Introducing Computer Science using Block-Based Programming

Z.D.R. Copic
Division of Computer Science
University of Minnesota, Morris
Morris, Minnesota, USA 56267
copic006@morris.umn.edu

ABSTRACT

This paper will focus on several studies dedicated to teaching Computer Science (CS) with Scratch and the results of said studies. These studies aim to make CS concepts easier to grasp and be retained. This paper will also survey various advantages in using block-based coding as an educational tool, studies done using the Scratch programming language, and the conclusions of the studies. Using Scratch could be a new and better way to teach programming at all ages given how easy Scratch is to use and manipulate. Also, this survey will focus on studies ranging from an hour after to school, to several hours a day in a Summer session.

Keywords

Block-Based Programming, Scratch, Computer Science Education

1. INTRODUCTION

With the ever present need to learn, the subject of how to teach computer science comes up on occasion. What is the best way to do it? Are there even different ways to go about teaching the concepts of computer concepts? This paper will focus on answering these questions with the block-based program Scratch and different ways to introduce computer science concepts through this language. We will begin with some background, on Scratch and other concepts that need it. We follow by some good-programming practices that should be present in all computer science coding. We will then delve into various setups to teach these concepts with Scratch, such as in a Summer Camp, After School, or in a regular class.

2. BACKGROUND

Scratch is a block-based coding system designed specifically for children ages eight to sixteen and was developed by MIT Media Lab's Lifelong Kindergarten group, led by Mitchel Resnik [8]. It is based on Squeak, which is a variant of Smalltalk, and it is an object-oriented, class-based, and reflective [9]. Reflection is the ability of a program to examine and modify its own structure and behavior at runtime [7]. Code is represented visually by blocks such as in Figure 1, and was started up in 2003 at MIT. It has garnered support from various companies such as Microsoft and Google. It has been steadily gaining steam but recently took off in 2013 with more comments from its users, more projects, and more users[1]. Also, it is used globally. It is available in more than 150 different countries and it also is translatable into forty plus languages making it a very versatile starting language. Looking at Figure 1, we can see a simple script built in Scratch. Figure 2 is the Graphical User Interface (GUI) in which the script in the second column would be interlocked together and ran. The first column has several categories in which some premade scripts are already there for the user. The third column, or pane, is where the sprite would be if there is one, and this is also where if one were to say "Hello, world!" this is where it would appear. A sprite is a two dimensional bitmap that is integrated into a larger scene. Going through Figure 1, it is read from the top to the bottom. When the green flag is clicked (the green flag is in the upper right of Figure2), the phrase "Hello, world!" will appear in the third pane of the Scratch GUI. The third block down will end the script and the phrase used previously will disappear from the GUI.

Another concept used in this section is story-boarding. Story-boarding is a way of graphically organizing ones ideas in the form of slides or panes in a sequential, visual way and is used in programming practices, theater, and any other interactive media such as in Figure 3 [10]. It is useful for identifying specifications for a given software that you or a team is developing. Furthermore, it can also help develop the interface the client wants in a much more interactive manner instead of just describing the software with a list or general outline. It also points out to the user how the given software will work and this is also a far cheaper solution than modifying software to appeal to the user.

3. GOOD PROGRAMMING PRACTICES

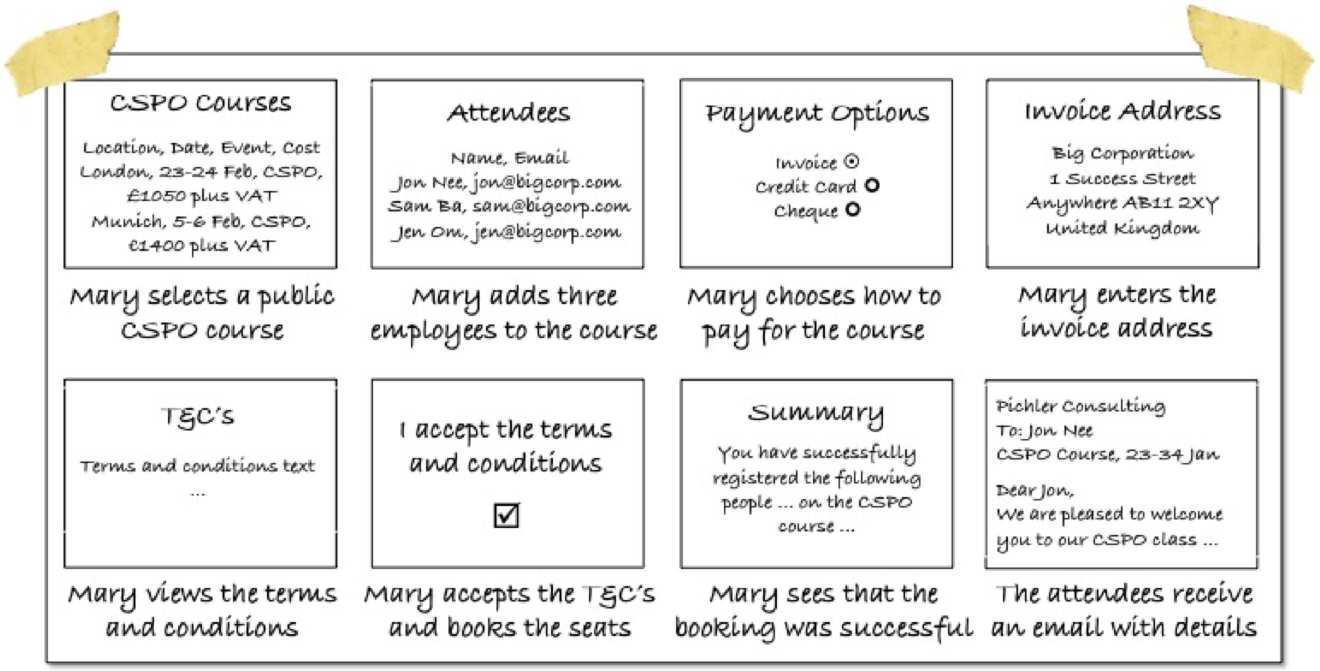
This section discusses testing and documentation. It also includes what was done in a few studies, and the problems and solutions associated with them.

3.1 Documentation

One aspect of good programming practices is proper documentation. Trying to instill this in a middle/high school-

[1]`https://scratch.mit.edu/statistics`

UMM CSci Senior Seminar Conference, April 2016 Morris, MN.



CSPO Courses

Location, Date, Event, Cost
London, 23-24 Feb, CSPO, £1050 plus VAT
Munich, 5-6 Feb, CSPO, €1400 plus VAT

Mary selects a public CSPO course

Attendees

Name, Email
Jon Nee, jon@bigcorp.com
Sam Ba, sam@bigcorp.com
Jen Om, jen@bigcorp.com

Mary adds three employees to the course

Payment Options

Invoice ●
Credit Card ○
Cheque ○

Mary chooses how to pay for the course

Invoice Address

Big Corporation
1 Success Street
Anywhere AB11 2XY
United Kingdom

Mary enters the invoice address

T&C's

Terms and conditions text
...

Mary views the terms and conditions

I accept the terms and conditions

☑

Mary accepts the T&C's and books the seats

Summary

You have successfully registered the following people ... on the CSPO course ...

Mary sees that the booking was successful

Pichler Consulting
To: Jon Nee
CSPO Course, 23-34 Jan

Dear Jon,
We are pleased to welcome you to our CSPO class ...

The attendees receive an email with details

Figure 3: An example of a what a storyboard might look like

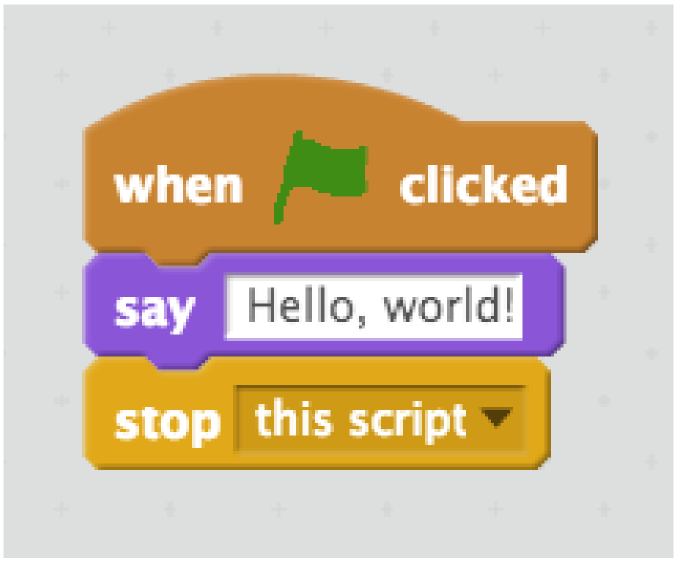


Figure 1: An example of Scratch and its block-based coding

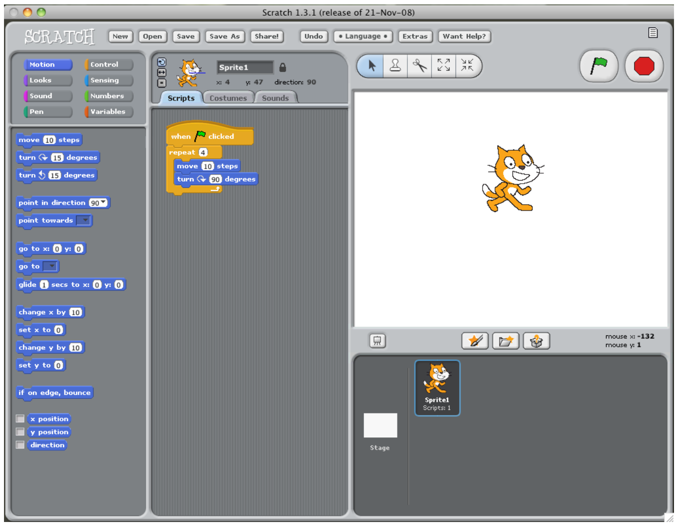


Figure 2: The interface of Scratch

er can be rather challenging. However, the Web Platform (which will hence forth be referred to as WP) was built for this and implemented in a workshop that uses a system very similar to story-boarding. In the WP, the user takes snapshots and gives them captions to try and accurately describe what is going on. The workshop used a Visual Programming Environment, Scratch, to guide a group of students aged, thirteen to eighteen, on their journey in constructing and coding a "smart cup" to actively document what they were doing [6]. The smart cup is a normal cup with hardware attachments that is programmed to display the temperature of the liquid within the cup. The reasoning behind making the smart cup was to have the students make several gadgets using Scratch and see how well they could use their knowledge learned and apply it to the hardware of the smart cup. They met once each week, thus adequate documentation was needed to remember where they left off, what were their intentions, and where they need to go. In theory, this seemed like a good approach but in practice did not work out due to several reasons. The first and biggest reason as to why the WP did not take was due to how Scratch works. Scratch is a visual based code, which leaves documenting afterwards unnecessary and thus left the WP unused. Also, the students felt that the extra step was highly unneeded and was more of an after-thought than anything. Given this information and results, the students did learn how to document code, though not in the way intended, and why documenting is a good idea as it reminds one where they left off from.

3.2 Testing

Games are a superb framework in which to work out how something works or how one is supposed go about building it. Constructionist Gaming really emphasizes this, and it creates a fun environment in which to work. Constructionist gaming is the idea of making a game (from nothing or based on an already established game), and implementing it within a group of peers. This sounds very simple and straight forward but it involves a lot of logic, coding, teamwork, and *testing*. In doing constructionist gaming, a group of students (high school freshman, four girls, thirteen boys, ages thirteen to fifteen) use technology mixed with board games to make augmented board games. These students had to test their board game in at least two ways [5]. The first was play-testing, where they had to work out the specifics of their game. The hardware side of testing would be similar to this. They made a board game and rules that go along with it, then proceeded to play it repeatedly in order to make sure that: 1) The rules made sense in the context of the game. 2) The board pieces and future features (not yet functional) would actually work with what they already had, and 3) that it was *actually* fun to play. The second bit of testing they did was on the code they made for the electric parts of the board game they created, the augmented side of their workshop project. They strived to make sure the code performed as intended and that it would not break due to some unforeseen problem.

To prepare them for this, the workshop gave them several "debug'ems" where they were given either incomplete or broken code (in Scratch) and asked to solve or fix them. Through these debug'ems, they learned to recognize simple errors and were able to fix the little errors that came up during their project with minimal help from the instructors. [5]. Through this simple workshop they learned a lot about testing and how important it was in making sure that a given product they produce is fully functional and unlikely to break due to user interaction.

4. TIME SPENT ON LEARNING

This section is on several studies, their goals, and the results of the studies.

4.1 Several Hours a Day

This section is on a study that focuses on students in a summer camp as they learn various CS concepts such as conditionals and synchronizing sound with speech bubbles in Scratch.

4.1.1 Setup

Summer is an oppurtune time to gather data from children for research purposes as this is the time they have the most free time avaiable. Summer camps are intended to teach children through disciplined, organized activities so that the lessons learned are retained for those who may wish to pursue further instruction in these areas. A group of researchers took on a group of middle-school students to teach basic lessons about computer science without the use of exams or similar artifacts. In this summer program, they will be using Scratch to code various projects that are given out in the two weeks the camp takes place. The camp had 5 lessons, with each lesson taking about 2 days to complete. This subsection, and the following subsections unless otherwise specified, take their info from study by [3].

4.1.2 Goals

The outreach program had three goals in total. The first goal was to attract middle school students from underrepresented groups with non-CS themed backgrounds. Second, 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. And finally, to assess the learning of CS content that took place during the camp. It consisted of 35 individuals, and 10 of these campers were repeats (they attended the camp before). The studies assessment focused on the individuals that were taking this camp for the first time. The curriculum were subject to two constraints. (1) The interdisciplinary themes influenced the computer science content of the lessons because of the culminating project- a digital story-telling project. (2) Assessment influenced the structure of the lessons. Each lesson was in two parts, the first was not assessed and it was the teaching of one or more CS concepts with scaffolding and support. The second part of the lesson was assessed and it consisted of a small project where the participants applied their new knowledge to a similar, but new, problem.

4.1.3 Learning Process

Lesson 1 was Scratch basics and it's mini project was a Name Poem. The basics were the "stage" (where the code was placed), sprites, how blocks are combined into scripts, and 2 CS concepts, **Event-driven programming** and **Initial state**. Event-driven programming in Scratch means that when blocks that say "When Green Flag clicked" are clicked an event will occur depending on what that script contains. Sprites have certain attributes, such as position

Table 1: The Side-Scroller Quiz

Question	Answer	Results
If you want the cat rather than a panda, what would you change?	sprite	100%
If you want the cat to go into a cave rather than a temple, what do you change?	scene or background	82%
How could 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%

and size, that if changed need to be initialized when the "green flag" is clicked to restart after completion. The Name Poem project had the campers spell out their assigned animal, and when each letter was clicked an adjective starting with that letter would appear on screen. [3] For example, the character "Bear" would become **B**ig, **E**lusive, **A**daptable, **R**ough.

Lesson 2 was about multi-sprite synchronization through conversation between two sprites. In this lesson, 2 more CS concepts are introduced and they are **Broadcast/Receive** and **Say/Sound Synchronization**. Broadcast/receive is used to control timing in Scratch. A collection of scripts can be associated with each sprite and runs as a separate parallel thread. This can be thought of as "message-passing" in other languages. Say/Sound Synchronization is used when a speech bubble is displayed at the same time as an audio file plays. Unintentionally, this also taught another CS concept, thinking sequentially. The blocks of speech and sound must be done in a sequence that is logical and lines up with the audio, otherwise the two sprites will make no sense. The project done here was having two sprite animals have a conversation with one another.

Lesson 3 was about scene changes and the project for this lesson was Mayan Conversation. The scene changes were used to expand on message passing and sequential order by introducing **Visibility** (the ability to hide or show an object) and backgrounds. The Mayan Conversation reinforces message passing, sequential order, visibility to make a conversation in the Mayan Language. The campers are given the first two glyph with their respective recordings and they are to sequence them like a scene.

Lesson 4 was about Complex Animation and the project was another Name Poem but with motion. **Complex Animation** teaches the campers how to create more realistic motion with sprites and it requires several of the above concepts to do. The Lesson 4 project has the campers add motion to the Name Poem they created in the first lesson with their assigned animal.

The Culminating Project was introduced after the fourth lesson. The project was about animating a Mesoamerican Animal Myth, this project must also include their assigned animal, three scenes, and at least two characters.

Lesson 5 was an Interactive Conversation with an optional project, a side-scrolling game. This lesson contained warm-ups introducing **Input**, **Variables**, and **Conditionals** (if/then/else).

4.1.4 Assessment

Campers were given a quiz at the end of the program to determine what they have learned and retained. The quiz, shown in Table 1, displays the questions asked and what percent of the campers answered correctly. As one can see, all were adequately answered by the majority except for if conditionals but seeing as it was the last thing taught with little time left, this is not unexpected. In spite of the constraints, this camp managed to teach middle school children basic computer science concepts in just two weeks using Scratch. The researchers also realize that their sample size is small, and with a Summer program, they will only reach a small part of the population. However, it is highly likely that the integration of computer science will have a broader impact. They expect that by pairing their lessons with a curriculum that teaches variables, loops, and conditionals, students can use computer science thinking and apply it to other subjects required in schools.

4.2 Weekly

This section focuses on a study done for one hour each week after school.

4.2.1 Setup and Goals

A second approach used to teach kids about Computer Science is after school workshops. In Kafai and Vasudevan's afterschool workshop they taught once a week in a small program that ran for eight weeks after school. [5] The idea was to introduce coding in a Constructionist manner in order to teach students Computer Science concepts, such as testing mentioned earlier. The researchers were aiming to highlight intersections between learning programming and creating games across digital and tangible modalities. [5] They focused their analysis on the high school students' projects, interactions, and reflections on how they conceptualized the integration of screen and board game elements, realized computational concepts and practices, and reflected on their board game design experience connecting *coding* and crafting.

4.2.2 The Workshop

The researchers, Y. Kafai and V. Vasudevan, gradually introduced gaming, coding and then crafting to the students. In Session 1, the students had the workshop explained to them in detail and they played board games. After the games were done, they reflected on the design of the games. Sessions 2 and 3 consisted of the students designing their board games and testing them for bugs or flaws in logic. Sessions 4, 5, and 6 were about Scratch. They were taught the basics of Scratch programming. To ease their comfort with Scratch some simple programs were created, that were either incomplete or wrong, and the students were tasked with fixing or completing them. These programs or "debug'ems" were directly related to any coding that might be necessary for their augmented game boards. Also, the students tested the digital components of their board games, such as digital dice, to get a feel for how the technology would affect game-play. Sessions 7 and 8 focused on completing their board game designs, Scratch code, and integrating other digital components. In the final session, they played their created games and got feedback from adults that worked in various creative fields. [5]

4.2.3 The Code Used

The researchers taught the students concepts like movement, appearance, event-based functionality, and some basics of parallelism. The students broke up into pairs, and solved "debug'ems". These programs are simple, and almost complete but intentionally have flaws. One team opted to code in digital dice, musical introduction, and digital locations (e.g. skip a turn, go back to start) to move in their game. [5] They also included player selection, and abilities as part of their code. All this was done in Scratch, with the majority of it being done from scratch.

4.2.4 Results

In this setting there were mixed results. The students tended to gravitate towards what they thought they were good at. For instance, one of the students in team Cairo gravitated toward Scratch because he felt comfortable in a logical environment and was not at at all comfortable in an artistic setting. A different student, Jordan, gravitated towards Scratch at first because he knew nothing about it but then because he found it too challenging shied away from it but not before he completed a die rolling program. On the other end of the spectrum, one student did not really participate at all due to her lack of comfort with both programming and artistic skills. [5] What we can really draw from this study is that more time than once a week is needed to teach basic CS concepts and to attract more students to the course. Increasing comfort levels with coding makes it easier to bridge the gap of confusion concerning CS concepts and still providing a learning environment in which students can easily learn and apply these concepts.

4.3 Applying certain techniques and results

This section focuses on several studies as well. It also gives background on two techniques and applies these to both secondary and post secondary school.

4.3.1 Background on Prominent Techniques Used

A different approach is in a school setting and also uses Scratch. Two researchers focused on two different techniques for transferring knowledge from one context to another. They are termed "bridging" and "hugging". [4] Bridging is learning a concept in one area through real life examples within a single language. When presenting a new topic in a language, a *Bridging* conversation would include how the concept is applied in other contexts like in a real-world scenario or a different coding language. Hugging on the other hand, is more aligned with the second interaction. For instance, a concept is taught in one language and then the same concept is taught in a different language but using very similar problems to those used in the first language.

4.3.2 College Level

In a college level class, one researcher applied bridging and hugging to her mixed Alice/Java course. [2] Alice is a VBE similar to Scratch. Compared with her Java only class, students achieved on average a twenty percent increase in contrast to her Java-only course, with similar gains in other subjects of the course. Also, another result of her class that became apparent was that the block-based language, Scratch, lacked sufficient support for simple problems concerning inheritance and parallelism that are supported in Java. She also had problems with teaching conditionals to her students in both courses. Conditionals seem to be a recurring theme in being difficult to understand no matter the background of the student.

4.3.3 Middle School Study 1

In a middle school setting, one researcher taught a group of students with Scratch then later had some of those same students take a High School CS course. [1] Those that had previously taken the Scratch course performed much better than those who had not. These students scored higher than other students in the same class when bounded loops were taught. Though one subject that did not seem to matter if one had taken a previous course or not was variables and conditionals. The two groups both seemed to be at the same level of understanding when it came to this subject. Again, the concept of conditionals and how they are applied was a difficult concept to understand. Overall, the teachers found that they could cover material at a significantly higher rate because of their previous knowledge, and concepts learned with Scratch were easier to teach in a different language.

4.3.4 Middle School Study 2

The setup for this study involved ten 4th to 6th grade classroooms at five schools across California. [4] In two of these schools (Schools B and E) they collect only snapshots of projects and do not observe them. In the other three schools (Schools A, C, D), they observed instruction of the students and they also interviewed students. The researchers in observing the three classrooms wrote field notes following each class session. In all of the schools, they collected the final projects and snapshots of the projects when each project was run (green flag clicked) with 4 or more changes. Their goal was to assess students in their ability to initialize objects in Scratch through identifying features in their final projects in hopes of correcting their teaching methods for further classes. They specifically wanted a cat sprite to be initialized in an Activity called Animal Race. The correct script initializes two of the cats attributes, size and position.

The researchers met and compared notes. They discussed what worked and what did not, changes in lessons, and focused on points where students made errors from the perspective of experts in computer science. The sum of this meet-up was that differences in Text Based Language (TBL) like Java and Visual Block-Based Language (VBBL) were made much more apparent because the experts had been using TBLs their entire life and made analyzing the field notes somewhat difficult. [4] Even with this minor setback, they identified pieces of knowledge derived from the lens of traditonal TBLs but expressed in a way applicable to Scratch. [4]

The results of the study showed when initialization occured in a given students project in the final snapshot. As in Figure 4 we see that School C was the best in correctly initializing the cat before the sprint and that School B was the worst as a little over 50% of the class did not even initialize the cat. School C did so much better than the other schools because it specifically spent three sessions on Ani-

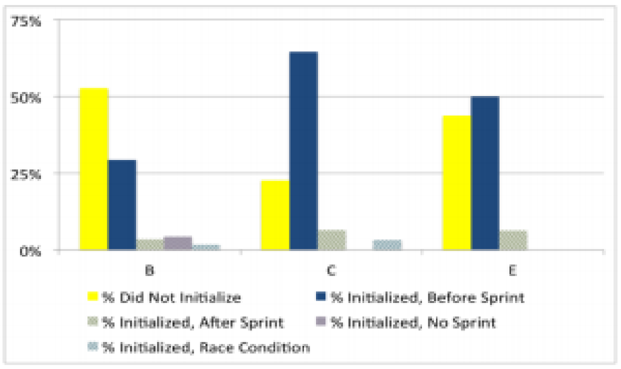


Figure 4: Whether or not the cat was initialized and the timing of the initialization

mal Sprint with very targeted help. The other schools did not spend nearly as much time and it shows with roughly half in both schools not initializing the cat. Also, students in schools B and C had some trouble with a concept call a race condition. A race condition occurs when two or more threads try to update an object at the same time. Surprisingly, School E had either no trouble with this error as they sequenced their project in a logical manner. In Scratch, a race condition would occur when one would have two scripts for one sprite both run at the same time.

5. CONCLUSIONS

Teaching with Scratch first and as early as possible makes the transition to TBLs much easier. Ranking each strategy, I would say that most effective way to teach Computer Science using Scratch would be in a class-like setting for a few hours each day as early in life as possible. The earlier the teaching it seems the more likely it is to be learned at a faster paced, except for conditionals. Conditionals really seemed to bog everyone down even into the college level and it was not until my second year into the CS discipline that I fully understood the logic behind them. With this in mind, I conclude

Acknowledgments

I would like to acknowledge Nic McPhee, Peter Dolan, KK Lamberty, and Elena Machkasova for their help, support, and feedback. I would also like to acknowledge Stephen Adams for his feedback. Also, big thanks to my parents for putting up with me for all these years and supporting my decisions.

6. REFERENCES

[1] M. Armoni, O. Meerbaum-Salant, and M. Ben-Ari. From scratch to &ldquo;real&rdquo; programming. *Trans. Comput. Educ.*, 14(4):25:1–25:15, Feb. 2015.

[2] W. Dann, D. Cosgrove, D. Slater, D. Culyba, and S. Cooper. Mediated transfer: Alice 3 to Java. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, SIGCSE '12, pages 141–146, New York, NY, USA, 2012. ACM.

[3] D. Franklin, P. Conrad, B. Boe, K. Nilsen, C. Hill, M. Len, G. Dreschler, G. Aldana, P. Almeida-Tanaka, B. Kiefer, C. Laird, F. Lopez, C. Pham, J. Suarez, and R. Waite. Assessment of computer science learning in a scratch-based outreach program. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 371–376, New York, NY, USA, 2013. ACM.

[4] D. Franklin, C. Hill, H. A. Dwyer, A. K. Hansen, A. Iveland, and D. B. Harlow. Initialization in scratch: Seeking knowledge transfer. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, pages 217–222, New York, NY, USA, 2016. ACM.

[5] Y. Kafai and V. Vasudevan. Hi-lo tech games: Crafting, coding and collaboration of augmented board games by high school youth. In *Proceedings of the 14th International Conference on Interaction Design and Children*, IDC '15, pages 130–139, New York, NY, USA, 2015. ACM.

[6] E.-S. Katterfeldt and H. Schelhowe. Considering visual programming environments for documenting physical computing artifacts. In *Proceedings of the 2014 Conference on Interaction Design and Children*, IDC '14, pages 241–244, New York, NY, USA, 2014. ACM.

[7] Wikipedia. Reflection (computer programming) — wikipedia, the free encyclopedia, 2016. [Online; accessed 23-March-2016].

[8] Wikipedia. Scratch (programming language) — wikipedia, the free encyclopedia, 2016. [Online; accessed 23-March-2016].

[9] Wikipedia. Squeak — wikipedia, the free encyclopedia, 2016. [Online; accessed 23-March-2016].

[10] Wikipedia. Storyboard — wikipedia, the free encyclopedia, 2016. [Online; accessed 24-March-2016].