this section describes each of these lessons in more detail.

3.2 Lesson Descriptions

Students work in pairs at a single computer. Each pair is assigned an endangered animal from Mesoamerica that is used in several projects.For reasons of space we omit detailed descriptions of the warm-up exercises.

Lesson 1: Scratch Basics familiarizes students with the basic elements of Scratch programming, including concepts such as the stage, sprites, the way blocks are combined into scripts and two core CS concepts.

Event-driven programming is used in interactive programs, using blocks such as "When Green Flag clicked" and "When Sprite clicked" to determine when actions should occur.

Initial state, in particular its initialization, was previously observed (anecdotally) to be a common source of bugs. Sprites have certain attributes (e.g.position, orientation) that, if changed during the program, need to be initialized if the "green flag" is clicked again to restart after completion or the "red flag" is clicked.

Lesson 1 Project: Name Poem [20] is based on the camper pair's assigned animal—the name of their animal is spelled out, and when each letter is clicked, a descriptive word appears starting with that letter appears. For example, the character "fish" might become Friendly, Intelligent, Shy, Hairless.

Lesson 2: Conversations teaches multi-sprite synchronization through "conversations" between two characters, where the dialog is simultaneously in say bubbles and sound, requiring a tricky Scratch solution. This introduces two new CS concepts.

Broadcast/Receive is used to control timing between sprits. In Scratch programming, a collection of scripts can be associated with each sprite, and each script runs as a separate parallel thread. To create order between threads, students use broadcast/receive blocks. This can be considered an example of the more general CS concepts of "message-passing" and "event driven programming".

Say/Sound Synchronization is used when displaying a "say" bubble – the kind used to show dialog in comics – at the same time as an audio file narrating the text. It turns out not be nearly as straightforward to implement as one might hope. We found the most robust and pedagogically straightforward solution to be placing an indefinite say bubble on the screen, playing the sound until finished, and then removing the say bubble using a blank say bubble. This approach emphasizes an important CS concept—one so basic, that we sometimes forget that novice programmers have to *learn* it—the importance of putting blocks in the correct *sequence* to achieve the desired effect.

Lesson 2 Project: Animal Conversation allows students to teach others about their animals. Students split up and pair with others, implementing a conversation between their two animals in which they are trying to learn about each other. This reinforces their new knowledge in synchronization.

Lesson 3: Scene Changes expands on message passing and sequential instructions by introducing visibility(hide/show) and backgrounds in order to make scene changes.

Lesson 3 Project: Mayan Conversation reinforces message passing, sequential instructions, and visibility (hide/show) to make a conversation in the Mayan language. Students are given the first two glyphs and recordings. They sequence them like scenes.

Lesson 4: Complex Animation teaches students how to create realistic motion (e.g. people walk, birds fly, snakes slither, fish swim), and requires several CS concepts to complete.

Complex Animation goes beyond the simple "glide to" block provided by Scratch. To make the motion realistic requires costume changes, motion, timing, and repetition control structures (loops).

Lesson 4 Project: Name Poem w/ Motion has students add several complex animations to their name poem from lesson 1, including one of their animal.

Culminating Project: Animating a Mesoamerican Animal Myth involving their animal, containing at least three scenes and two characters. A checklist contained require elements to ensure that the resulting Scratch program would allow us to assess CS concepts taught.

Lesson 5: Interactive Conversation was taught after the students had already started the culminating project, and consisted of a series of warm-ups introducing input, variables, and conditionals (if/then/else).

Optional Project: Side-Scrolling Game contains a main character (Scratch the cat) who must battle foes such as the banana-throwing monkey, a field of thorns, bees, and a giant bat, with nothing more than foul-smelling fish as weapons. Although too complex for students to code on their own, students explored parts of implementation. Due to the lack of a project, we used a written quiz-like assessment (Table 4) for the concepts taught in this lesson.

4. ASSESSMENT METHODOLOGY

We faced several challenges in designing an assessment plan for our curriculum. The major hurdles were (1) the short timescale (this is a 2-week camp that combines several subjects, so time is very tight) (2) the student perspective that camps are supposed to be fun, so anything that reminds them of "school" (e.g. anything that smells like a "test") is to be avoided, and (3) our use of pair programming for all projects (preventing data-gathering by individual student).

In particular, we did not control for prior knowledge using pretests, nor did staff members take extensive notes because of negative camper feedback last year. We did ask about prior programming experience on a survey, but found this to be inaccurate because campers differ in their interpretation of what "programming" is. Instead, we relied on informal discussions with campers, and only two reported specific programming knowledge (one in Scratch). For this reason, we are not stating definitively that students *learned* these concepts within the two-week camp, but that they *demonstrated competence* by the end of the two-week camp.

#	Explanation
0	**Validation:** Students want confirmation, not information
1	**Where:** Only needed help navigating the Scratch gui
2	**What:** Only needed a reminder of the name of the concept
3	**How:** Given name of concept, still needed help to complete task
4	**Reteach:** Had to reteach the entire lesson (concept and execution)

Table 2: Help levels recorded in field notes.

Our major artifacts are the completed projects, but it is not sufficient to analyze these in isolation, since completion of the project does not necessarily imply competence in the material. Throughout the camp, we had at least one staff member available for every six students, so students were able to receive ample help, including being stepped through the solution, if necessary.

Therefore, we use a combination of analysis of the artifacts and information from the staff members about what help they gave the students on the projects associated with each lesson, as well as the culminating project. Note that we do not assess competence based on completion of the warm-up lessons—we assess only the final projects in each lesson.

While working, students have access to a set of Scratch Reference Cards, including a relevant subset of the original MIT Scratch Cards as well as cards we designed for each lesson. These are tied directly to the project in the lesson, and they are used as reference for the smaller projects and culminating projects. If a student completes the project using only Scratch cards, we consider that the student learned/understood the material.

4.1 Staff Notes

Campers have staff members available for help. Each time a student asks a question related to a concept that has been taught, staff members explore the level of understanding of the student. The levels of help are listed in Table 2. The goal of the staff member is to first attempt to give help as if the student required only level 0 help, then proceed to level 1, etc., until the student is able to figure out the solution. As long as at least level 1 help was given, the staff member records the level of help given and the concept covered. Some also recorded level 0 help.

4.2 Project Analysis

Each assessed project has a list of concepts required to complete it, as shown in Table 1. For the competence, we looked at the projects in two ways—the execution and the implementation. For each category, we have four levels. The top level is "perfect"—looking at the project playing and the implementation. The next level is that the project itself looks fine when played, but the implementation is not robust or did not apply the computer science concept correctly (i.e. synchronization has a potential timing issue or the student used repetitive code where a loop would have been more appropriate). The next level is that there is a noticeable defect when running the project (i.e. two sounds play at the same time, or the animation was too fast to see). The lowest level is used when the implementation is absent or incomplete—i.e. a critical piece is missing.

Our assessment involved several phases. First, for each project, three students independently scored the first five programs manu-

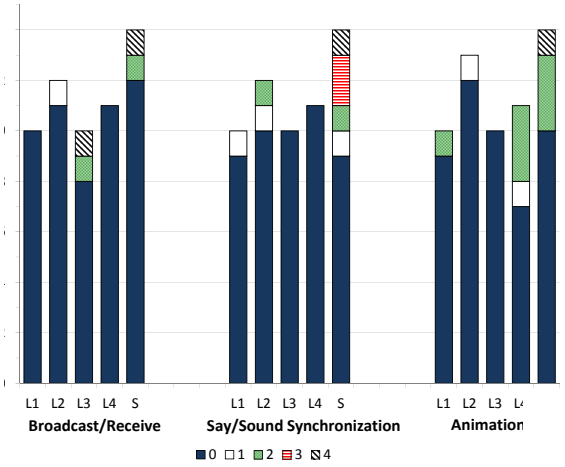

Figure 1: Field notes taken by staff members.

ally, discussed and reconciled their scores, independently scored the rest of the programs, and reconciled the rest of their scores. Our Scratch static analysis tool, Hairball, was developed to determine the scores as a fourth entity [6]. Whenever Hairball's results disagreed with what the manual assessors concluded, a final, more detailed manual assessment occurred to determine which was correct.

5. ASSESSMENT RESULTS

In this section, we present the results of our assessment. We preface these results with a frank recognition of the fact that the population under study is very small—results are reported for only 10 campers pairs. Our purpose in presenting these results, therefore, is not to claim anything "statistically significant". Rather it is to provide a model for integrating assessment of learning into the design of a Scratch-based curriculum—one that can be followed both by future designers of outreach programs, and in-school curricula alike. This allows us to not only gain insights into student competence in certain subjects, it can tell us if students were rushed through particular assignments.

In our camp, because of the complex integration of different subjects and the scheduling involved, students were a little bit rushed with the projects for Lessons 3 and 4 (Mayan Conversation and Name Poem with Motion).

5.1 Field Notes

Each staff member recorded a note for each interaction that required them to help the students. Figure 1 shows the level of help given to each pair for each concept. For the most part, students asked validation question such as: "Should I do this?" and the staff member merely confirmed the students' own knowledge. Only a few pairs needed what we consider substantive help—help levels 3 and 4. We believe this is for one of three reasons. First, by placing the warm-up exercise immediately before the transfer project, students are able to ask questions and practice once a short time before being expected to complete it by themselves.

Second, because students were working in pairs, only one of the two needs to understand the material. Pairs were assigned randomly by age and school, so we had no opportunity to match them by learning speed / skill level. Anecdotally, we did find that some pairs contained one strong partner and one weak partner. Finally, students could be completing the projects incorrectly or avoiding challenging implementations. We will see that this is true for several pairs when implementing animations in the culminating project.

Lesson and Event Descrption	Results	
(1) click green flag⇒character says something	8/10	80%
(1) click letter⇒letter says word	9/10	90%
(2) click green flag ⇒conversation starts	10/10	100%
(4) click letter⇒turn into animal	6/11	55%
Final Project: click green flag⇒ story begins	10/10	100%

Table 3: Assessments of event-driven programming.

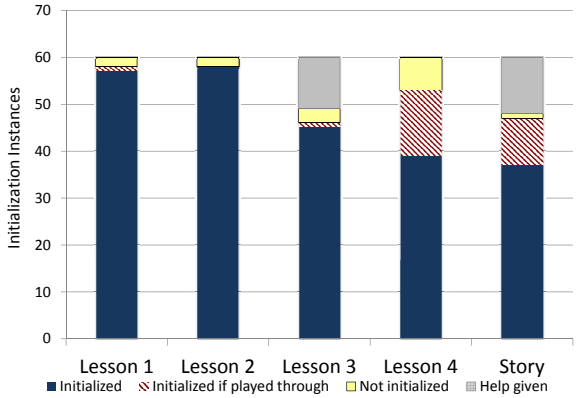

Figure 2: Initialization results broken down by project.

5.2 Event-Driven Programming

Table 3 shows part of our results of assessing understanding of event-driven programming, i.e. implementing scripts that are triggered by specific events (the other part being the results for Broadcast/Receive, discussed later in this section.)

In the project for Lesson 1, students were required to implement event-handling for both green flag and sprite clicked events—80% and 90% were successful. By Lesson 2, students has mastered "green flag" events; 100% correctly started the required actions this way as instructed.

We are not showing results for Lesson 3, because very few students completed that Lesson successfully. We believe that this, and the fact that only 55% of the students correctly implemented the assessed click event in Lesson 4 have more to do with extraneous factors; at this point in the camp, students may have felt more rushed. In addition, in Lesson 4, they may have been concentrating more on realistic motion (presented later) rather than this checklist item.

5.3 Initialization

We performed two tests for initialization. The first is to manually open the program and run it twice from beginning to end to see if the second execution matches the first. The second is to use our Hairball tool to assist in detecting if something is not initialized before it is used, leading to incorrect execution if it is stopped in the middle and restarted. Hairball is not perfect—it flags possible changes that do not have an initialization in the beginning of the program, but it misses some valid later initializations, so a manual check is still necessary.

We first show initialization broken down by project in Figure 2. The X axis shows the project, and the Y axis shows initialization errors or successes. Initialization attempts are broken down by category, not sprite, so each project has 6 initialization checks.

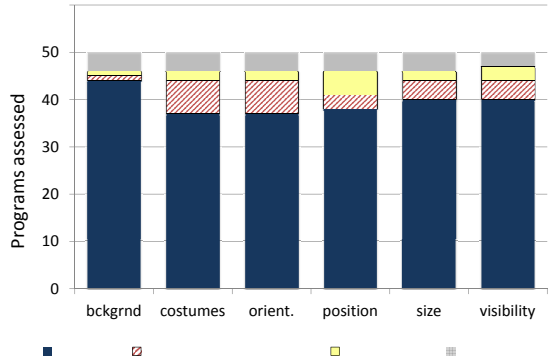

Figure 3: Initialization results broken down by category.

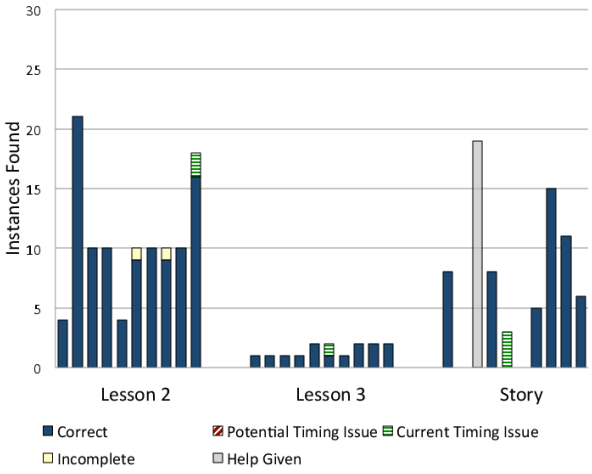

Figure 4: Results for Sound/Say Synchronization.

For the most part, students remembered to initialize items that needed to be reset when the project was played to completion and restarted. As the projects became more complex, there were more opportunities to forget initialization, especially if the program ran correctly twice in a row. For each of Lesson 3 and the story, two groups were retaught initialization (one of which was the same group both times). We do not have a record of which type of initialization was retaught, so we deducted all initialization opportunities.

We now present the initialization errors by category in Figure 3. There was not one particular category that dominated the errors. Instead, the errors were fairly evenly distributed across all the categories except for backgrounds, where there were virtually no errors.

5.4 Sound/Say Synchronization

Figure 4 shows that few of our students had any trouble with the concept of say/sound synchronization. The issues that did arise were evident on playback, so given additional time to work on the project, the pair may have been able to detect and fix the bug. There were a few students who did not choose to implement synchronized say/sound blocks in the culminating project; they chose other ways of incorporating sound.

5.5 Broadcast/Receive

Results for Broadcast/Receive are shown in Figure 5. These results show that it is often the case that a project has one incomplete Broadcast/Receive, either because they did not clean up their dead

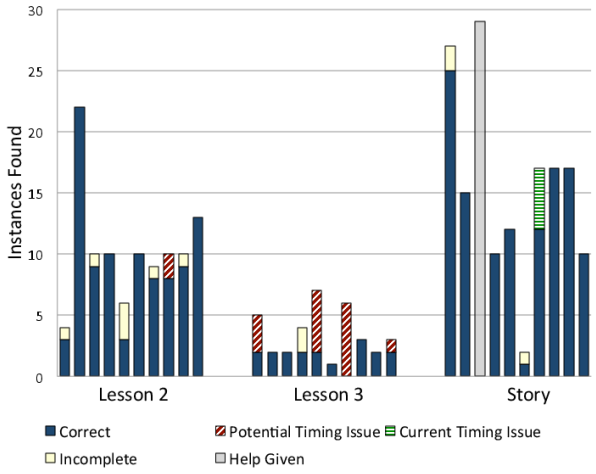

Figure 5: Results for Broadcast/Receive.

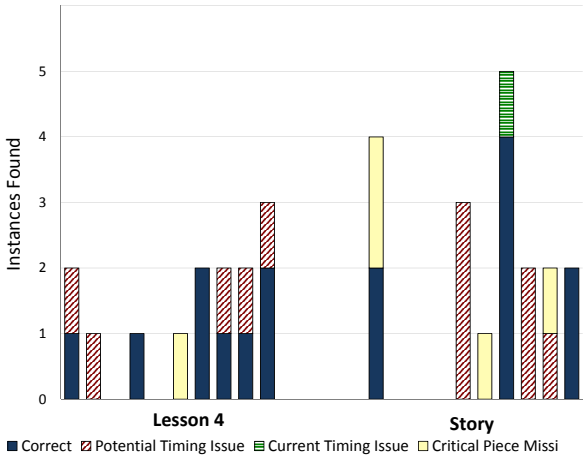

Figure 6: Results for Animation.

code or they did not quite complete the project. In Lesson 3, especially, many students were rushed in their implementations. We see that for the Story, almost all students used Broadcast/Receive properly, except for one group that required level 4 help and another that had timing issues due to two sequential broadcasts.

5.6 Complex Animation

The checklist for the name poem with motion and the culminating project specifically indicated that campers should include Complex Animation in the form of realistic motion of their animal in their projects. Most pairs added realistic motion to their name poem, but four had current timing issues, and two were missing something critical in the code. For the culminating project, we found that many students chose not to implement realistic motion, instead using the "glide" block to slide their sprite across the screen. In the end, only three pairs out of ten successfully implemented complex animation in their culminating project.

5.7 Side-Scroller Warmup Assessment

Table 4 shows the questions and results from the side-scroller quiz. Most of the students assessed knew when to employ costumes and background changes (91% and 82%). 73% of the students knew to use a repeat or forever block to make the bees keep go-

Question	Answer	Results
If you want the cat rather than a panda, what would you change?†	sprite or costume	100%
If you want the cat to go into a cave rather than a temple, what do you change?	scene or background	82%
How do you make the monkey turn red?	color effects or costume	91%
How do you make the bees keep going up and down?	repeat or forever	73%
How can you keep track of how many times the bat has been hit by a banana?	variable	73%
How do you make the bat disappear on the third hit, not the first hit?	if	28%

†Note that the first question on the assessment worksheet contained a typo—the words "cat" and "panda" should have been reversed. We are reporting the question as it actually appeared on the questionaire.

Table 4: Written assessment of concepts for side-scroller.

ing up and down and the same number knew that a variable should be used to keep track of how many times the bat had been hit. Only three pairs (28%) recognized that the difference between disappearing on the third hit rather than the first hit required an *if* block.

6. CONCLUSIONS AND FUTURE WORK

Despite the constraints of our camp—a two-week, non-academic, interdisplinary, "fun oriented" camp—we were able to perform assessments demostrating that students attained competence with several computer science concepts. In just two weeks, with a curriculum not entirely focused on computer science, students displayed—and we were able to assess—competence with event-driven programming, initialization of state, message passing, and say/sound synchronization.

Further, we have adapted and piloted a technique for asssessing computer science competence that relies very little on written assessments in the context of Scratch programming. Our technique uses a scaffolded curriculum, with warm-up lessons, projects to assess mastery, measurment of help given, and both automated and manual examination of the Scratch programs produced by students. Our approach can be adapted for other Scratch-based curricula, and/or other Computer Science concepts.

As with similar research, we have only a small sample size—thus, future work may include applying this assessment technique with a larger sample size. This could take place across multiple instances of an outreach program or in the context of a K-12 CS curriculum implemented within the regular school day.

We also note that, outreach programs, while valuable for the students that participate, reach only a small segment of the population. It is likely that integration of computer science into K-12 standards and curricula will have a broader impact. By pairing our curriculum with a curriculum teaching variables, loops, and conditionals, students can be exposed to a broad set of topics that can be applied to many different subjects required in schools today.

7. ACKNOWLEDGMENTS

This work is supported by the National Science Foundation BPC Award CNS-0940491 to Franklin, Aldana, and Conrad.

8. REFERENCES

[1] J. C. Adams and A. R. Webster. What do students learn about programming from game, music video, and storytelling projects? In *SIGCSE '12*, pages 643–648, 2012.

[2] G. Aikenhead. Students' ease in crossing cultural borders into school science. *Science Education*, 85(2):180–188, March 2001.

[3] S. Alliance. The stars alliance: A southeastern partnership for diverse participation in computing. NSF STARS Alliance Proposal. http://www.itstars.org/.

[4] I. Arroyo et al. Effects of web-based tutoring software on students' math achievement. In *AERA*, 2004.

[5] T. Bell, I. H. Witten, and M. Fellows. *Computer Science Unplugged*. 2006.

[6] B. Boe, C. Hill, M. Len, G. Dreschler, P. Conrad, and D. Franklin. Hairball: Lint-inspired static analysis of scratch projects. In *SIGCSE '13*, March 2013.

[7] K. Brennan and M. Resnick. New frameworks for studying and assessing the development of computational thinking. In *AERA 2012*, 2012.

[8] Q. Burke and Y. B. Kafai. The writers' workshop for youth programmers: digital storytelling with scratch in middle school classrooms. In *SIGCSE '12*, pages 433–438, 2012.

[9] W. Dann, S. Cooper, and R. Pausch. Making the connection: programming with animated small world. *ITiCSE*, 2000.

[10] D. Franklin, P. Conrad, G. Aldana, and S. Hough. Animal tlatoque: attracting middle school students to computing through culturally-relevant themes. In *SIGCSE '11*.

[11] M. Gordon et al. Spaghetti for the main course?: observations on the naturalness of scenario-based programming. In *ITiCSE '12*, 2012.

[12] C. S. Hood and D. J. Hood. Teaching programming and language concepts using legos. In *ITiCSE*, June 2005.

[13] C. Lee. Why we need to re-think race and ethnicity in educational research. *Educational Researcher*, 32(5):3–5, June 2003.

[14] C. M. Lewis and N. Shah. Building upon and enriching grade four mathematics standards with programming curriculum. In *SIGCSE '12*, pages 57–62, 2012.

[15] J. Maloney et al. The scratch programming language and environment. *Trans. Comput. Educ.*, 10(4):16:1–16:15, Nov. 2010.

[16] O. Meerbaum-Salant et al. Learning computer science concepts with scratch. In *ICER '10*, pages 69–76, 2010.

[17] O. Meerbaum-Salant et al. Habits of programming in scratch. In *ITiCSE '11*, pages 168–172, 2011.

[18] J. Moschkovich. A situated and sociocultural perspective on bilingual mathematics learners. *Mathematical Thinking and Learning*, 4(2/3), 2002.

[19] P. A. G. Sivilotti and S. A. Laugel. Scratching the surface of advanced topics in software engineering: a workshop module for middle school students. In *SIGCSE '08*, 2008.

[20] U. Wolz et al. Computational thinking via interactive journalism in middle school. In *SIGCSE*, 2010.