SPAE: A Scratch Project Analysis tool for Educators

A Thesis
by
Joseph O'Neill

Submitted to the Graduate School
at Appalachian State University
in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE

December 2018
Department of Computer Science

SPAE: A Scratch Project Analysis tool for Educators

A Thesis
By
Joseph O'Neill
December 2018

APPROVED BY:

_____
Dr. James B. Fenwick Jr.
Chairperson, Thesis Committee

_____
Dr. Raghuveer Mohan
Member, Thesis Committee

_____
Dr. Cindy Norris
Member, Thesis Committee

_____
Dr. Rahman Tashakkori
Chairperson, Department of Computer Science

_____
Michael J. McKenzie, PhD
Dean, Cratis D. Williams School of Graduate Studies

Abstract

SPAE: A Scratch Project Analysis tool for Educators

Joseph O'Neill
B.S., University of Tennessee
M.S., Appalachian State University

Chairperson: Dr. James B. Fenwick Jr.

Middle school and high school educators are always seeking new ways to offer differentiation and personalization of learning to their students. Computer programming can provide a differentiation technique as well as strengthen mathematical and problem solving skills. However, text-based computer programming languages are difficult for younger students to learn. Scratch is a visual, block-based programming environment that targets these younger novice learners. Scratch has been very successful at breaking down this learning barrier, evidenced by the 35 million projects created by 33 million unique users in twelve years.

Educators that wish to use Scratch in the classroom now have a new problem in understanding how to evaluate and assess student projects. This thesis describes the Scratch Project Analysis for Educators tool (SPAE). SPAE is an easy to use web application that provides a summary of Scratch project characteristics that teachers can use in evaluating student work. SPAE is implemented on a variety of hardware and software platforms to ensure accessibility to any teacher. The reliability of SPAE was demonstrated through the analysis of nearly one million Scratch projects.

Acknowledgments

First, I wish to thank my advisor, Dr. Jay Fenwick. Through his constant enthusiasm in his daily teaching and involvement in the COSMIC afterschool program, he has shown me how far a little bit of kindness and a lot of passion for your work really does translate into results. Without his compassion, I very likely wouldn't have survived my Master's program, let alone have the drive to write this thesis. I would also like to thank my thesis committee, Dr. Cindy Norris and Dr. Raghuveer Mohan, as well as Dr. Mitch Parry and Dr. Rahman Tashakkori. Only through their constant involvement in my education and availability outside the classroom was I able to work my way through this program and thesis.

Next, I wish to thank Mary Michael Forrester, who did wonders in helping me stay sane and on task through this program. Whether it was proofreading a paper, talking over some homework problems, or simply getting me some coffee when I desperately needed it, she was always there for me whenever I needed someone most. Without her, the motivation to earn this degree would have left me on day one.

Finally, I wish to thank my family and friends for their constant support. Through my doubts and the rough times, they continued to stand by and support me in everything I did. Whether it was a listening ear, a ride to the store, help with some debugging, or simply someone that wasn't a machine to talk to after a long day, I always managed to find someone who cared for me when I needed them the most.

Dedication

To my family, teachers, and colleagues in education, who have dedicated their lives to the education of their students and children. Without their devotion to their craft, this thesis would not have been possible.

Table of Contents

List of Tables

List of Figures

Chapter 1 - Introduction

Educators have long known that techniques for learning which work well with one student do not necessarily work well with another student [69]. Differentiation is the method of providing varying means of learning appropriate for different students. This differentiation, or personalization, has driven educators to explore new methods of teaching their students the skills needed for success. Along with the rise of differentiation within the classroom, many teachers have begun to work on teaching strategies utilizing the mathematical and problem-solving skills of programming. Over the past couple of decades, more and more schools are including computer programming courses in their schedules [46]. Originally developed for students in high school, teachers in middle and elementary schools now realize the mathematical and problem solving skills found in programming can also be beneficial to students in their own classrooms. This interest inspired a need for programming languages that are easier to understand and use than traditional text-based languages.

One language that has gained popularity in recent years is MIT's Scratch programming environment. Scratch is a free visual programming language using a drag-and-drop interface. Directed at young adults, Scratch concentrates on event-driven actions where students, parents, and teachers can build and implement various projects, games, and stories. These projects can then be shared with their peers, allowing them to see the code behind these projects and 'remix,' or modify, them to their own liking. Scratch is a web-based application, which simplifies its use in schools because there are no installation issues. Through Scratch and other simple programming languages, users are able to learn the basic skills (if-else statements, loops, conditionals, process development, etc.) needed to succeed in computer science. And, equally as important, is the use of these languages to help teachers provide differentiation within their

classrooms. For example, instead of a traditional book report a student could submit a dynamic and interactive presentation. As another example, the common diorama project can be replaced with a Scratch project that uses actual images of desired landscapes, flora, and fauna.

The use of Scratch in classrooms has driven the need for tools that can be used to evaluate Scratch projects. Teachers can have a difficult time determining the difference in complexity between a Scratch project with many simple blocks versus a project with fewer, but more challenging, blocks. The process of evaluating a classroom full of projects in detail is relatively time consuming and repetitive, even for experienced educators and programmers. Combined with the continued increase in classroom sizes nationwide [55], similar project-based assignments have become less appealing to educators who need to cover a wide range of topics and material over the school year. A tool assisting with the assessment of student work would be invaluable in supporting educators use of Scratch as a differentiation strategy.

This thesis describes the development of SPAE[1], the Scratch Program Analysis for Educators tool. SPAE allows students and educators to easily analyze their work through a simple web-accessible interface. SPAE adapts and combines several third-party modules to extract the internal information of a specific Scratch project and provides the user with a summary of the project. Chapter 2 covers the history of Computer Science in the classroom, as well as the effect analysis tools have on the teacher-student dynamic. Chapter 3 discusses previous examples of Scratch project analysis and related work. In particular, the SCATT third-party module is discussed because it is used internally by SPAE. Chapter 4 describes the software components of SPAE developed for this thesis as well as the stages of their development. In addition this chapter covers the hardware used in this project and the testing

[1] Spae, old Scottish for “to prophesy; foretell; predict.” The SPAE application is used to inspect and interpret student work.; in other words, to help teachers “see” student projects in a new light.

methodology. Chapter 5 presents the results of testing in terms of SPAE performance, as well some insight into the usage of the popular CS First curriculum. Finally, Chapter 6 summarizes and presents areas of future work. All code, a copy of this thesis, and the thesis defense presentation can be found at the following URL: https://github.com/oneilljo/SPAE.

Chapter 2 - Background

2.1 History of Computer Programming Education

Computer programming within the classroom has dated back to the early days of personal computers. While universities in the United States often had their own systems in place to teach concepts of computer science as early as the mid 1960s [70, 9], primary and secondary schools were slower to adopt computing education because of the high costs, hardware shortages, and lack of training [24]. Even when primary and secondary schools had computers, these devices were primarily used to aid in the already established administrative and teaching methods of the time. It wasn't until the late 1980s, when computers and software became more readily available, that teaching programming and computer literacy began to emerge within the classroom.

With the development of more user-friendly software and the explosion of the Internet, educators began using software to aid in their student's education. In the late 1990s and early 2000s, Microsoft Office programs, such as Word, PowerPoint, and Excel, were widely used by students and educators to promote computer learning and aid in the mastery of educational concepts [6, 44]. These programs allowed students to write reports for classes, explain learned concepts to educators, and teach basic computer literacy to anyone with access to the programs. Other software was also developed for more specific educational topics, such as mathematics and science. The Minnesota Educational Computing Consortium (MECC) [34] was one such organization that concentrated on developing games for students to achieve academic success [39]. MECC developed games like *Word Munchers* [33] that concentrated on teaching students basic grammar skills, including identifying verbs, nouns, and other parts of speech. The Learning Company [68], developed similar educational games such as *Super Solvers: Treasure MathStorm*

[10], which is a mathematical based game that taught students concepts like addition, multiplication, understanding clock time, and simple money concepts.

2.2 Effective use of Technology in the classroom

The two previously cited games were a few of many pioneers in educational software that allowed students to master subject concepts educators were teaching. Each of these games allowed educators to see how well students were understanding concepts through a simple scoring system. These methods also allowed students to work on their computer literacy skills, as well as provide a fun format to involve them in the subject matter. While these earlier games only provided a simple feedback of success/failure to educators, educational software has advanced to give much more detailed feedback to teachers.

While these advancements can provide educators with more knowledge of student progress, the software is only as effective as its ability to keep student attention and interest. "A Pedagogy must, however, be subordinate to story… the entertainment component comes first. Once it's worked out, the pedagogy follows." [71] An effective educator should tailor their education plan around the student's interests. According to Sharyn O'Neill, "Effective teachers personalize the learning for their students. They understand that students develop at different rates and that in every classroom there will be a range of student abilities." [22]

Another obstacle that educators must hurdle with computer education is cheating and plagiarism. Computer science students are caught cheating more often than in other disciplines. [40]. There are many suggestions why this may be the case, such as poor understanding of how cultural collaboration works or that computer science solutions are more prevalent on the Internet for copy and paste. However, Bidgood and Merrill argue that another explanation is that computer science teachers have more tools to detect cheating in their courses [5]. Plagiarism

detection software has been around for a few decades and has been effective at catching and discouraging the unethical actions from occurring in the first place. Regardless, in 2015, nearly 100 students in a University of Berkeley class of 700 violated course policy by copying code [4]. Moreover, this is not just a problem with college students. In 2010, 59% of high school students admitted cheating on a test during a single school year, and 34% self-reported cheating at least two times [43].

Educators and researchers have been reporting more student involvement and success when they have prior interest with technology [27]. However, combining all of the aspects of developing an effective curriculum for each individual student in the classroom, teachers are often overwhelmed with the workload required to develop such a plan. According to a study from 2012, teachers work an average of 53 hours per week [67]. Utilizing technology for effective teaching becomes even more time consuming when educators must analyze individual projects and reports for student mastery. This juggling of tasks reveals the need for easy-to-learn interactive activities with teacher oriented automated analysis tools in the classroom.

2.3 The Scratch Programming Tool

Scratch [57] is an event-driven, block-based programming language developed to help students understand basic programming concepts and build their own programs. Students use the Scratch tool to tell stories, animate backgrounds, manipulate variables, and create games. The following sections detail various aspects, functions, and issues with the Scratch programming language.

2.3.1 Overview of Scratch

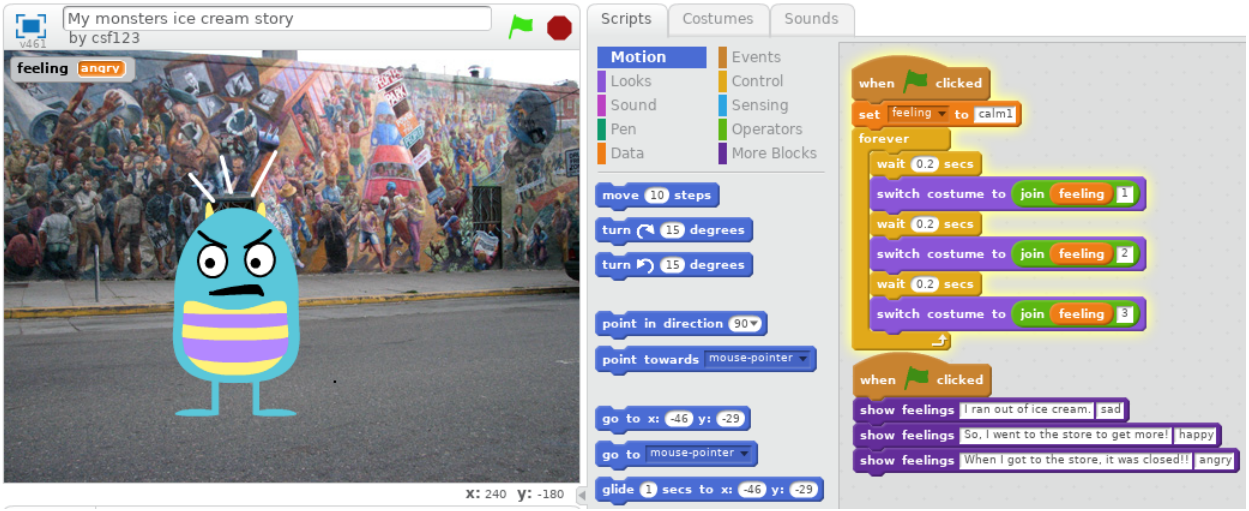

Figure 2.1: An example Scratch project interface window

Figure 2.1 shows a sample script that uses *Events*, *Control*, *Operators*, *Data*, and *Looks* blocks to animate a character. The left side of Figure 2.1 shows the "project stage." Much like a theatrical performance, the project stage can have backdrops (Figure 2.1 shows a street with a mural as an example) and players or actors (for example, the blue alien here) called "sprites." The action of a program occurs on the project stage and users interact with their project using a mouse, keyboard, and even a webcam. The middle section, known as the blocks palette, allows the user to select blocks that will perform an action or operation for the project. There are ten categories of blocks in the blocks palette:

- Motion: Blocks that provide instructions to sprites for movement around the stage.
- Events: Blocks that control events that occur in projects such as beginning scripts and detecting keyboard and mouse events.
- Looks: Blocks that change a sprite's appearance; often used to animate a character.
- Control: These blocks manage the order of blocks within a script and include conditional statements and loops.

- Sound: Blocks that produce sounds and provide MIDI functions such as volume and tempo.
- Sensing: Blocks used to detect various environmental characteristics of a project, including mouse location, collision detection, and sprite location.
- Pen: These blocks provide the old Logo [31] functionality of allowing a sprite to control a pen that would draw on the stage.
- Operators: Perform arithmetic, logical, and relational functions and string handling for variables, etc.
- Data: Blocks for creating and managing lists and individual data variables.
- More Blocks: Custom blocks and unique blocks used with supported third-party hardware vendors such as Legos [28], PicoBoard [47], Finch robots [14], and Raspberry Pi [52, 54] to name a few.

The right section of Figure 2.1 is the scripts space, where individual blocks are combined together to create action. The user drags a block from the palette area into the scripts space and "snaps" it into place. The blocks also have shapes that help prevent syntactical errors because only the right shape block can snap into place. For example, in Figure 2.1 Scratch will not allow a blue, jigsaw-shaped motion block to replace a green, round operator block. In the figure, preset images give the appearance of the sprite reacting to a lack of ice cream.

2.3.2 Scratch History and Usage

Scratch, originally released in 2002 by the Lifelong Kindergarten Group at MIT, is designed for students age 8 to 16. Scratch 2.0 was released to the public in 2007 in its current, web accessible format. Figure 2.2 shows the dramatic growth of the use of Scratch since 2008. In addition, according to [59], during the month of September 2018 (historically the slowest month

of the year), the Scratch website recorded 195,696,610 pageviews from 17,128,700 unique visitors. During this same period, the Scratch website also recorded 839,370 new projects and 942,409 new users [59].

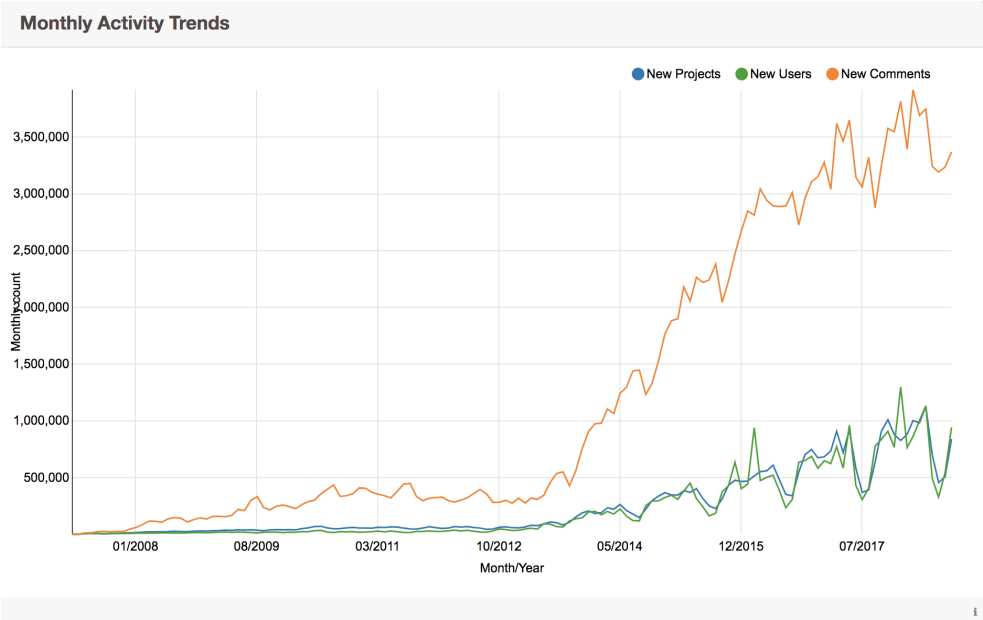

Figure 2.2: Growth seen by Scratch since its official release in 2007

Early next year, Scratch will be releasing the next version of their tool, Scratch 3.0, which will better support usage from mobile devices. With the newly introduced ability to use mobile devices, Scratch expects to see a large uptick in the program's use over the coming years [58].

2.3.3 Scratch Collaborations

The Scratch project has also developed an active education community that is focused on sharing curriculum ideas and developing student projects. Scratch-ED [61], a website maintained by the Harvard Graduate School of Education, has thousands of projects, stories of success, and classroom resources geared towards primary and secondary educators interested in using the Scratch tool to teach students a variety of concepts and skills.

Scratch also takes part in the yearly Computer Science Education Week [11], most notably through the Hour of Code program [23]. The goal of the Hour of Code program is "to

demystify code" [23], and provides individuals aged 8 to 104 with a basic understanding of computer science. Hour of Code, started in 2013 by Code.org [12], has helped teach more than 600,000,000 individuals in over 180 countries [23] about Computer Science.

2.3.4 Scratch File Formats

Since Scratch has been around for over 15 years, the application has gone through quite a few changes, including the handling and storage of Scratch projects. Between 2003 and now, Scratch projects have been stored in three separate file formats. The filename was project specific and the "extension" of the file indicates the file format version. A file extension of ".scratch" was the original Scratch project format during beta testing. The file extension of ".sb" was used for Scratch 1.x projects; and this will be referred to henceforth as an SB file. Lastly, a file extension of ".sb2" indicates the file format for current Scratch 2.x projects [64]. Henceforth, these will be referred to as SB2 files. The ".scratch" files have become widely deprecated [56].

The SB2 format is essentially a ZIP file containing all the media files (e.g., audio, images, etc.) used in the Scratch project. In addition, the ZIP file contains a JSON file that encodes project information including sprites, scripts, etc. Figure 2.3 shows a portion of a Scratch project JSON file. Near the top, a sprite object named "Goal" can be seen. Continuing to examine the JSON file shows the "scripts" attribute. The script shown in the figure contains block commands that describe the behavior that when the object named "Ball" touches this "Goal" object, a caption saying "You win!" should pop up in the project space. Continuing to look at the JSON file shows that this Goal sprite can make a "meow" sound, has one "costume" defined, and its original position on the stage was the Cartesian (x,y) coordinate of (202,-179).

```
        {
                "objName": "Goal",
                "scripts": [[70,
                                33,
                                [["whenGreenFlag"],
                                        ["doForever", [["doIf", ["touching:", "Ball"], [["say:", "You win!"]]]]]]],
                "sounds": [{
                                "soundName": "meow",
                                "soundID": 0,
                                "md5": "83c36d806dc92327b9e7049a565c6bff.wav",
                                "sampleCount": 18688,
                                "rate": 22050,
                                "format": ""
                        }],
                "costumes": [{
                                "costumeName": "costume2",
                                "baseLayerID": 2,
                                "baseLayerMD5": "1cb230768f17fe4cc75afd2dcb32a928.svg",
                                "bitmapResolution": 1,
                                "rotationCenterX": 16,
                                "rotationCenterY": 21
                        }],
                "currentCostumeIndex": 0,
                "scratchX": 202,
                "scratchY": -179,
                "scale": 1,
                "direction": 90,
                "rotationStyle": "normal",
                "isDraggable": false,
                "indexInLibrary": 2,
                "visible": true,
                "spriteInfo": {
                }
        }],
"info": {
        "videoOn": false,
        "swfVersion": "v461",
        "scriptCount": 7,
        "userAgent": "Mozilla\/5.0 (Macintosh; Intel Mac OS X 10_13_6) AppleWebKit\/537.36 (KHTML, like Gecko) Chrome
        "flashVersion": "MAC 31,0,0,148",
        "spriteCount": 2,
        "projectID": "10128431",
        "hasCloudData": false
}
```


Figure 2.3: Portion of a JSON from a SB2 file

Another helpful aspect of these JSON files are the “info” lines, which describe certain elements of the project. Here, information such as the number of scripts, number of sprites, and the project ID are provided. For the SPAE tools, the project ID is particularly useful, as every Scratch project created on the Scratch website has one of these unique identifying values attributed to it. This value can also be seen on each individual Scratch project webpage URL. In the case of Figure 2.3, the URL would be https://scratch.mit.edu/projects/10128431/, where the last section, 10128431, is the unique project ID.

Originally, Scratch saved project files in a folder on the local machine, but with Scratch 2.0 moving to a web application, all project work is stored on the Scratch website. The programmer has the option to download the SB2 file onto their local machine. While the Scratch

project is stored on the Scratch website, it is not visible for other Scratch users to see unless it is explicitly "shared" by the Scratch programmer. However, even an unshared project can be accessed via its project ID if the project ID can be somehow learned. Finding these project IDs was an important part of testing SPAE on a large number of projects as described in Chapter 4.

2.3.5 Scratch Project Remixing

"Remixing" is the term Scratch uses to describe when "a Scratcher makes a copy of someone else's project and modifies it to add their own ideas" [56]. Remixing is a keystone to the Scratch tool, allowing users to learn how any shared project works and modify a project to their own personal liking by remixing it and modifying the project source code. The developers of the Scratch tool believe "...that viewing and remixing interesting projects is a great way to learn to program, and leads to cool new ideas" [56]. Because of this ideology, all projects that are shared on the Scratch website are able to be remixed.

2.4 Google CS First

Google's CS First curriculum [21] uses the Scratch tool to develop computer science skills for students aged 9-14. Organized as "clubs," this program allows students to pick a theme that interests them and guides students through a step-by-step curriculum based on their selection. CS First currently has nine themed curricula: Storytelling, Art, Game Design, Fashion, Music & Sound, Friends, Social Media, Sports, and Animation. Each of these programs have eight self-paced lessons that allow students to follow along and learn new programming concepts through the Scratch program. Each lesson is tailored to last between 60 - 90 minutes, perfect for a class-long activity or a before/after-school program.

Each themed curriculum also provides mechanisms for the "club creator" to keep track of student progress: Lesson plans, solution sheets, passports to keep track of usernames and

passwords, badges show completed tasks, informational flyers promote interest, and even promotional videos and student rosters. Each of the eight lessons start with an introductory video to show what project they will be working on that day, a link to a Scratch starter project, and a simple explanation on how to complete each step. In Figure 2.4, the Scratch interface is nested inside the curriculum window. The top of the window identifies the "Game Design" curriculum and the current lesson is lesson 4 on creating a winning condition. On the right side are instructions and links to starter projects.

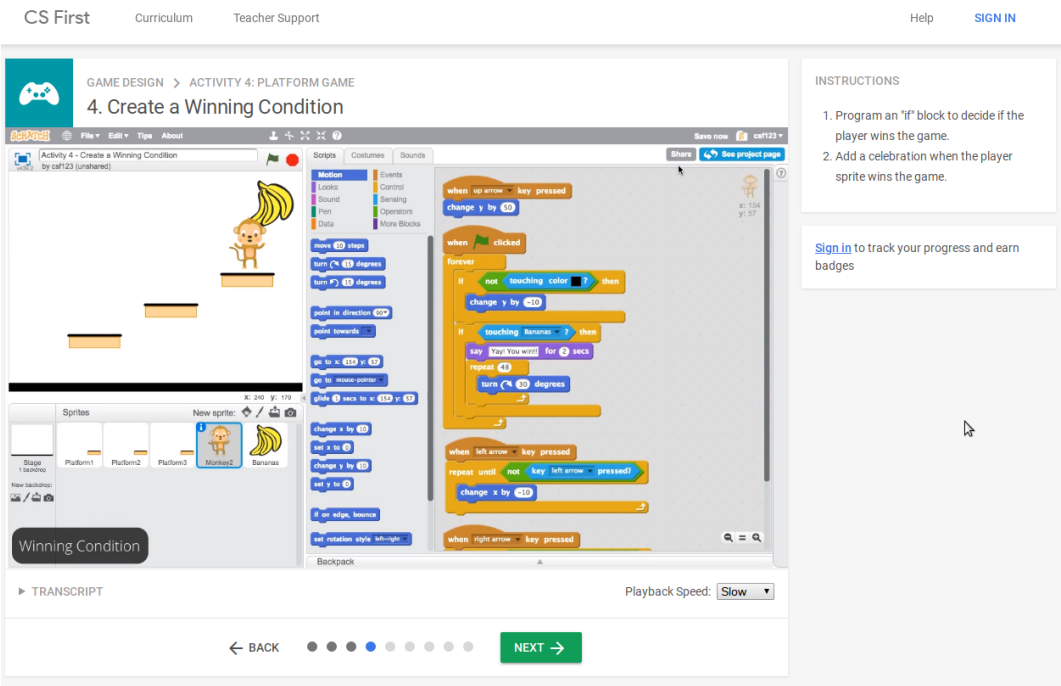

Figure 2.4: CS First Game Design walkthrough page

One example of a successful program designed around the CS First curriculum is Appalachian State's "COmputer Science for Middle schools In Caldwell county" (COSMIC) program [8]. COSMIC was designed to improve competencies in programming skills among middle school aged children of all backgrounds and upbringings. The program hosted an after-school program for three years at four middle schools in Caldwell County, North Carolina, where each club typically contained about 20 students. Students chose their own CS First curriculum

and worked in a self-paced way through each lesson. Professors, graduate students, and teachers from area schools were available to help guide students as needed. The program was successful at introducing computer science concepts to middle school students and was highly valued by the schools and teachers.

While the CS First program is effective at teaching students the major concepts of programming, there are a few areas that do fall flat. For the curriculum as a whole, there is a fair amount of video watching required. In addition, without the right facilitator and interest in the theme, students can lose interest in the projects [32]. Even with the detail oriented descriptions, some of the tasks can be challenging to students who are being introduced to programming. Yet, in spite of these minor drawbacks, the amount of detail, provided resources, and extra activities the program provides make CS First a highly effective introductory Computer Science tool.

Chapter 3 - Related Work

3.1 Dr. Scratch

Dr. Scratch [13, 16] is a Scratch project analysis tool developed by Jesús Moreno León of Spain. Originally developed in 2015, Dr. Scratch allows users to analyze projects for good coding practices, checks for "dead code," duplicate code segments, and good naming conventions [35]. The below sub-sections describe Dr. Scratch in more detail.

3.1.1 Description of Dr. Scratch

Using a simple Scratch URL or a downloaded Scratch SB or SB2 file, this web application will a score a project based on preset criteria. As seen in Figure 3.1, these criteria encompass a selection of foundational computer programming skills:

- Flow Control: The use of Scratch *control* blocks, such as forever and repeat until blocks, allowing for multiple repeated actions without the duplication of scripts.
- Data Representation: The use of Scratch *data* blocks storing data into variables and lists. This includes strings, colors, numbers, and pointing to other elements of a Scratch project by name.
- Abstraction: The use of the Scratch "more blocks" feature to create custom blocks that reduce repetitive scripts. Also includes using the clone block to create exact copies of functions and sprites.
- User Interactivity: The use of blocks that involve a webcam, keyboard, mouse, or microphone determines this criteria.
- Synchronization: The use of blocks requiring a specific criteria or event order to take place before executing. Examples are the broadcast and "wait until" blocks in the *Events* category.

- Parallelism: The use of blocks allowing multiple actions to occur simultaneously; for example, having multiple sprites react when a key is pressed or a mouse button is clicked.
- Logic: This concept is scored by using conditional blocks including *and, or,* and *if.*

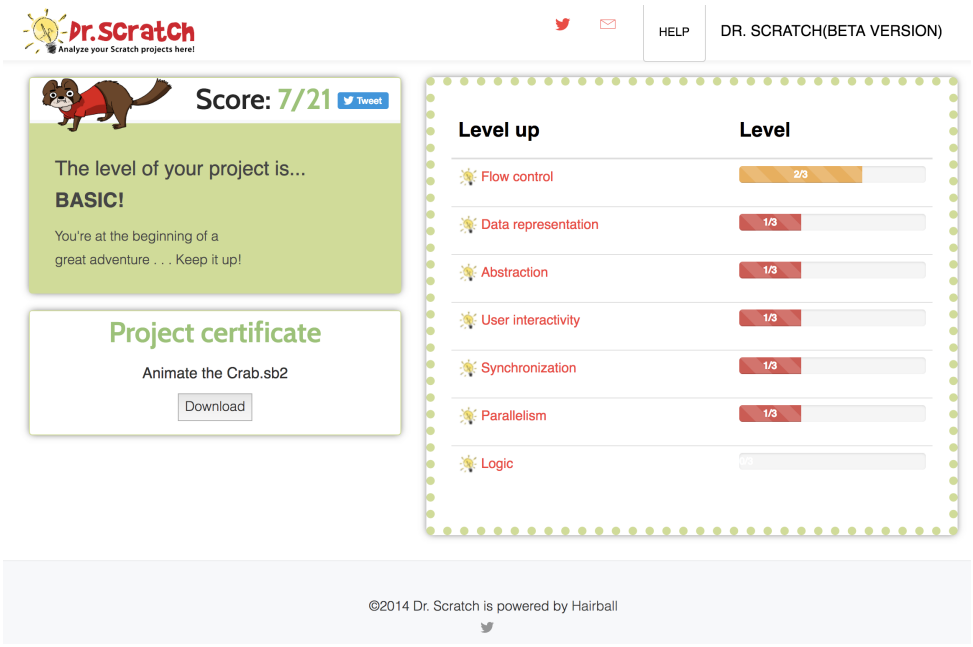

Figure 3.1: Dr. Scratch analysis page for a project at the Basic level

All of these are extremely helpful concepts and provide an excellent way to critique projects of any scope. Dr. Scratch also uses a simple three point scoring system to provide users with feedback for each concept. With a maximum score of 21 points, every project is categorized into one of three levels: 0-7 points for *Basic,* 8-14 points for *Developing,* and 15-21 points for a *Master* project. As a user progressively uses more advanced methods in their project, Dr. Scratch begins to report on more aspects of their project. In Figure 3.2, a Master project is shown with an additional section "Best Practice." This section further investigates the project being analyzed, showing the user issues with duplicate scripts, sprite naming conventions, dead code, and sprite attributes.

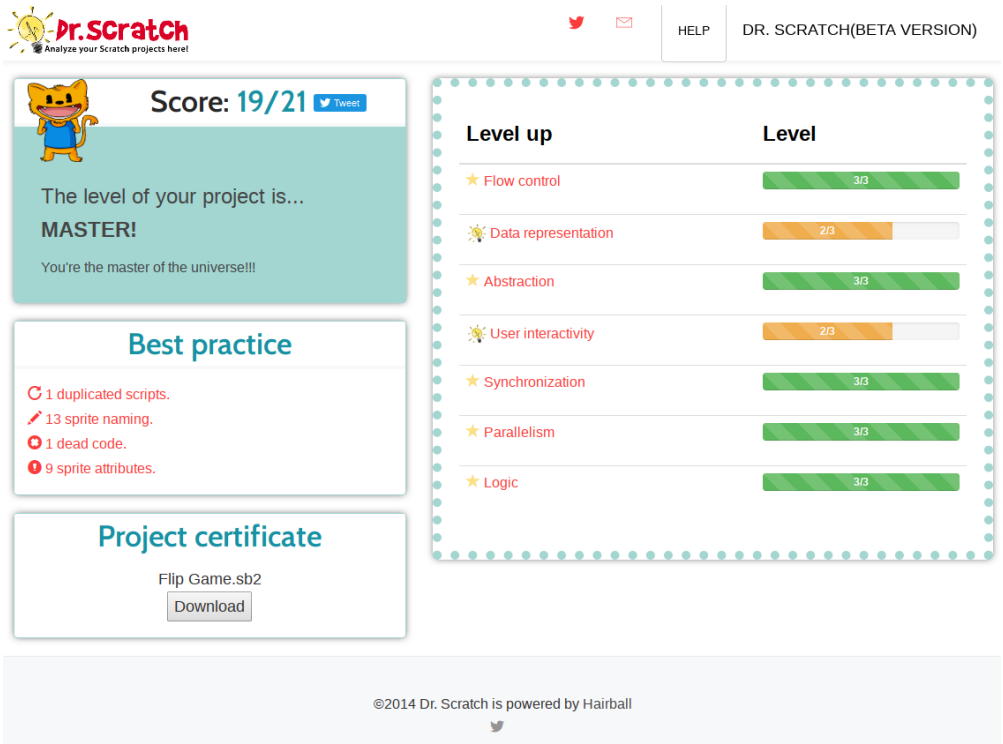

Figure 3.2: Dr. Scratch analysis page for a project at the Master level

3.1.2 Issues with Dr. Scratch

Dr. Scratch in particular is relatively unreliable when it comes to analyzing projects via URL. While the Scratch API [63] is regularly updated and well documented, most projects submitted via URL fail to be processed and analyzed through the Dr. Scratch website. Another issue that arose was with newer projects uploaded via the SB2 file. When these projects had relatively newer blocks in their projects, the tool failed to process the project altogether. This is more likely is due to its reliance on Hairball, which is described below. Another shortcoming of the Dr. Scratch tool is the analysis guidelines [37]. The guidelines themselves are excellent practices that, when followed as intended, promote growth in knowledge of programming. Unfortunately, these guidelines are explicitly explained on the website, showing how to obtain the maximum score with a set of blocks. When copied as displayed, a non-working project can achieve the maximum score. Alternatively, because the guidelines simply look for a specific

formula or block (e.g., an “or” or “and” operator block [37], “repeat until” loop block [36], or using a microphone or video camera [38]), projects that do show progress and effective programming skills without these blocks can receive low-level scores. While impossible to develop a tool that effectively covers all of the concepts of Scratch and programming in general, the Dr. Scratch tool unfortunately provides the means to allow users to beat its own system, which means it is not a good choice for educators needing help to evaluate student accomplishment and growth. Lastly, Dr. Scratch does not check to see if the project has been remixed, or copied, in the first place.

3.2 Hairball

Developed by the UC Santa Barbara Computer Science Education Group, Hairball [19, 31] is a low-level tool intended to be used by educators and students alike to promote “safe and robust programming practices” [7]. The goal of Hairball is to provide educators with a quick and thorough analysis of Scratch projects and promote use of programming practices within the classroom. “The main contribution of Hairball, however, is the framework and set of available plugins that support more sophisticated analysis. We want to answer questions not just about the use of Computer Science (CS) constructs, but about the competence demonstrated for different CS concepts” [7]. Discussed in the sections below is the relationship of Hairball with this thesis and issues that were uncovered during testing.

3.2.1 Description of Hairball

While Dr. Scratch is an interactive web application, Hairball is the Python-based framework for the static analysis used within Dr. Scratch. Hairball only reports on what is actually applied or run within the project, leaving out any blocks that go unused. Hairball calls these unused elements “dead code,” reporting them to the user upon final analysis. Hairball can

also report on the blocks provided in the report, but failed altogether when newer (post-2015) blocks were used within projects. Hairball does support the analysis of older Scratch 1.4 projects, although this is no longer really a needed feature as Scratch 1.4 projects become more obsolete.

3.2.2 Issues with Hairball

Hairball also has a fair amount of issues, including lack of standardized output and testing. Hairball also has a few processing issues as well. In *Hairball: Lint-inspired Static Analysis of Scratch Projects* [7], the author points out that, during a test of recognizing concepts manually vs recognizing concepts via the Hairball tool, "...manual analysis resulted in 32 false positives, and Hairball resulted in 33 false negatives...we consider a false positive to be an instance that was labeled correct, when in fact it is not, and a false negative to be an instance that is actually correct, but was not labeled as such..." [7].

Another issue with this tool is the lack of regular updates. As of this writing, Hairball has seen only minor updates since early 2014 [20]. Because of the Scratch tool's continued development and rapid growth, Hairball has become obsolete, not recognizing many of the new blocks introduced in recent years. This causes issues throughout other tools that use Hairball as a framework, an example being Dr. Scratch.

3.3 Kurt

Kurt [18] is another Python library which can be used to analyze and create original Scratch SB files [60]. The oldest Scratch analysis tool reviewed, Kurt, is a console based tool that analyzes a SB project file. This tool can be used to count the number of specific blocks used in a project, write a new project based on a raw text file, quickly import hundreds of images into a Scratch project. The newest installment, Kurt2, can convert the current Scratch format, SB2, back to the older SB file format.

Kurt has a few issues impeding its use as well. The first of these issues lies within the aging Python 2 version. Kurt currently requests Python 2.7.2 to be installed, which was released in July 2011. Along with this, the current Python 2.7 documentation states that "there will be no new full feature releases for the language or standard library" [65, 48]. Given the lack of any significant updates to the tool in over the five years, updates to the tool seem unlikely. Unfortunately, this stagnancy in production and lack of updates has also allowed multiple bugs to arise in the tool. No new blocks added in the last 5 years are currently recognized by the project analysis performed by the tool. These issues leave the tool mostly unusable for the purpose of this application. However, the older Scratch project SB files can now be converted to the new SB2 format by simply uploading them to the Scratch website. While still a good resource for older SB Scratch project analysis, the stagnancy of the Kurt code repository has caused the tool to continue to lose usefulness and thus is not used in this application.

3.4 SCATT

Students at Appalachian State University also worked on a Scratch analysis engine called SCATT, the SCratch Automated Textual Transcriber [15]. Unlike the Hairball analysis tool, SCATT solely analyzes the code based on the number of specific elements used within the project. The purpose of the software was to take in a Scratch project SB2 file and produce a report on the number of distinct blocks, costumes, scripts, and other elements within the Scratch project. SCATT is not an interactive graphical tool but rather a low-level, command line tool that generates text file reports.

Another issue that arose was the SCATT reliance on user input. The SCATT tool is passed a SB2 project location to the program via a user supplied directory. This directory is then used to locate the uploaded project to be analyzed. With some slight code changes to SCATT,

this directory location input was configured for automation. This also means that the modified SCATT tool had to be packaged and built into the SPAE tool.

3.5 Conclusion

While Dr. Scratch and Hairball are the most frequently cited Scratch Project analysis tools, further investigation proved that these tools were incomplete, had a number of bugs, or simply hadn't been updated in multiple years. SCATT is more reliable than these two analysis tools, but lacks web-based access and is solely used via a terminal. This lack of options combined with the shortcomings provided an excellent opportunity to develop a new framework for analyzing Scratch projects in a new and objective way. Learning from the shortcoming of these tools, the modified SCATT tool will be the basis of analyzing Scratch projects for this thesis.

Chapter 4 - Methodology

The Scratch Program Analysis for Educators tool (SPAE) is a web application that provides a user with the ability to analyze Scratch projects. SPAE is intended to provide primary and secondary educators with objective feedback about student Scratch projects that can assist in evaluation. However, it can also be used by students or parents to monitor understanding and progress. It is important for SPAE to be reliable and responsive; teachers may be submitting dozens of Scratch projects in a single session and may be using older school-issued hardware. This chapter describes the components of SPAE including the Scratch API, the *ProjectDownloader.py* Python script [17], the SCATT engine [15], and other implementation aspects.

4.1 Overall SPAE Architecture

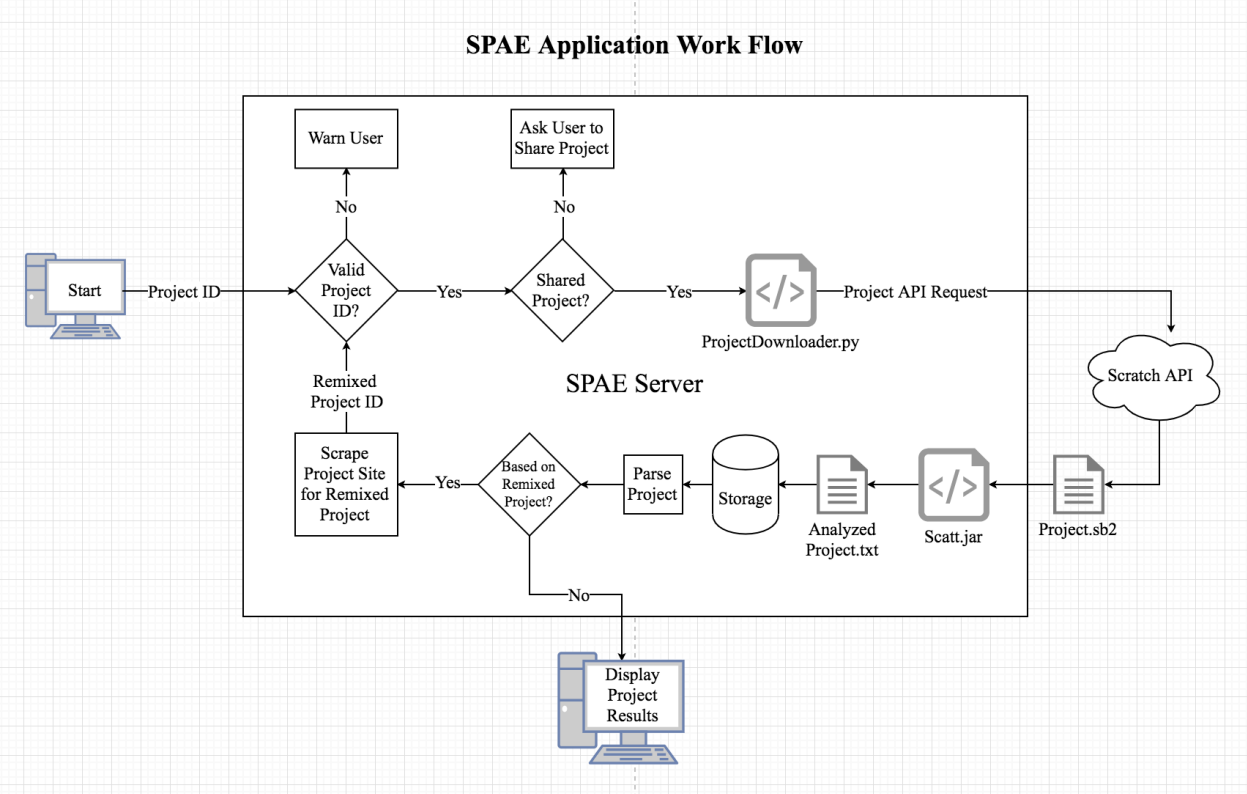

Figure 4.1: Overview of the SPAE Application

Figure 4.1 shows the complete process for the SPAE tool, from the Scratch project ID submission to the display of a Scratch project's statistics. The SPAE user submits a Scratch project ID or URL to the server via the SPAE home page, which can be seen in Figure 4.2. Upon submission, the server checks that the submission is formatted correctly and that the project is shared. If either of these criteria aren't met, an error message is shown to the user suggesting corrective actions. After confirming these criteria, the server tests for the project's existence in the Scratch API. If it is accessible, the *ProjectDownloader.py* utility is invoked. This utility communicates with the Scratch API to receive the Scratch project SB2 file.

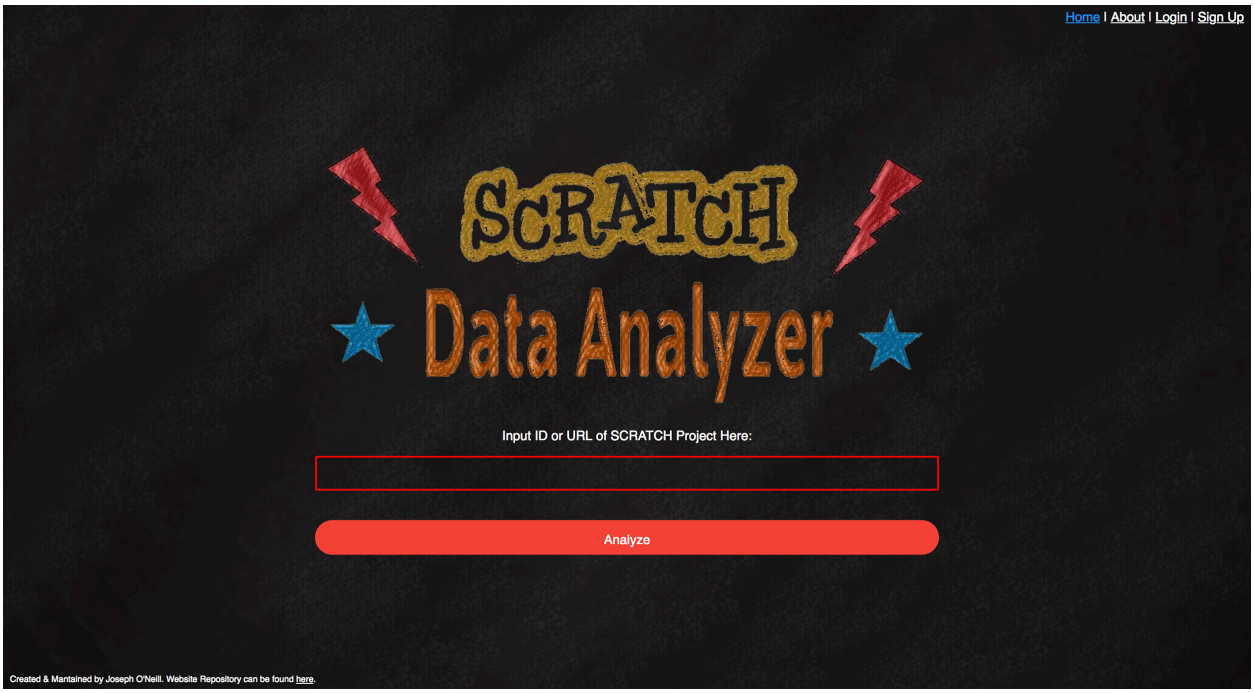

Figure 4.2: The SPAE Home Page

After returning a successful download, the back-end script then invokes the SCATT utility which will generate a text file as output. The server then scrapes the Scratch project site for any remixes this project may be created from. If the project is a remix of a previous project or has been remixed multiple times from an original project, the server grabs both the original project and the most recently remixed project IDs, downloads those corresponding Scratch

projects, and runs the analysis again, creating a text file for each of the scraped projects. The script then accesses all newly created text reports from SCATT and parses them to a final webpage, which can be seen in Figure 4.3. On this page, the user can see the parsed elements of each individual project. To name a few, these include the number of scripts, sprites, and variables used, as well as the number of specific block types, e.g., *Control* blocks, *Data* blocks, *Event* blocks, etc. The Report menu option allows the user to view the raw SCATT analyzed project document. The project's actual Scratch project page at the Scratch website is also available via the Project Page menu option.

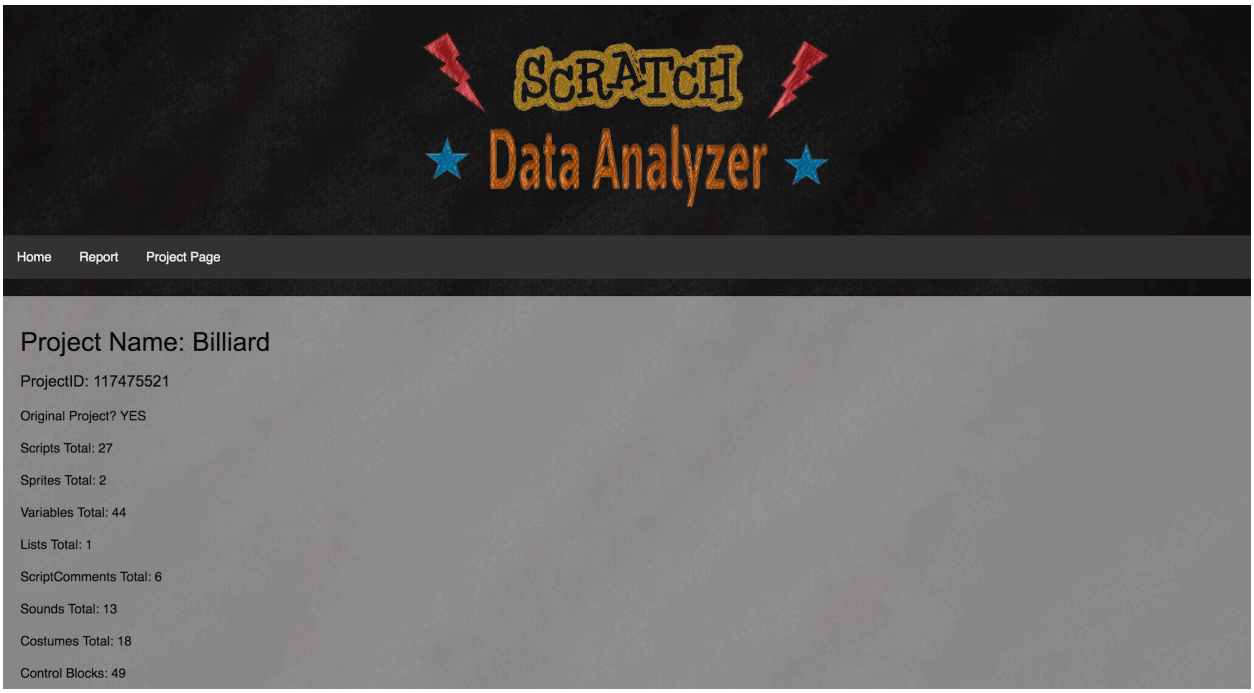

Figure 4.3: Scratch Project Analysis Page

4.2 Scratch API

There are currently two primary instances of the Scratch API, one for the older Scratch 1.4 projects [62] and one for the newer Scratch 2.0 projects [63]. The Scratch 2.0 API was selected for this project as the older 1.4 version will soon begin to phase out. Future references to the Scratch API in this thesis will refer to the Scratch 2.0 API. The Scratch API is well documented and provides information that is publicly available on the Scratch website. These

queries include the current health of the Scratch website, the current number of Scratch projects stored on the site, the current featured projects, and even current featured news about Scratch. In addition, the API[2] allows a user to download a Scratch SB2 project file.

Originally, SPAE was envisioned loading a Scratch project file from a user's local machine. But upon discovery of the Scratch API and given that most users no longer save their projects locally, the SPAE tool design changed to incorporate the use of the Scratch API to access projects. This decision brought an issue to the surface very quickly. The size of the Scratch project files can impact the speed of the download process. Ranging from a few kilobytes to nearly half a gigabyte, the size of downloaded SB2 files proved to be difficult to predict. The large variance in the sizes of downloaded projects was found to arise from the number of images and sounds used in each project. As could be expected, more intricate projects often had more elements for various actions. With every image used in a Scratch project, the corresponding SB2 file became larger as well. In fact, during a round of preliminary testing of SPAE, the tool downloaded just over 50 GBs of data from 5000 random Scratch projects. While this is an extreme use case of the tool, this realization prompted the rethinking of the storage of downloaded projects. Thus, SPAE employs some rudimentary data management techniques because testing the tool requires downloading hundreds of thousands of Scratch projects. Full SB2 projects are currently saved for a period of two hours, while the much smaller SCATT analysis output text files are stored indefinitely. The API also makes it possible to ignore the media components and only download the JSON file contained within a Scratch project, which contains the basic description of the project and is much smaller in size. This option was not

[2] The APIs and descriptions are accessible from the https://en.scratch-wiki.info/wiki/Scratch_APIs website.

pursued in the hopes that future analysis and development could be attained by examining the full SB2 files.

4.3 Project Downloading

In order to sufficiently analyze the command and inner workings of a Scratch project, it is necessary to retrieve the SB2 project file. Dylan Beswick has developed a Scratch project downloader script in Python [17]. A major benefit of using this tool over directly interacting with the Scratch API is the tool's ease of use. By simply providing a Scratch project ID, all of the packaging required to produce a SB2 file is taken care of, while also allowing the user to specify the storage location. Unfortunately, the Scratch project downloader script was developed to be used via a console. This required modification to the script to allow for automation.

An impactful issue arose at this point that involved how the contents of a Scratch project are encoded for download. The original Python project downloader script used CP850 encoding and decoding for downloading Scratch projects. Whereas, the Scratch API uses UTF-8 encoding. These two approaches, unfortunately, do not always match up, particularly with non-English characters such as the Spanish characters â, ñ, and ó. Whenever there was a mismatch in the encodings, the download would fail. This inevitably led all projects using non-CP850 characters to fail to be converted.

When the discrepancy was found, the SPAE tool had already been tested against the entire Scratch project data set of over 800,000 projects. However, all Scratch projects that had failed were saved for closer inspection. After changing the encoding in the Python project downloader script from CP850 to UTF-8, the previously failed projects were re-run. This change resulted in 533 new successes of the 561 previously failing projects. The remaining 28 failing projects were due mostly to SB2 files that contained very large audio files.

4.4 Project Scraping

In order to test its reliability, SPAE needed to be tested against a large number of Scratch projects. At first, testing SPAE was attempted by using random numbers as project IDs. This proved troublesome, as many projects are not shared publicly, other projects previously using a project ID had been deleted, and many random numbers were not project IDs at all. While unshared projects are still accessible through the API, the Scratch project page is not, making remix analysis impossible. Because of the popularity of CS First and the curriculum's use of a wide variety of Scratch blocks, it was decided to attempt to web scrape as many CS First projects as possible. In addition, CS First student projects are often based on a remix of a starter project making these good choices to ensure the proper functioning of that SPAE analysis feature.

The first step towards this goal required finding the original CS First projects. These were easily found at the CS First home website, which provided an initial, valid project ID. Using this project ID, SPAE can access the project's remix page that provides links to every remixed project based on this project. Figure 4.4 shows an example of the remix page for a popular CS First Scratch project named "Impossible Cube!" Thus, from this one project it is possible to get valid project IDs for many other Scratch projects. While the smaller or newer CS First activities only had a few thousand remixed projects, those in the more popular curriculum themes, such as Game Design, could have well over 100,000 remixes per project. For testing, the SPAE tool would simply need a large number of project IDs. Deciding to use this as a starting point, the web scraping tool *getRemixedProjects.sh*, a Bash script used to retrieve remixed projects, was developed to retrieve as many projects as possible.

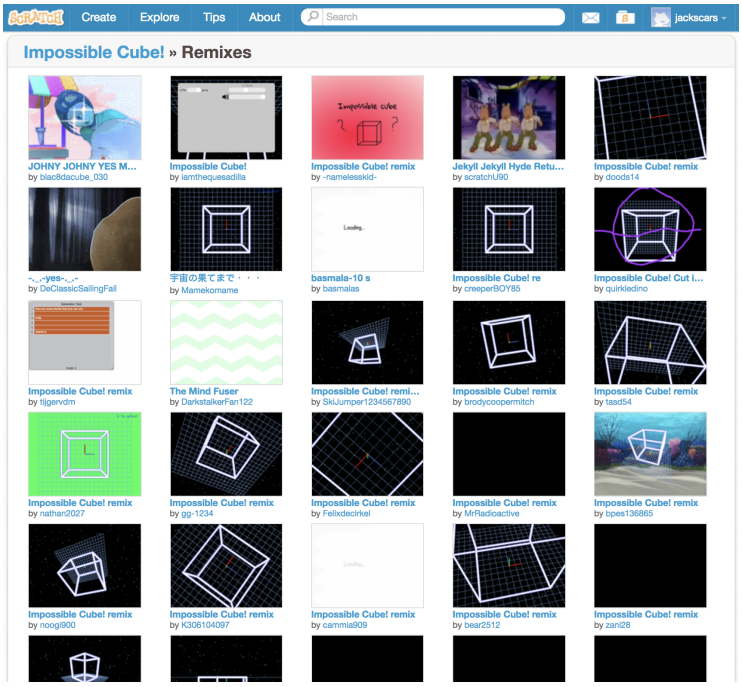

Figure 4.4: Example of the project remix page for web scraping

4.5 Project Sharing

Part of the drive behind the SPAE tool is the desire to compare the amount of original work a Scratch programmer has put into a project with the amount that was inherited from the remix project. Because a major part of the Scratch tool is the ability to share projects with other users, being able to check if a project is not an exact copy of a previous project will allow educators to better evaluate their students. However, often times users are not prepared to publicly share a project that is in the process of being developed. This isn't an issue for the API, as the service can still access and download the project. However, in regards to Scratch projects, the API can currently only retrieve the images, sounds, blocks, and Scratch project elements in the SB2 file but not the project page that lists remixes.

The Scratch API does not keep track of any remixing related to a Scratch project. Thus, there is no way to check if a project was remixed or is an original work through a Scratch SB2 file. Currently, the only way to check for project remixes is through the Scratch project's

webpage. A user can see whether a project is an original work or has been remixed from another project in the credits section of a project page. This section allows us to scrape for any original or remixed project simply by knowing what project id we need to evaluate. However, this solution does come with a single caveat: in order for our application to be able to scrape the web page, the project must be shared on the Scratch website. Without this, only an analysis of the original Scratch project can be achieved.

4.6 SCATT - A Scratch Project Analyzer

Now that a Scratch project SB2 file can be obtained, the next step is to examine the contents of the SB2 file. As described in the related work chapter, there are two Scratch project analysis engines available: SCATT and Hairball. The major benefit of the SCATT tool is its high level of reliability. During testing, SCATT provided accurate analysis of Scratch projects 100% of the time. Because of the analysis and organization that SCATT uses, the reports provide an easy and efficient method for cross-examination of projects. Hairball takes a much different approach, providing the user with a higher level of analysis, such as the ability to test for dead code, duplicate scripts, and say/sound block synchronization [7]. However, its lack of reliability in successfully completing analysis on many projects, its lack of software updates, and its output standardization were particular areas of concern.

The results of the SCATT analysis tool, as can be seen in Figure 4.5, provide the basic and necessary elements of a Scratch project. The file section describes the overall elements within a Scratch project. The stage counts section describes the elements within each project stage or sprite. However, SCATT did need to be automated in order to work with SPAE. The original SCATT tool required a user to manually input a Scratch SB2 project via command-line.

This in turn had to be rewritten to work in conjunction with the Python project downloader script.

```
SCATT Report October 17, 2018 5:37:02 PM

File: project-244493453.sb2
---------------------------------
Scripts Total:                  25
Sprites Total:                  6
Variables Total:                0
Lists Total:                    0
ScriptComments Total:   0
Sounds Total:                   3
Costumes Total:                 16
Control Blocks:                 18
Data Blocks:                    0
Event Blocks:                   31
Looks Blocks:                   53
More Blocks Blocks:     0
Motion Blocks:                  12
Operator Blocks:                2
Pen Blocks:                     0
Sensing Blocks:                 1
Sound Blocks:                   7

Stage Counts
---------------------------------
Scripts:                                5
Variables:                              0
Lists:                                  0
ScriptComments:                 0
Sounds:                                 2
Costumes:                               9
Control Blocks:                 12
Data Blocks:                    0
Event Blocks:                   6
```

Figure 4.5: Partial example of a SCATT project report

4.7 Other Software and Tools

Along with the major pieces of software described earlier, there was also a need to develop tools to aid in server management, web scraping, and the dynamic display of web pages. Python and Java were used in the SCATT and Python project downloader tool, respectively. However, most development for the web server was done in Perl, using previously developed packages to handle program processing and web page representation.

Perl 5, hereafter referred to simply as Perl [3], is a particularly powerful parsing language, borrowing many elements from AWK, sed, and C programming [50]. Also capable of being utilized as a Common Gateway Interface (CGI) scripting language, Perl allows users to run

scripts and applications which in turn write dynamic web pages. Gaining popularity in the late 1990s, Perl has a vast array of tools (also known as “Packages”) for numerous applications in data munging, web development, and managing systems work. These reasons, alongside the ability to parse large documents, develop dynamic web pages, and execute other various scripts on the server, made Perl the clear choice in the development of this application.

4.7.1 Third Party Software

Various third party software components were used in conjunction with Perl to produce the SPAE web application. This software consists of the following.

- **Apache Web Server**

  A web configuration tool used to allow web access from a server [2]. Apache is a free and open source HTTP web server that can be used on most operating systems. Developed in the mid 1990s, Apache is the most used web server in the world, accounting for around 39% of all website configurations [41].

- **RRDTool**

  The Round Robin Database Tool (RRDTool) [42] is another free tool designed to track regularly timed data, such as CPU Usage, memory usage, and other system data. Often paired with Perl, RRDTool is used during SPAE testing to graphically observe the system impacts of SPAE.

4.7.2 Program Specific Software

Software was also developed for use in the SPAE to test the tool for hardware usage and reliability. Software was also developed to web scrape large numbers of projects from Scratch remix webpages:

- **Autorun Bash script**

  A Bash script developed as a part of this thesis to enable thorough testing of the SPAE web application. This script imitates a user submitting a job through the web interface. The script allows for the back to back processing of Scratch projects, without the need to submit a project ID or URL by hand.

- **Remixed projects Bash script**

  A shell script developed to support the goals of SPAE related to student projects being based on other projects. This script calls the remix page of a specific Scratch project, then scrapes the Scratch website specific to the requested project for any remixed projects. While this script was used and developed to retrieve most of the CS First projects used in the results section and for hardware testing, the script can also be used on any Scratch project with remixes.

4.8 Hardware and Access

Public school teachers considering Scratch have very limited funds, so an important complementary goal of SPAE is to be a resource that users could set up easily, for a very low cost, and require minimum maintenance. Therefore, several different hardware configurations were evaluated. Early testing of the software used in this program proved that the tools used within the application are not particularly taxing on the hardware. CPU and memory usage was minimal, typically staying below 20% total CPU usage and 30% total memory usage. This prevents the need for expensive server machines and allows relatively affordable and available devices to host the application. In addition to the capabilities of the host machine, the SPAE application needs a very reliable Internet connection to deal with the sometimes large Scratch project files being downloaded from the Scratch API.

As a hosted web application, many users should be able to access the system. To ensure the effectiveness and reliability of SPAE, it was necessary to perform a number of tests on different and varying systems for two reasons. First, it is necessary to ensure that this application would be tested and succeed on different platforms using different hardware for users with different budgets. Second, understanding application behavior on multiple platforms can provide a better idea of what device may be the best option for a particular user. It was decided to test the application on three different sets of hardware: a Raspberry Pi, a cloud server service, and an older, surplus personal computer.

4.8.1 Raspberry Pi

The Raspberry Pi [53], as seen in Figure 4.6, is a simple System on a Chip (SoC), single-board computer. Officially released in 2012 [49], Raspberry Pi systems are known for their affordability ranging in price from $5 to $35. The systems come equipped with most modern peripherals and can also utilize their on-board 40-pin pinout [51].

Because of its low cost and robust hardware, the Raspberry Pi 3 Model B [1] was selected. With the high level of documentation and number of projects being developed on Raspberry Pis, it was quick and painless to install all of the necessary components on the device to prepare for testing and device comparisons. Figure 4.6 shows the Raspberry Pi device used for this study. Across the top of the device there is an SD storage card and a white Ethernet cable for Internet connection. Along the right side is an HDMI connector to a display screen for monitoring and controlling the device and a power cable. While not shown in Figure 4.6, the Raspberry Pi does allow for a usb or bluetooth connected keyboard and mouse, which were used during the installation of the SPAE tool.

Figure 4.6: Raspberry Pi 3 Model B

4.8.2 Cloud Server Service

For cloud server testing, prior successful experience with the Linode hosting service was a deciding factor in choosing the cloud service provider. In terms of specific service configurations, Linode's cheapest option, the Nanode 1GB service, was selected. The $5 per month option [30] provides Internet access, a selection of popular Linux distributions, and roughly around the same device specifications as a Raspberry Pi 3 Model B. While the service does not provide direct access to a physical machine, it does allow configuration of the cloud server to the user's specification.

Aside from eliminating the necessary physical hardware maintenance, a cloud server service has many benefits over a traditional physical server:

- Cloud server hosting has an incredible track record for maintaining up-time. In fact, most cloud server hosting services guarantee an uptime of over 99.9% [29, 25].

- Most cloud server hosting services allow customization of the server to nearly any specification, including what operating system to deploy.
- Cloud server services are typically centered around areas with excellent Internet access. Internet access is also included in all hosting plans, but there is typically a limit to the amount of downloading and uploading on the server.
- Hosting plans for lower tiered plans are very affordable, often costing less than $10/month.
- Cloud server services are scalable, allowing for affordable options to be upgraded as necessary, including the number of CPU cores available, memory usage, transfer volume and disk space.

One of the other major benefits to this service is the built in monitoring included with a subscription. Using the RRDTool package, the service tracks CPU, memory, and disk usage, allowing users to quickly track down processing issues and receive alerts whenever usage hits a user defined threshold. Figure 4.7 shows an example of this tool in action. During testing, the cloud server was running a large selection of Scratch projects using the SPAE tool. One of these Scratch projects caused the tool to max the CPU at 100% for a period of about a day. The cloud service noticed this issue after about two hours and sent a message to the affiliated account email. This allowed for a fast response that included stopping and researching the behavior.

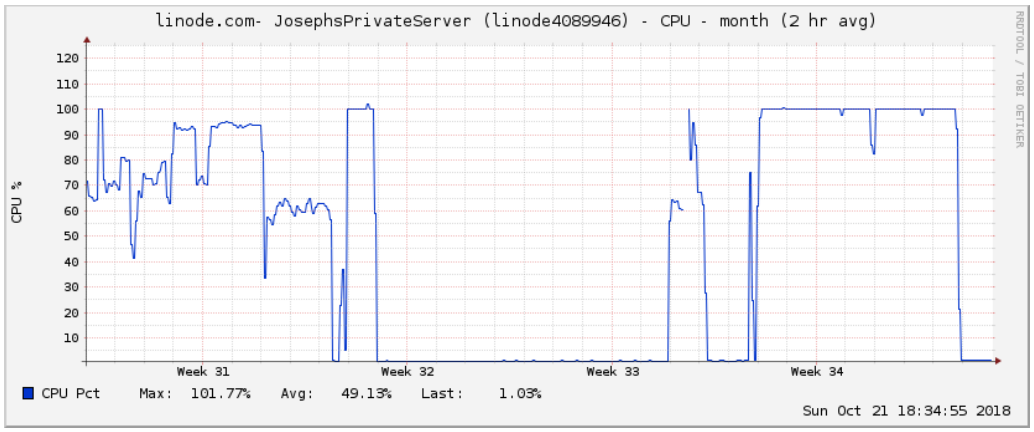

Figure 4.7: Example of Linode CPU Monitoring

Included with this service is the occasional upgrade in memory and disk size, which often proves to be expensive on a traditional server. Along with a nearly constantly Internet connection, the device can be accessed and managed from anywhere with reliable internet access. These elements made installing the SPAE software to the machine a simple task. After registering with the cloud server provider, SPAE was deployed on the new cloud server within 30 minutes.

4.8.3 Personal Computer

While most software is typically designed for use on current or even future hardware, more often than not educators must work with strict budgets that often leave them hesitant to spend what little money they do receive on something they haven't worked with before. Even more, teachers are often supplied with older machines and technology for their classrooms, leading them to make do with what is provided for them. Thus, it was desired to use an old and free item of hardware to ensure the SPAE software can be utilized by anyone, regardless of their budget and constraints.

For this test, a 2008 Apple MacBook [26] was used. With an average lifespan of around four and a half years [66], most computer users have a working desktop or laptop that hasn't been used since a recent upgrade. Also, most school teachers know of donation programs that

provide older computers to individuals that meet a few criteria [45]. Unfortunately, more often than not, older machines require software updates, which can add to the deployment time of the SPAE software and can often lead to dealing with issues of incompatible versions of software. For the 2008 MacBook machine, several hours were required to update the OS X operating system before installing the SPAE software, which did end up having a few dependency issues. Regardless, a successful installation of SPAE was achieved.

Figure 4.8: Old MacBook used for testing

Figure 4.8 shows the laptop used for the testing and installation of SPAE. At the time this photo was taken, the laptop was running OS X Mountain Lion. Unfortunately, the issues encountered trying to upgrade this outdated system may be too difficult for someone without a fair amount of experience. With the realization that an outdated operating system could also negatively impact continued SPAE development, an alternate operating system may be needed. The user-friendly, easy-to-install, and free Linux operating system, Ubuntu Bionic Beaver, was chosen to resolve these issues. The SPAE software was then promptly and successfully installed with a noticeable increase in performance. The only issue of note was with wireless connectivity.

This is a common issue among laptops with Linux distributions and so there is ample documentation online for each distribution to help diagnose and resolve the issue.

4.8.4 Hardware Comparisons

The SPAE software and required dependencies (e.g., Perl) were installed and properly configured on all tested hardware, albeit with varying levels of difficulty for certain dependencies. Each piece of hardware was set to factory settings prior to installation and then configured as needed. Table 4.1 displays the primary hardware features as well as some of the more notable differences between the tested hardware.

Table 4.1: Comparison of Hardware used for Testing [48, 31, 3]

Hardware	Raspberry Pi 3 Model B	Linode Nanode 1GB	2008 13” Apple MacBook
Operating System	Ubuntu 16.04.4 LTS	Centos 7	Ubuntu 18.04.1 LTS
Cost	$35 (+ $15 for SD card)	$5/Month	Free
CPU Cores	4	1	2
CPU Frequency	1.3 GHz	2.8 GHz	2.4 GHz
RAM	1 GB	1 GB	4 GBs
Disk Space	30 GB	25 GB	1000 GB
Network	100 Mbit/s Ethernet, 802.11n Wireless	Net Out: 1000 Mb/s Net In: 40 Gb/s	1000 Mb/s Ethernet, AirPort Extreme (802.11a/b/g/n)

Chapter 5 - Results

5.1 Project ID Scraping

Testing for the SPAE tool required the use of a large assortment of Scratch projects. Because of the educational nature of CS First, most blocks and concepts contained within Scratch are covered throughout the program's nine curricula. This provided an excellent testing ground for various Scratch concepts against the SPAE tool as well as providing the option of a basic understanding of the effectiveness of the CS First program.

The first task of obtaining the CS First remixed project IDs was finding all of the original CS First projects. This proved to be less problematic than originally feared by simply trying out each curricula from the CS First website whereupon the 147 starter projects were provided in the associated activities. Using the remixed projects Bash script, 134 of these 147 CS First starter projects were successfully web scraped for remixed projects. The software was unable to download the 13 remaining starter projects because of their popularity; the download would timeout. At the end of the scraping process initiated from the 134 starter projects, a total of 802,191 project IDs were scraped from the CS First projects. When attempting to load a popular Scratch project remix page, the Scratch website can timeout, resulting in an error. It is estimated that approximately 500,000 Scratch projects were excluded because of this, particularly among the popular Game and Sports themes. While revisiting this issue and improving on the process of attaining Scratch project IDs for future developments of the SPAE tool is preferable, the number of project IDs secured proved to be more than sufficient for our testing purposes.

5.2 Hardware Reliability Testing

To test how well the SPAE software can run on the different platforms, 5000 of the CS First Scratch projects from our web scraped data set were randomly chosen. This selection was

run on each machine, tracking the average CPU and memory usage every 5 seconds for the duration of each run. Successful runs vs total runs and start/end times were also tracked to test for reliability and speed of each system. To collect this data, the autorun Bash script used a TXT file containing the 5000 Scratch projects to be analyzed back-to-back. This script also used the *sar*[3] command package to track CPU and memory usage. Table 5.1 shows the results of these tests in detail.

Table 5.1: Comparison of Runtime testing over 5000 Random Scratch Projects

Hardware	Raspberry Pi 3 Model B	Linode Nanode 1GB	2008 13" Apple Macbook
CPU Usage Average	22.76%	75.38%	43.31%
CPU Usage Min/Max	0.2% / 97.39%	10.46% / 100%	1.09% / 67.77%
Memory Usage Average	88.60%	82.33%	65.97%
Memory Usage Min/Max	58.01% / 97.22%	79.16% / 86.71%	61.93% / 67.58%
Total Run Time (HH:MM:SS)	08:23:08	01:57:35	04:00:19
Analysis Average Time (sec/project)	6.04 sec / project	1.41 sec / project	2.88 sec / project
Successful Run %	5000/5000 = 100%	5000/5000 = 100%	5000/5000 = 100%

As can be seen from Table 5.1, all three systems were successful in analyzing the 5000 projects with a 100% success rate. The fastest system was the Nanode server, which completed its testing in just under 2 hours with an average of 1.41 seconds per project. The old personal

[3] Sar, or "System Activity Report," is a system monitor used to report various activities being performed on Linux machines

computer finished in just over 4 hours, averaging just over 2.88 seconds per project. The Raspberry Pi finished in nearly eight and a half hours, averaging just over 6 seconds per project. This speed increase in project processing time correlated with the type of storage used on each device. The Nanode server, old personal computer and Raspberry Pi used SSD, HDD, and SD card storage respectively. The write speeds on each of these devices corresponded with the increase or decrease in analysis speed on each device; SSD access is typically about 200 MB/sec, HDD is about 100 MB/sec, and SD card speeds are about 25-50 MB/sec. Incorporated with the slightly slower Internet speeds for the old personal computer and Raspberry Pi, the run results match up appropriately.

The CPU averages for each system also correspond with the number of cores within each CPU. For example, since each instance of the SPAE tool is a single process, most processing was performed on a single core at a time. Where the Nanode server had a single core and averaged about 80% CPU usage, the Raspberry Pi and its 4 cores averaged about 20% CPU usage. This suggests a relatively level and manageable amount of CPU balancing across systems when using the SPAE tool. However, the min/max CPU usage of both the Raspberry Pi and Nanode server varied wildly, ranging from around 10% all the way to 100% CPU usage. The old personal computer, however, showed a fair amount of stability in this area, never rising above 70% usage.

Average memory usage suggested relative stability across all systems. This testing also proved that a system with as little as 1 GB of RAM would be sufficient enough to handle our tool, even after hours of repetitive analysis. However, both the Raspberry Pi and Nanode servers once again showed a high level of stress, averaging well above 80% memory usage, whereas the old personal computer was fairly stable in the mid 60% area.

All of the systems tested reported positive results, particularly with each system's 100% analysis success rate. In fact, the largest factor for the SPAE tool seemed to be the storage device used in each system. Memory and CPU usage also did not seem to be a large issue for any device, even with the constant bombardment of Scratch projects. With this knowledge, all of these systems can be recommended to run an instance of SPAE, with a few considerations. After viewing the system results, it is highly recommended to use SSD storage on your system to maximize performance. If the user anticipates a large number of project analysis in a short period of time (e.g. during an after-school program such as COSMIC), a multi-core CPU would be recommended. However, this test also proved that, given a tight budget, the SPAE tool can be effectively and successfully used on most recent systems with Internet connectivity, free of cost.

5.3 Project Analysis

Because of the large number of projects to be analyzed, the Nanode server was used to analyze the CS First Projects web scraped in the previous section. Using the autorun Bash script, the entire dataset was loaded and run over the course of about 17 days. Figures 5.1, 5.2, and 5.3 show a sample of a single day of runs. Figure 5.1, showing CPU usage, shows a few "jumps" in CPU usage. This is to be expected, as projects are lumped together based on what original project they are being parsed from. For example, the low plateau around 9:00-14:00 shows all projects remixed from the first of CS First Storytelling starter project. The graph then spikes upward around the 14:00-23:00 time slot, showing that the remixes to the second CS First Storytelling starter project have begun to be analyzed. These spikes continue throughout testing and provide insight to when a new batch of remixes from another starter project have begun to be analyzed.

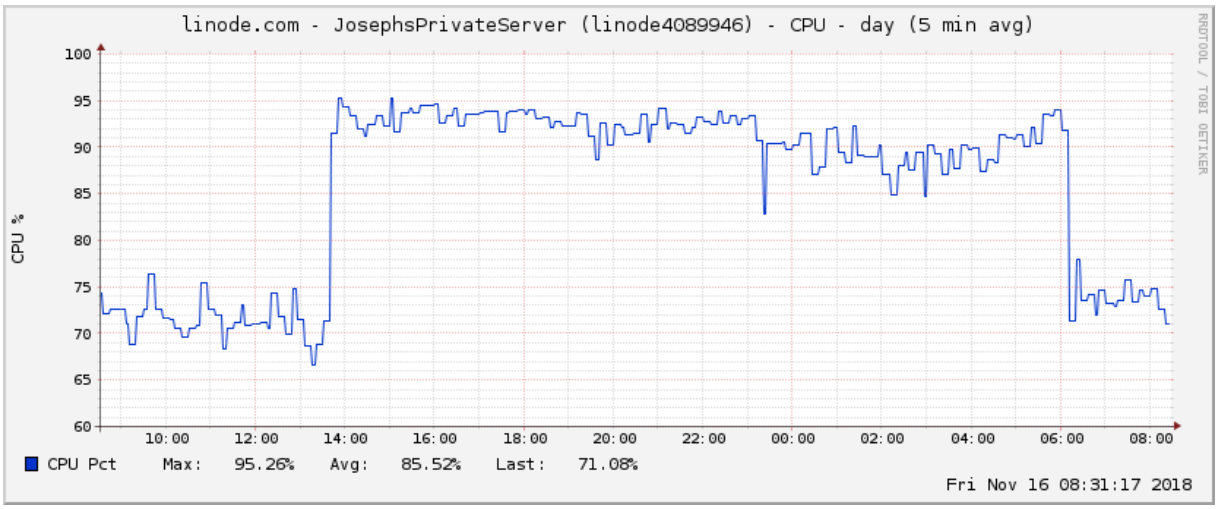

Figure 5.1: Example of CPU usage for a single day of continuous runs

Figure 5.2, which shows the network activity during the same time period, suggests that larger projects also contribute to CPU usage by requiring the hardware to wait for a download to complete. The bottom portion of Figure 5.2 also shows the total traffic for the tool. The figure shows that in a single day with a high level of activity, it is possible to transfer over 80 GB of data using SPAE. While this kind of transfer is unlikely to be encountered outside of testing, a server with a high level of use may require more disk space.

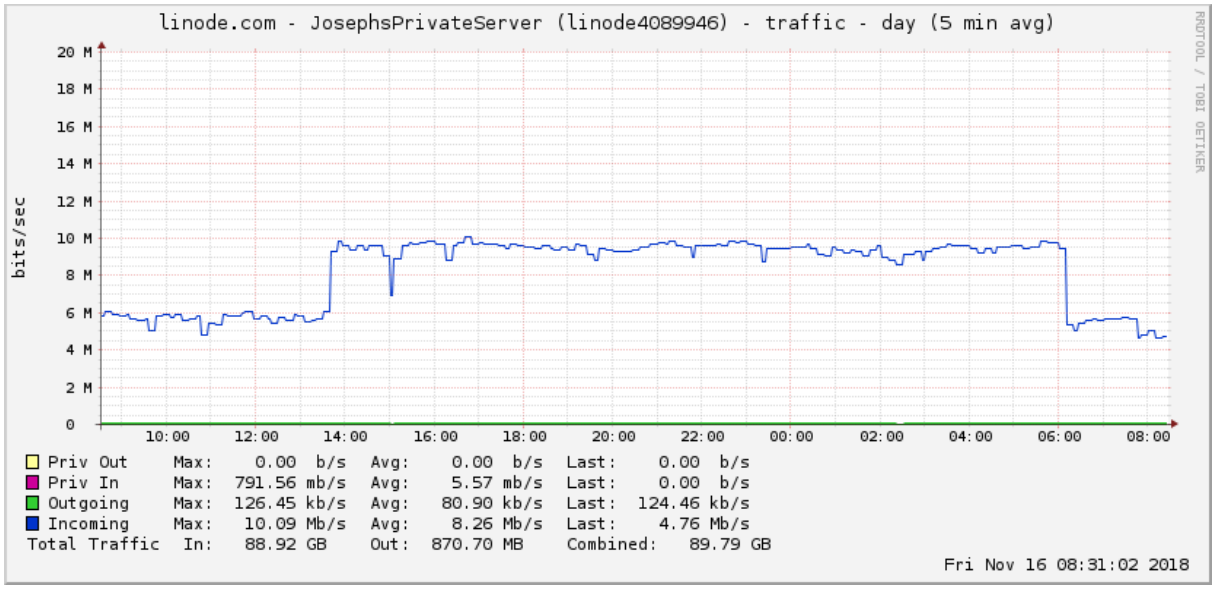

Figure 5.2: Example of Network Traffic for a single day of continuous runs

Figure 5.3 shows the CPU usage and Figure 5.4 shows the Network traffic of the entire dataset. In these figures, it becomes easier to see how the remixed projects are lumped together to run based on their original project. The "peaks and valleys" seen in these two figures line up with the timing that each new set of starter project remixes begin to be analyzed. One has to deal with image and sound intensive projects, which require more network traffic to download with no to little increase in analysis results. This results indicates why it may be beneficial to downloading only the JSON portion of the SB2 file and not the full project. The two figures also match up relatively well at three particular spots that demonstrate the same exception of SPAE crashing due to errors in SCATT. In the third and fourth days of Week 30, Figure 5.3 shows CPU usage hitting 100% for an extended period of time. This occurs again at day 6 in Week 31. Similarly, Figure 5.4 shows the network traffic falling to 0 at these times. Upon realization of these errors in Week 30, the failing project IDs were recorded, the SCATT process was killed, and the run continued to the next project. Upon completion of the entire run in Week 31, the two failing projects were given twelve hours each to complete analysis with SCATT; and, both projects failed once again. Section 5.4 provides a deeper analysis of these failed projects.

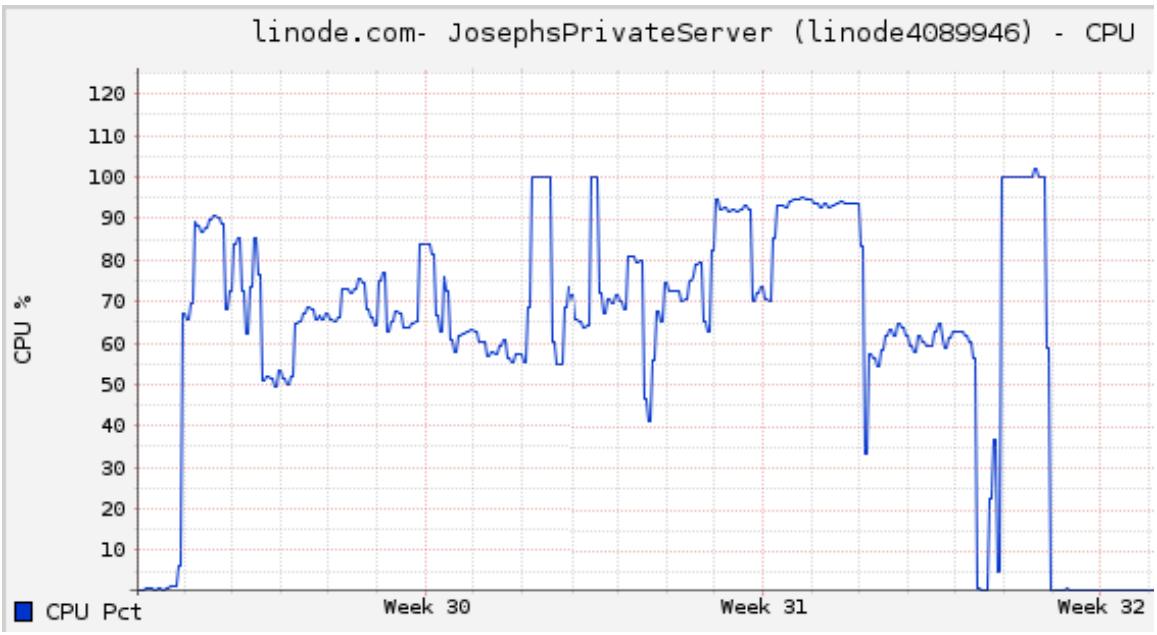

Figure 5.3: CPU usage during entire CS First dataset run

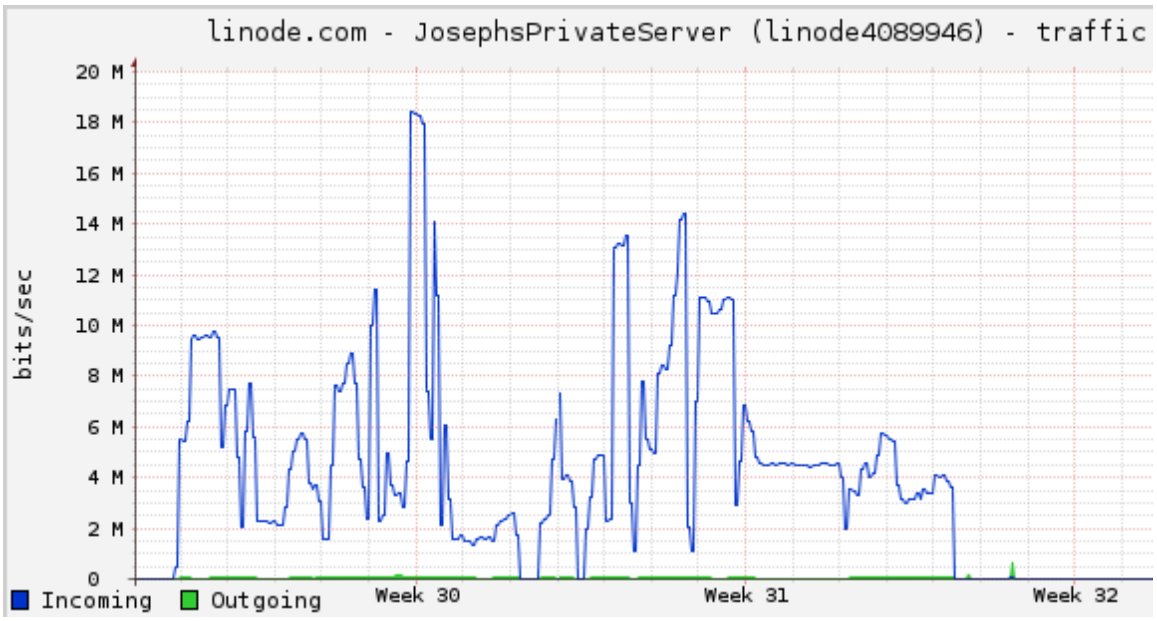

Figure 5.4: Network traffic during entire CS First dataset run

Aside from these two projects, the autorun Bash script performed analysis on all projects. On the original run, the encoding issue with the Python project downloader script discussed in section 4.2 had not been corrected, which resulted in errors with 18,872 projects. Upon completion of the initial full run and fixing the encoding error in the Python project downloader script, these projects were rerun, reporting successful runs for 18,837 of the 18,872 original

failed runs. Including the two projects that crashed the SPAE tool, this results in 802,156 projects being successfully analyzed out of 802,191 projects, or a 99.9956% success rate of project analysis using SPAE.

5.4 SPAE Analysis Errors

Nearly all of the Scratch projects were successfully analyzed using the SPAE tool. While most of these projects failed from an encoding issue or a timeout because of a particularly large music file or image file, there were two special cases that resulted in the tool crashing entirely.

The first project, with a project ID of 83710710, was based on the Quest Game project, the 7th starter project for the Game Design curriculum in CS First. This project involves a user interacting with a surrounding environment and completing a quest involving finding specific artifacts or foods. Because of the images used in this project, the download sizes were often larger than normal. However, once the project was downloaded without timing out or failing from encoding, two red blocks were noticed that had not been seen in any previous Scratch projects. These red blocks, as seen in Figure 5.5, were revealed to be obsolete blocks supported by an earlier version of Scratch. Because the SCATT tool did not recognize these blocks, the tool entered into an infinite loop, causing the SPAE tool to fail. However, once these blocks were removed from the project, the SPAE tool was able to analyze the project successfully.

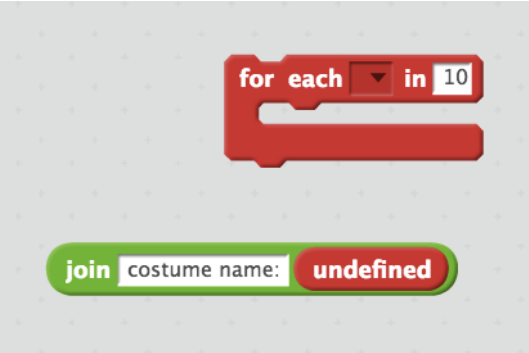

Figure 5.5: Two red “obsolete” blocks which cause SPAE to fail

The other project, with a project ID of 109234200, is based on the first starter project for the Social Media curriculum in CS First. This project was interesting, as the project failed regardless of what was removed from the Scratch project. In fact, the project continued to fail when all elements were removed from the project, leaving only a blank backdrop remaining. Two theories were that the project file is corrupt or that the JSON file is formatted improperly. However, manual inspection showed no differences or anomalies proving these theories to be false. Upon further troubleshooting, the issue was pinpointed to be within the SCATT tool. However, SCATT does not fail or report anything to the log files. After much research and troubleshooting, the issue remains unresolved.

5.5 CS First Curriculum Analysis

Figure 5.6 shows the totals for all 802,156 processed CS First projects. According to the complete SPAE analysis, blocks in the *Looks*, *Control*, and *Event* categories make up most of the blocks used in all CS First Projects. The blocks-to-script ratio is approximately 11.43 blocks/script, scripts-to-project ratio is approximately 6.55 scripts/project, and the blocks-to-project ratio is approximately 76.39 blocks/project. Table 5.2 breaks down these results by CS First theme. Future work may indicate that educators can use these ratios as benchmarks for evaluating student work.

```
Total # of Projects:                    802191
Total # Successful Analyzed Projects:   802156

       Totals For Analyzed Projects

     Total Scripts:           5357529
     Total Sprites:           3118792
     Total Variables:         388465
     Total Lists:             14310
     Total Script Comments:   35896
     Total Sounds:            4801948
     Total Costumes:          14832691
     Total Control:           14603656
     Total Data:              2192590
     Total Events:            11712700
     Total Looks:             17259146
     Total More Blocks:       849374
     Total Motion Blocks:     7072392
     Total Operators:         2202092
     Total Pen Blocks:        719398
     Total Sensing Blocks:    3214726
     Total Sound Blocks:      1452286
```

Figure 5.6: Overall results from entire Scratch Project ID dataset

Table 5.2: Results based on CS First Curriculum

CS First Theme	# of Projects	# of Scripts	# of Sprites	# of Blocks	Blocks/ Script	Scripts/ Project	Blocks/ Project
Animation	40292	285947	143441	4023210	14.07	7.10	99.85
Art	140976	589006	364573	5718802	9.71	4.18	40.57
Fashion	49007	257960	131583	2439034	9.45	5.26	49.77
Friends	47575	201242	139003	2701538	13.42	4.23	56.78
Game	154263	1199632	687294	15202442	12.67	7.78	98.55
Music	92852	290036	191359	4444158	15.32	3.12	47.86
Social	56324	627467	467291	4303492	6.86	11.14	76.41
Sports	73013	367315	245567	5563988	15.15	5.03	76.20
Story	185688	1183884	549041	14971966	12.65	6.38	80.63

Chapter 6 - Conclusions and Future Work

6.1 SPAE Tool

The SPAE tool discussed in this thesis presents a simple web interface allowing users to submit a Scratch project ID or URL and receive a straightforward analysis of the project. The SPAE tool successfully analyzed over 99.9% of projects from a test suite of over 800,000 projects, which is a strong indication of tool reliability. Unlike related Scratch project analysis tools, SPAE provides a comparison with any remixed sources of a project. In addition, SPAE is successful at analyzing recent Scratch projects whereas related Scratch project analysis tools have fallen into disrepair and fail frequently. The SPAE tool was tested extensively on multiple software and hardware platforms to demonstrate its viability in different school environments. Upon configuration, SPAE runs as a web application that allows any user with Internet access to use the tool. The SPAE tool is also capable of storing large amounts of analyzed project data, proving to be a capable tool for educators to store student projects and user data for future use.

Over 800,000 Scratch projects submitted by students worldwide as part of the CS First curriculum were analyzed using SPAE. The cumulative analysis provides teachers with potential benchmarking information on the characteristics of student projects in terms of the number of scripts, the number of blocks per script, the number of blocks per project, etc.

6.2 Future Work

There are many enhancements that can be made to the SPAE tool. While the success rate of the tool is currently very high, there are a few projects that fail when used with the SPAE tool. One solution is to simplify the download process from the Scratch API. This would require a change to the Python project downloader script to request just the JSON file rather than the full SB2 file. A modification would also need to be changed to the SCATT tool, where just the JSON

would be analyzed rather than unpacking the full SB2 file to retrieve the JSON for analysis. This would most likely speed up the analysis process, require less network activity, and reduce memory and CPU usage.

A better storage mechanism would also allow for increased readability of project results. MySQL would be an excellent resource to store and manage projects that have been analyzed using this tool. This likely would reduce the overall disk usage as well, as the current analyzed project results are being stored in TXT files. Storage into a MySQL database would also speed up search times of past projects, opening up the possibility to compare past projects against current projects with the same ID, allowing users to track progress of student work. This would also make tracking of total results more efficient and give rise to other new advancements which would be more difficult with the current storage schema. One example of this would be the ability to produce a class or group wide report for teachers and educators to keep track of a student's progress through their Scratch project ID. This would allow educators to quickly view a the progress of an entire class at the click of a button, allowing educators to make adjustments to coursework as deemed necessary without the need of manually analyzing every project.

The design of SPAE enables expansion of existing work as well as extension into new analysis areas. An example of an expansion is to have SPAE report on new SCATT analyses such as recognizing duplicate scripts or dead blocks (as described by Hairball). An example of an extension is to have SPAE perform analysis itself on the SCATT report to provide a categorization of block usage patterns similar to Dr. Scratch's "flow control," or "logic" categories. It would also be interesting to have teachers be able to define their own rubric for block usage patterns.

Another area of potential performance improvement involves making SPAE a multithreaded application. In this way, one project can be analyzed by SCATT while another is being downloaded. This becomes even more important if teachers are working on an entire class of student projects.

Bibliography

[1] James Adams. 2016. Eben Upton Talks Raspberry Pi 3. (February 2016). Retrieved October 17, 2018 from www.raspberrypi.org/magpi/pi-3-interview/. [1]

[2] The Apache HTTP Server Project. 2018. About the Apache HTTP Server Project. Retrieved November 19, 2018 from httpd.apache.org/ABOUT_APACHE.html. [2]

[3] Elaine Ashton. 2001. The Timeline of Perl and its Culture. Retrieved November 5, 2018 from history.perl.org/PerlTimeline.html. [3]

[4] Jess Bidgood and Jeremy Merrill. 2017. As Computer Coding Classes Swell, So Does Cheating. (May 2017). Retrieved October 16, 2018 from www.nytimes.com/2017/05/29/us/computer-science-cheating.html. [4]

[5] Jess Bidgood and Jeremy Merrill. 2017. Why Cheating Is on the Rise in Computer Science Classes. (August 2017). Retrieved October 16, 2018 from www.eab.com/daily-briefing/2017/08/01/3-reasons-cheating-is-on-the-rise-in-comp-sci. [5]

[6] Noel Bitner and Joe Bitner. 2002. Integrating Technology into the Classroom: Eight Keys to Success. Journal of Technology and Teacher Education, 10(1), 95-100. Norfolk, VA: Society for Information Technology & Teacher Education. Retrieved November 16, 2018 from https://www.learntechlib.org/primary/p/9304/. [6]

[7] Bryce Boe, Charlotte Hill, Michelle Len, Greg Dreschler, Phillip Conrad, and Diana Franklin. 2013. Hairball: Lint-inspired Static Analysis of Scratch Projects. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*. ACM Press, New York, NY, 215-220. DOI: https://doi.org/10.1145/2445196.2445265. [7]

[8] Caldwell County Schools. 2018. Hour of Code Highlights Computer Science for Middle School Students. Retrieved November 19, 2018 from www.caldwellschools.com/site/default.aspx?PageType=3&DomainID=8&ModuleInstanceID=18275&ViewID=6446EE88-D30C-497E-9316-3F8874B3E108&RenderLoc=0&FlexDataID=17774&PageID=9. [8]

[9] Martin Campbell-Kelly. 1998. Programming the EDSAC: Early Programming Activity at the University of Cambridge. *Annals of the History of Computing*, 20, 4 (October 1998), 46-67. DOI: https://doi.org/10.1109/85.728229. [9]

[10] Classic Reload. 2017. Treasure Mathstorm. (May 2017). Retrieved October 21, 2018 from https://www.classicreload.com/treasure-mathstorm.html. [10]

[11] Code.org. 2018. Computer Science Education Week. Retrieved October 18, 2018 from www.csedweek.org/. [11]

[12] Code.org. 2018. What Will You Create? Retrieved November 19, 2018 from www.code.org/. [12]

[13] Dr Scratch. 2014. Analyze Your Scratch Projects. Retrieved November 1, 2018 from www.drscratch.org/. [13]

[14] The Finch. 2017. Finch Robot. Retrieved November 19, 2018 from www.finchrobot.com/software/scratch. [14]

[15] Github – JaysGitLab. 2018. JaysGitLab/Cs-5666-Scatt-Gp. Retrieved November 1. 2018 from www.github.com/JaysGitLab/cs-5666-scatt-gp. [15]

[16] Github – Jemole. 2018. Jemole/DrScratch. Retrieved November 10, 2018 from www.github.com/jemole/drScratch/. [16]

[17] Github – PolyEdge. 2018. PolyEdge/Project-Downloader. Retrieved November 1, 2018 from www.github.com/PolyEdge/project-downloader. [17]

[18] GitHub – Tjvr. 2017. Tjvr/Kurt. Retrieved November 1, 2018 from www.github.com/tjvr/kurt. [18]

[19] Github - ucsb-cs-education. 2018. Ucsb-Cs-Education/Hairball. Retrieved November 10, 2018 from www.github.com/ucsb-cs-education/hairball. [19]

[20] Github - ucsb-cs-education. 2018. Contributors to ucsb-cs-education/hairball. Retrieved November 10, 2018 from https://github.com/ucsb-cs-education/hairball/graphs/contributors?from=2012-06-05&to=2018-11-10&type=c. [20]

[21] Google. 2018. Teach Computer Science & Coding to Kids - CS First. Retrieved October 19, 2018 from www.csfirst.withgoogle.com/en/home. [21]

[22] Government of Western Australia, Department of Education and Training. 2018. Effective Teaching: An Initiative of the Director General's Classroom First Strategy. Retrieved October 15, 2018 from https://www.education.wa.edu.au/documents/43634987/44524721/Effective+Teaching.pdf/5dcc8207-6057-3361-ade8-cf85e5a2c1ab. [22]

[23] Hour of Code. 2018. The Hour of Code Is Coming. What Will You Create? Retrieved October 19, 2018 from www.hourofcode.com/us.

[24] Bob Johnstone. 2003. *Never Mind the Laptops: Kids, Computers, and the Transformation of Learning.* iUniverse Inc., Lincoln, NE. [24]

[25] KnownHost. 2016. 99.9% Uptime - What Is It and What Does It Mean? (October 2016). Retrieved November 19, 2018 from https://www.knownhost.com/blog/99-9-uptime-guarantee/. [25]

[26] Kyle Media LLC. 2017. Apple MacBook Core 2 Duo 2.4 13" (White-08) Specs. (May 2017). Retrieved November 4, 2018 from www.everymac.com/systems/apple/macbook/specs/macbook-core-2-duo-2.4-white-13-early-2008-penryn-specs.html. [26]

[27] Kwok Wing Lai. 2008. ICT Supporting the Learning Process: The Premise, Reality, and Promise. In *International Handbook of Information Technology in Primary and Secondary Education*. Springer International Handbook of Information Technology in Primary and Secondary Education, Vol. 20. Springer, Boston, MA, 215-230. DOI: https://doi.org/10.1007/978-0-387-73315-9_13. [27]

[28] Lego Education. 2018. WeDo 2.0 - Products - LEGO Education. Retrieved November 19, 2018 from www.education.lego.com/en-gb/product/wedo-2. [28]

[29] Linode, LLC. 2018. Linode Guides & Tutorials. Retrieved November 5, 2018 from www.linode.com/tos. [29]

[30] Linode, LLC. 2018. No Calculator Required. Retrieved November 3, 2018 from www.linode.com/pricing#all. [30]

[31] Logo Foundation. 2014. Scratch Day @ TC. Retrieved November 19, 2018 from el.media.mit.edu/logo-foundation/what_is_logo/index.html. [31]

[32] Galen McQuillen. 2018. Google CS First Review for Teachers. (August 2018). Retrieved October 19, 2018 from www.commonsense.org/education/website/google-cs-first.

[33] MECC. 2007. Number Munchers: MECC. Retrieved October 15, 2018 from https://archive.org/details/msdos_Number_Munchers_1990.

[34] MECC. 2007. Welcome to MECC on the Internet. Retrieved October 15, 2018 from https://web.archive.org/web/19970203052023/http://www.mecc.com:80/index.html.

[35] Jesus Moreno-León, Gregorio Robles, and Marcos Román-González. 2015. Dr. Scratch: Automatic Analysis of Scratch Projects to Assess and Foster Computational Thinking. *Revista de Educación a Distancia,* 46 (2015): 1-23.

[36] Jesús Moreno-León. 2014. Dr. Scratch - Flow. Retrieved November 1, 2018 from www.drscratch.org/learn/Flow/.

[37] Jesús Moreno-León. 2014. Dr. Scratch - Logic. Retrieved November 1, 2018 from www.drscratch.org/learn/Logic/.

[38] Jesús Moreno-León. 2014. Dr. Scratch - User Interactivity. Retrieved November 1, 2018 from www.drscratch.org/learn/User/.

[39] Mathieu Muratet, Patrice Torguet, Jean-Pierre Jessel, and Fabienne Viallet. 2009. Towards a Serious Game to Help Students Learn Computer Programming. *International Journal of Computer Games Technology*, 2009, Article 470590 (January 2009), 12 pages.

[40] William Hugh Murray. 2010. Cheating in Computer Science. *Ubiquity*, 2010, October, Article 2. DOI: https://doi.org/10.1145/1865907.1865908.

[41] Netcraft. 2018. August 2018 Web Server Survey. (September 2018). Retrieved November 19, 2018 from https://news.netcraft.com/archives/2018/08/24/august-2018-web-server-survey.html. [41]

[42] Tobias Oetiker. 2017. The Time Series Database. (February 2017). Retrieved November 19, 2018 from oss.oetiker.ch/rrdtool/. [42]

[43] P.org. 2017. Plagiarism: Facts & Stats. (June 2017). Retrieved October 16, 2018 from www.plagiarism.org/article/plagiarism-facts-and-stats. [43]

[44] Seymour Papert. 1980. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, New York, NY. [44]

[45] Josh Patoka. 2018. Looking for a Laptop? 20 Ways to Find (Almost) Free Laptops. (April 2018). Retrieved November 3, 2018 from www.moneypeach.com/get-a-free-laptop/. [45]

[46] Richard E. Pattis. 1995. *Karel the Robot: a Gentle Introduction to the Art of* Programming. (2nd. ed.). John Wiley and Sons Inc., New York, NY. [46]

[47] PicoBoard. 2010. PicoBoard - Sensor Board That Works with MIT's Scratch. Retrieved November 19, 2018 from www.picocricket.com/picoboard.html. [47]

[48] Python 2.7.15 Documentation. 2018. What's New in Python 2.7. (November 2018). Retrieved November 19, 2018 from docs.python.org/2/whatsnew/2.7.html. [48]

[49] Raspberry Pi Foundation. 2013. And Breathe... (March 2013). Retrieved October 17, 2018 from https://www.raspberrypi.org/blog/and-breathe/. [49]

[50] Raspberry Pi Foundation. Raspberry Pi 3 Model B. Retrieved October 17, 2018 www.raspberrypi.org/products/raspberry-pi-3-model-b/. [50]

[51] Raspberry Pi Foundation. Raspberry Pi - GPIO. Retrieved October 17, 2018 from www.raspberrypi.org/documentation/usage/gpio/README.md. [51]

[52] Raspberry Pi Foundation. Raspberry Pi - Scratch. Retrieved October 17 2018 from www.raspberrypi.org/documentation/usage/scratch/. [52]

[53] Raspberry Pi Foundation. Teach, Learn, and Make with Raspberry Pi. Retrieved October 17, 2018 from www.raspberrypi.org/. [53]

[54] Raspberry Pi Forums. 2017. Scratch 2.0: All-New Features for Your Raspberry Pi. (June 2017). Retrieved November 19, 2018 from www.raspberrypi.org/blog/scratch-2-raspberry-pi/. [54]

[55] Diane Schanzenbach. 2014. Does Class Size Matter? Retrieved October 20, 2018 from http://nepc.colorado.edu/publication/does-class-size-matter. [55]

[56] Scratch. 2018. Scratch - FAQ. Retrieved November 19, 2018 from scratch.mit.edu/info/faq. [56]

[57] Scratch. 2018. Scratch - Imagine, Program, Share. Retrieved October 18, 2018 from www.scratch.mit.edu/. [57]

[58] Scratch. 2018. Scratch - Scratch 3.0 FAQ. Retrieved October 18, 2018 from www.scratch.mit.edu/3faq. [58]

[59] Scratch. 2018. Scratch - Statistics. Retrieved October 18, 2018 from www.scratch.mit.edu/statistics. [59]

[60] Scratch Wiki. 2018. Kurt. (October 2018). Retrieved November 19, 2018 from en.scratch-wiki.info/wiki/Kurt. [60]

[61] ScratchEd. 2018. ScratchEd - Homepage. Retrieved October 18, 2018 from www.scratched.gse.harvard.edu/. [61]

[62] Scratch Wiki. 2018. Scratch API (1.4). (October 2018). Retrieved November 1, 2018 from www.en.scratch-wiki.info/wiki/Scratch_API_(1.4). [62]

[63] Scratch Wiki. 2018. Scratch API (2.0). (October 2018). Retrieved November 1, 2018 from www.en.scratch-wiki.info/wiki/Scratch_API_(2.0). [63]

[64] Scratch Wiki. 2018. Scratch Versions. (September 2018). Retrieved November 19, 2018 from en.scratch-wiki.info/wiki/Scratch_Versions. [64]

[65] Sharpened Productions. 2015. Scratch Project File. (January 2015). Retrieved December 19, 2018 from www.fileinfo.com/extension/scratch. [65]

[66] Statista. 2018. Desktop PCs Average Age in the US 2013-2022. Retrieved November 3, 2018 from www.statista.com/statistics/267465/average-desktop-pc-lifespan/. [66]

[67] Valerie Strauss. 2012. Survey: Teachers Work 53 Hours per Week on Average. (March 2012). Retrieved October 16, 2018 from www.washingtonpost.com/blogs/answer-sheet/post/survey-teachers-work-53-hours-per-week-on-average/2012/03/16/gIQAqGxYGS_blog.html. [67]

[68] TLC. 2018. Education and Learning Resources. Retrieved October 15, 2018 from https://www.hmhco.com/. [68]

[69] Carleton Washburne. 1953. Adjusting the Program to the Child. *Educational Leadership,* 11, 3, (December 1953), 139-140. [69]

[70] Brian Zink. 2002. Computer science pioneer Samuel D. Conte dies at 85. (July 2002). Retrieved November 12, 2018 from https://news.uns.purdue.edu/html3month/020701.Conte.death.html.

[71] Mike Zyda. 2005. From Visual Simulation to Virtual Reality to Games. *Computer*, 38, 9, (September 2005), 25-32. DOI: https://doi.org/10.1109/MC.2005.297.

Vita

Joseph O'Neill was born in 1990 in Winston-Salem, North Carolina to Michael O'Neill and Marketta Gouge. He graduated from the University of Tennessee with a Bachelor of Science in Mathematics in December 2012. Joseph taught Secondary Mathematics at Karns High in Knoxville for the 2013-14 school year. On August 2015, Joseph entered Appalachian State University to pursue a Masters in Computer Science. For his tenure, he worked as a graduate research assistant (GRA). GRA duties included supporting the COSMIC after school program, teaching and grading lab assignments, and helping maintain the Appalachian State ePortfolio system. While working with the ePortfolio system, he helped develop multiple automation programs still in use today within the system. Over the last year, Joseph has been working at the University Center for Atmospheric Research, helping with the maintenance, monitoring of systems, and development of software for the organization as a Software Engineer.