Adding computational thinking to your science lesson: what could it look like?

Abstract (200 words)
The Next Generation Science Standards (2013) call for computational thinking to be a key practice in science education. However, educators have found this to be a difficult task. Studies have shown that a set of conditions must be met for computational thinking tools to be used in K-12 education and that when they are used, there is a wide spectrum in the level of computational thinking that the tool requires. This study extends this work by examining how middle school students translate their science fair projects in Scratch and what evidence of computational thinking is present. Scraper, a tool designed to analyze Scratch projects was used. Overall, it was found that most students simply created a presentation of their project without much complexity. Eight students created interactive projects that required user participation and used more advanced computational concepts. This suggests the ability to document and begin to assess students’ computational thinking in a science education context.

Subject/Problem
In a promotional video recently released by code.org (2013), Will.i.am states “great coders are today’s rock stars”. The writers of the Next Generation Science Standards (NGSS, 2013) must have had a similar thought when making mathematical and computational thinking one of the key science and engineering practices. For example, HS-ETS1-4 states “use a computer simulation to model the impact of proposed solutions to a complex real-world problem with numerous criteria and constraints on interactions within and between systems relevant to the problem” (NGSS, 2013). However, not all modes of computational thinking are created equal or are feasible for the typical high school science classroom.

According to Basawapatna, Repenning, Koh, and Savignano (2014), examples of computational thinking in science education fall on a continuum ranging from consume to create. These examples may consist of animations, simulations, programming tools, models, or games. Each example demonstrates a tradeoff between convenience of classroom use and level of computational thinking involved. Examples on the “consume” end are animations that can be easily dropped into any class period, potentially helping students to better understand a concept, but not expressly asking students to use computational thinking. Examples on the “create” end are programming tools that require students to write some form of code, using computational thinking, in order for the computer to carry out a task (i.e. creating a game or simulation). However, these tasks use much more valuable class time. So how do we enable students to develop computational thinking in the classroom and make the best use of valuable class time?

In order to enable the effective development of computational thinking in the classroom, Repenning, Webb, and Ioannidou (2010) created a checklist of six conditions that must be met by the tool used. They include low threshold, high ceiling, scaffolds Flow, enables transfer, supports equity, and systemic and sustainable. Several of these conditions were the initial reasons for the creation of Scratch, which is a drag and drop, block based, programming language designed for use in elementary school through college (especially ages 8-16) in multiple
disciplines. It also contains a community of submitted works currently containing over 5 million projects and resources specifically for educators wanting to incorporate Scratch. For these reasons, Scratch was selected as the data source for this pilot study.

**Design/Procedure**

Brennan and Resnick (2010) describe a framework for computational thinking that can be used to help provide a better understanding of the elements of student projects and the computational thinking that underlies them. They describe the concepts, practices, and perspectives that key to computational thinking. This paper makes use of the seven concepts described in the paper (p. 3-6):

1. **Sequences**: a particular activity or action that is expressed as a series of individual steps that can be executed in order by computer
2. **Loops**: Mechanisms for running identical sequences multiple times
3. **Events**: “One thing causing another thing to occur”
4. **Parallelism**: Sequences of another thing happening simultaneously
5. **Conditionals**: Making decisions based on certain conditions, which can support multiple outcomes
6. **Operators**: Enable the programmer to preform numeric and string manipulations through mathematical, logical, and string expressions
7. **Data**: Storing, retrieving, and updating values such as variables and lists

**Research Questions**

In this study, publically available Scratch projects posted in August 2013 were used to answer the following questions:

1. How do students transfer their science fair projects to Scratch in a Digital Technologies class?
2. What evidence of computational thinking exists in science fair Scratch projects?

**Methods**

As stated above, the Scratch community is home to over 5 million shared projects posted by users. Using the search feature, “sciencefair” was entered as a keyword. Of the returned results, [Scratch Studio - Year 7 Science Fair Projects 2013](https://scratch.mit.edu/studio/265024956) was selected because it contained a set of 32 projects (29 of which were able to be analyzed) and the following description giving context to the projects: “Year 7 conducted awesome Science Fair investigations in Science. Then, in Digital Technologies they made these models of their experiments and investigations. Great work guys!” All students were in Year 7 (age 11) in New Zealand, which would be equivalent to 6th grade in the US. The projects were then descriptively coded (Saldana, 2012) to determine general characteristics of the projects. To better analyze the computational thinking concepts in the student projects, a text file of each project was loaded into Scrape (Wolz, Hallberg, & Taylor, 2011). The Scrape Local tool allows a quantitative analysis of the blocks used in the projects to be analyzed by individual project or as a group of projects.
A block is elements of code in Scratch. They are color-coded based on their function (i.e. orange is an event block). A sprite is a character or object to which the code is assigned to enact the code. For instance, if the block above was assigned to Sprite 7, when the user presses the down arrow on the keyboard, it will cause something to happen to Sprite 7 (i.e. it will move down).

**Findings and Analyses**

*How do students transfer their science fair projects to Scratch in a Digital Technologies class?*

We found that 14 of the 29 projects only allowed the user to view the project and 15 projects required user interaction (pressing keys to enable certain actions). The projects that were view only showed a rocket launch, 5 plants and the effect of sun, the life cycle of an eel, feeding chickens and others. The interactive projects showed the effect of different solutions on flowers, the phases of the moon, solutions that attract a bee, a volcanic explosion and others. Most (21 projects) of the text was achieved through broadcast, which is one of the first skills taught in basic programming. Switching costumes was a popular way (17 projects) to move or change objects in the project. However, this too is a simplistic skill in programming. Only 9 projects made use of more advanced concepts such as if/then (conditional), repeat (loop), and forever.

*What evidence of computational thinking exists in science fair Scratch projects?*

The Scrape tool shows a summary of the quantitative characteristics of the set of projects. In all, 1,496 blocks and 123 sprites were used among the 29 projects (Figure 1). Looking at the pie chart, 50.8% of all blocks used were for “looks”. These include blocks that “say” or “think” something, switch costumes, or hide.

**Figure 1.** Summary and pie chart from Scrape Local.

Control blocks were 34.4% of the total blocks used. A sample of these are shown in Figure 2. This category includes the customary starting block “when green flag
clicked”. In all, 85.2% of the total blocks used were basic functions within the projects. Only 14.8% of the blocks used consisted of more complex functions.

**Figure 2.** Control block breakdown from Scrape Local.

**Table 1.** Snapshot of data from Science Fair 2013 Scratch projects

<table>
<thead>
<tr>
<th>Proj #</th>
<th># of Scripts</th>
<th># of Sprites</th>
<th>Concepts</th>
<th># of Blocks</th>
<th>%Orange-Control</th>
<th>%Purple-Looks</th>
<th>%Blue-Motion</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>none</td>
<td>3</td>
<td>66.6</td>
<td>33.3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td>11</td>
<td>broadcast, switch costumes <em>(Sequence)</em></td>
<td>69</td>
<td>36.2</td>
<td>63.7</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>22</td>
<td>4</td>
<td>switch costumes <em>(Sequence)</em></td>
<td>43</td>
<td>50</td>
<td>45.4</td>
<td>4.5</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>7</td>
<td>broadcast, switch costumes <em>(Sequence)</em></td>
<td>64</td>
<td>26.5</td>
<td>64</td>
<td>9.3</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>6</td>
<td>Broadcast, switch costumes, <em>if/then</em> <em>(Sequence, Conditional)</em></td>
<td>222</td>
<td>26.5</td>
<td>63.5</td>
<td></td>
</tr>
</tbody>
</table>
The projects were also analyzed individually using the Scrape tool. Table 1 shows a snapshot of this analysis. Most project followed the overall trend of primarily Orange (control) and Purple (looks) blocks but 9 deviated using more sophisticated blocks (ex. Project #5). Also, in accordance with the framework developed by Brennan & Resnick (2012), sequence and events was used in almost all projects but loops, conditionals, and operators were not frequently used, demonstrating higher level concepts.

Conclusion
From these analyses, we can conclude that while more advanced computational thinking concepts are evident in these science fair projects, a more scaffolded project is needed. Our next step is to develop an “Hour of Code” science lessons in which the environment is better controlled and scaffolded to assess the students current and developing computational thinking practices.

Contribution
All ASTE members who are using or planning to use computational thinking tools will have an interest in this presentation. We will cover all levels of familiarity from what are computational thinking tools to advanced decisions regarding tool use in the classroom.

Selected References