SAGE Analytics: Assessment of Computational Thinking Proficiency in SAGE and Scratch data

Timotius Kartawijaya

Final Report

January 19, 2018

Contents

1 Introduction **2**

2 Related Work **2**
2.1 Hairball and Dr. Scratch 2
2.2 PECT 3

3 Vision **4**

4 Implementation **5**
4.1 Code Repository 5
4.2 Scratch Analyzer Enhancements 5
4.3 R Environment Set-Up 5
4.3.1 Installing R and RStudio 6
4.3.2 Installing tidyverse 7
4.4 Mapping Evidence Variables 7
4.4.1 SAGE data 7
4.4.2 Wild data 9
4.5 Data Visualization 9
4.6 Statistical Results 12

5 Limitations and Assumptions **12**
5.1 Exclusion of Block Tree Structure in Assessment 12
5.2 Differing Block Names by Scratch Version 13

6 Future Work **14**
6.1 Mapping to Design Pattern Variables and CT Concepts 14
6.2 Utilizing Tree Structure for Enhanced SAGE Data Scoring 14
6.3 Clarification of Project ID 14
6.4 Integration to Scratch Analyzer 15

1 Introduction

Scratch is an educational tool that has proven to be effective in developing computational thinking (CT), equipping its users with the CT foundations that enable them to excel in formal computer science training [1]. Even so, Scratch is not perfect in developing programming skills. Meerbaum Salant, Armoni, and Ben-Ari found that Scratch users tend to develop habits of programming which are in contrast with accepted computer science practice. Most notably, students tend to implement scripts not based on carefully thought-out design but rushes to code based on available resources or ”blocks”, also known as programming by *bricolage* [8].

In response, the Social Addictive Gameful Engineering (SAGE) has been developed. As past research suggests, Scratch is most effective in developing transferable programming skills if combined with some guided instruction[6, 10, 7]. SAGE seeks to provide this guidance, providing a system which maximizes engagement and CT development by enhancing Scratch with features such as intelligent tutoring, gameful interfaces, and analytics.

This report describes the work done to provide evidence of the efficacy of SAGE in comparison with Scratch. This project focuses on the post-deployment stage of SAGE, providing the tools necessary to quantify the effectiveness of SAGE in developing computational thinking both for research and for teaching.

2 Related Work

This section describes previous research and work that aided and inspired this project.

2.1 Hairball and Dr. Scratch

Hairball is a tool developed for automatic static analyses of students' performance in a Scratch project [5]. Using Python plug-ins, the tool receives Scratch project files (.sb2) as input and produces analyses of the projects based on block presence. Hairball is used to detect the implementation accuracy of a predefined set of CT concepts, which in this version are only of initialization, broadcast and receive, say and sound synchronization, and animation.

Dr. Scratch is a web application that builds upon Hairball, expanding its capabilities by implementing a rubric which encompasses components of CT thinking. Additionally, it provides an interactive assessment student interface and conducted a more detailed field experiment to assess its performance [9]. Results from the study indicate that Dr. Scratch had positive results: higher CT scores and consequently increased coding skills.

In comparison with Hairball and Dr. Scratch, this project analyzes Scratch projects similarly, using block presence to automatically quantify CT proficiency. Differing from both projects, this project attempts to expand the set of CT concepts the students are assessed on. Additionally, the project seeks to incorporate design patterns in assessing CT thinking in addition to block presence.

2.2 PECT

The Progression of Early Computational Thinking (PECT) model is a framework for assessing computational thinking that incorporates both block presence and coding design patterns [11]. In this model, blocks are mapped to evidence variables, which is a set of concrete characteristics of code in Scratch that assess proficiency in computing categories (e.g. Looks, Sound, Motion). These evidence variables are then mapped to design pattern variables, which are a set of proficiencies that indicate common coding patterns in Scratch. Finally, these design pattern variables are used to inform whether the students retain computational thinking concepts, as proposed by the CSTA [2]. This model has been pilot-tested, with results consistent to expectations, where younger students grasp the more basic patterns of coding and older students grasping the more complex ones.

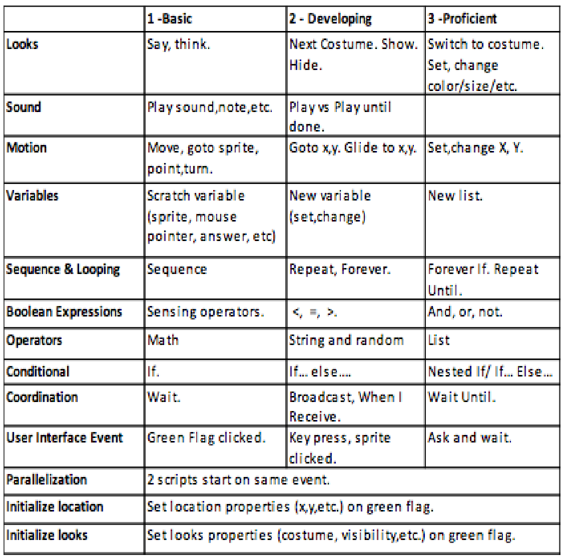
This project adopts and implements this model when automatically assessing student's CT proficiency. Block presence of each project are mapped automatically to evidence variable and design pattern scores, which informs CT proficiency.

3 Vision

This project aims to quantify the efficacy of SAGE in comparison with Scratch. When SAGE is deployed, a metric of success is needed to indicate how SAGE performs in developing CT among its students. For proper comparison, this metric must also be applicable to Scratch data (in this project called “Wild Data”), which is used as the control group.

The PECT model mentioned above is chosen as this metric. This project aims to implement this model. Using rubrics given by the model, blocks are automatically mapped to evidence variables, design patterns, then to CT proficiency. For example, the rubric for evidence variable mapping is shown in Figure 1. The model would be implemented on a per project basis both for SAGE and Wild data.

Furthermore, the project would need to produce data visualizations and descriptive statistics from the scoring results. These outputs would provide intuitive comparisons between SAGE and Scratch, providing evidence on the efficacy of SAGE.



	1 -Basic	2 - Developing	3 -Proficient
Looks	Say, think.	Next Costume. Show. Hide.	Switch to costume. Set, change color/size/etc.
Sound	Play sound,note,etc.	Play vs Play until done.	
Motion	Move, goto sprite, point,turn.	Goto x,y. Glide to x,y.	Set,change X, Y.
Variables	Scratch variable (sprite, mouse pointer, answer, etc)	New variable (set,change)	New list.
Sequence & Looping	Sequence	Repeat, Forever.	Forever If. Repeat Until.
Boolean Expressions	Sensing operators.	<, =, >.	And, or, not.
Operators	Math	String and random	List
Conditional	If.	If... else....	Nested If/ If... Else...
Coordination	Wait.	Broadcast, When I Receive.	Wait Until.
User Interface Event	Green Flag clicked.	Key press, sprite clicked.	Ask and wait.
Parallelization	2 scripts start on same event.		
Initialize location	Set location properties (x,y,etc.) on green flag.		
Initialize looks	Set looks properties (costume, visibility,etc.) on green flag.		

Figure 1: The rubric used to map blocks to evidence variable scores.

4 Implementation

4.1 Code Repository

The repository for this project can be found in **https://github.com/cu-sage/scratch-analyzer**, in the 'Development' branch.

4.2 Scratch Analyzer Enhancements

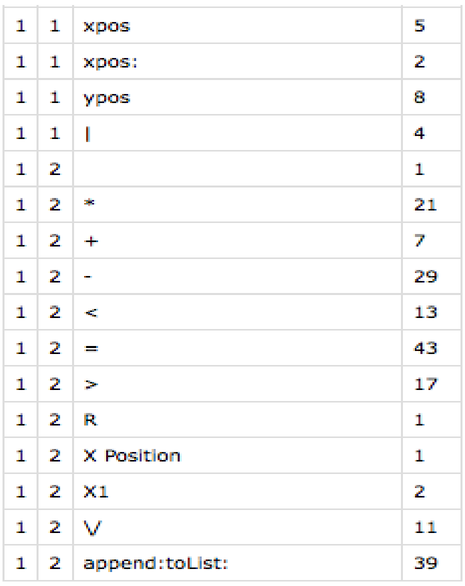
Scratch Analyzer is a Java tool that extracts Scratch project files (.sb2) to better formats for data mining and analysis (.csv) [3]. The tool has three components within it: Scratch Extractor, Scratch Dispatcher, and Scratch Traverser. First, Scratch Extractor extracts unnecessary components from the .sb2 files and creates a cleaner file in the form of a .se file. Second, Scratch Dispatcher uses these simplified .se files to output a lookup table and a .csv file containing the total number of blocks used by each student. Lastly, Scratch Traverser outputs a directory structure matching that of the Scratch Extractor output.

This project utilizes Scratch Analyzer to extract data from Scratch project files. Particularly, the output of Scratch Dispatcher is of interest, as the count of blocks of each project are needed for evidence variable scoring. However, the old version of Scratch Dispatcher only aggregates block count by student. For proper mapping, a per project output of blocks is needed. Therefore, Scratch Dispatcher was enhanced and now produces an additional .csv file called dispatch_perProject.csv, which displays the count of blocks by project ID and student ID (see Figure 2).

4.3 R Environment Set-Up

This project utilizes the R programming language, due to the author's familiarity with the language and its strength in statistical computing, data visualization, and dataset manipulation. Below are the steps to set up R on a Windows or Mac machine. More detailed instructions in installing R can be found at `https://cran.r-project.org`.

RStudio is an integrated development environment (IDE) for R which is helpful in installing packages, editing, and debugging R scripts. The IDE may not be necessary to



1	1	xpos	5
1	1	xpos:	2
1	1	ypos	8
1	1	|	4
1	2		1
1	2	*	21
1	2	+	7
1	2	-	29
1	2	<	13
1	2	=	43
1	2	>	17
1	2	R	1
1	2	X Position	1
1	2	X1	2
1	2	V	11
1	2	append:toList:	39

Figure 2: Output of current Scratch Dispatcher, displaying count of blocks by project ID and student ID.

run scripts related to the project, but its installation is recommended. Steps to install the IDE are also provided below.

4.3.1 Installing R and RStudio

For Mac users:

1. Download and extract the following package: `https://cran.r-project.org/bin/macosx/R-3.4.3.pkg`.
2. Download and install XQuartz: `https://dl.bintray.com/xquartz/downloads/XQuartz-2.7.11.dmg`.
3. RStudio: Download and install: `https://download1.rstudio.org/RStudio-1.1.383.dmg`.

For Windows users:

1. Download and install: `https://cran.r-project.org/bin/windows/base/R-3.4.3-win.exe`.
2. RStudio: Download and install: `https://download1.rstudio.org/RStudio-1.1.383.exe`.

4.3.2 Installing tidyverse

Tidyverse is a collection of R packages used for data science. The packages that are mainly used in this project are dplyr, a package that specializes in data manipulation, and ggplot2, a package for data visualization. To install tidyverse, follow these steps:

1. Using command line, go to the pect_analysis directory in the main repository.
2. Type: R CMD INSTALL tidyverse_1.2.1.tar.gz and the package should install.
3. **An alternative way using RStudio**: Open RStudio, and in the console, execute the line: install.packages("tidyverse")

4.4 Mapping Evidence Variables

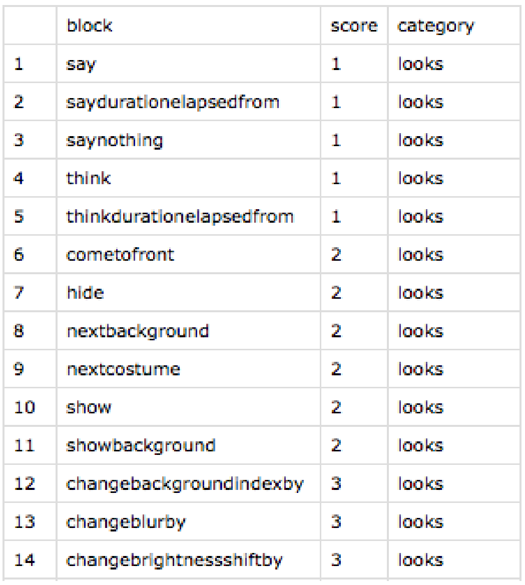
The mapping process of blocks to evidence variables are slightly different for SAGE data and Wild data. Both processes have a similar mapping process, but have differing input data formats and output results.

4.4.1 SAGE data

For SAGE data analysis, the files used are located in the pect_analysis directory. The main script used is called analysis.R, which automatically maps evidence variables, visualizes data, and provides descriptive statistics. To run the analyses, simply type in "RScript analysis.R" in the command line. Further instructions are included in a README file.

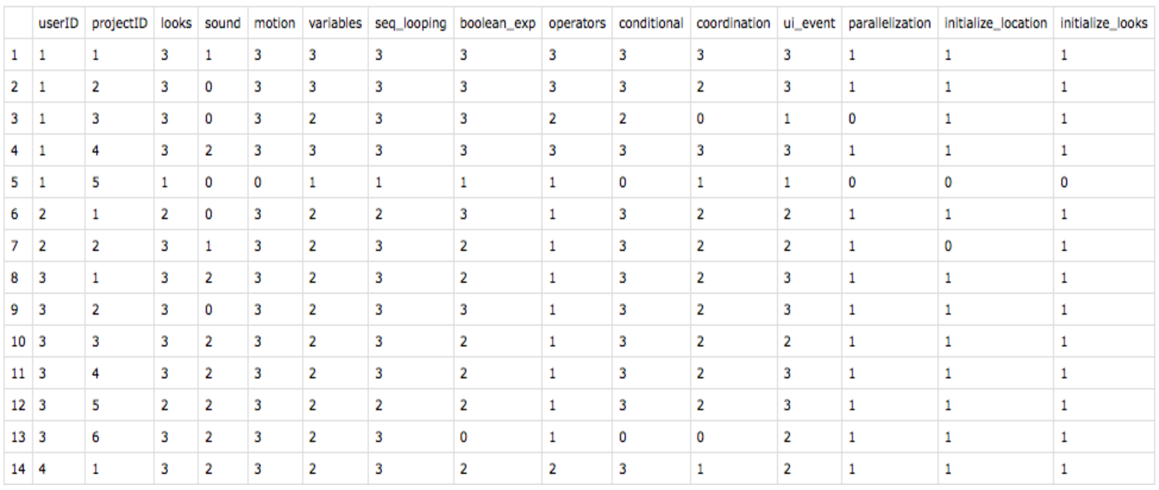
The input data for SAGE originates from the output of Scratch Dispatcher, as shown in Figure 2. A lookup table named ev_keys.csv is then created according to the PECT rubric (Figure 1), which maps each block type to a score and a category (Figure 3). This table can be manually altered if new block types arises or if there is revision in a block's category and score. Using the lookup table, the R script goes through each project, checks what blocks each project has, and gives the highest possible score for each category. For example, if a project has a "say" block and a "next costume" block, it would give a score of 2, since it fulfilled the criteria for the higher score. For parallelization and both initializations, the lookup table is not used, since it checks multiple blocks. In parallelization, a perfect score of 1 is given if two green flag blocks are present. For both initialization categories, a perfect score of 1 is given if a green flag and a property block are present.

The results of this scoring process is stored in a CSV file called ev_results.csv, as shown in Figure 4.



	block	score	category
1	say	1	looks
2	saydurationelapsedfrom	1	looks
3	saynothing	1	looks
4	think	1	looks
5	thinkdurationelapsedfrom	1	looks
6	cometofront	2	looks
7	hide	2	looks
8	nextbackground	2	looks
9	nextcostume	2	looks
10	show	2	looks
11	showbackground	2	looks
12	changebackgroundindexby	3	looks
13	changeblurby	3	looks
14	changebrightnessshiftby	3	looks

Figure 3: Lookup table that maps block name to evidence variable score along with its category.



	userID	projectID	looks	sound	motion	variables	seq_looping	boolean_exp	operators	conditional	coordination	ui_event	parallelization	initialize_location	initialize_looks
1	1	1	3	1	3	3	3	3	3	3	3	3	1	1	1
2	1	2	3	0	3	3	3	3	3	3	2	3	1	1	1
3	1	3	3	0	3	2	3	3	2	2	0	1	0	1	1
4	1	4	3	2	3	3	3	3	3	3	3	3	1	1	1
5	1	5	1	0	0	1	1	1	1	0	1	1	0	0	0
6	2	1	2	0	3	2	2	3	1	3	2	2	1	1	1
7	2	2	3	1	3	2	3	2	1	3	2	2	1	0	1
8	3	1	3	2	3	2	3	2	1	3	2	3	1	1	1
9	3	2	3	0	3	2	3	3	1	3	2	3	1	1	1
10	3	3	3	2	3	2	3	2	1	3	2	2	1	1	1
11	3	4	3	2	3	2	3	2	1	3	2	3	1	1	1
12	3	5	2	2	3	2	2	2	1	3	2	3	1	1	1
13	3	6	3	2	3	2	3	0	1	0	0	2	1	1	1
14	4	1	3	2	3	2	3	2	2	3	1	2	1	1	1

Figure 4: Output of analysis.R, containing the evidence variable scores for each student's projects.

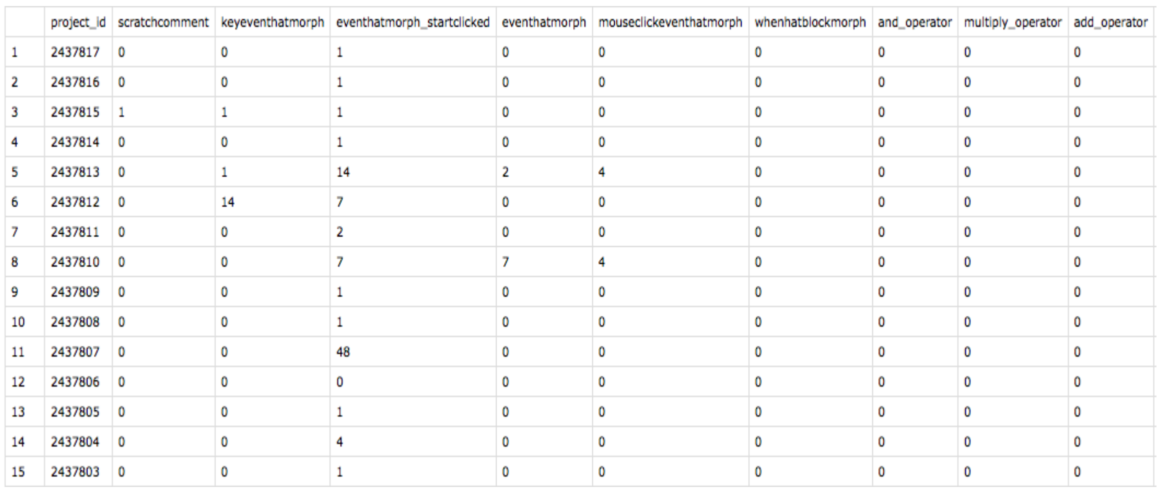
4.4.2 Wild data

For Wild data, the files used are in the wild_analysis directory. The main script used is called computeWild.R, which has the same features and running procedures to analysis.R.

The input data comes from the Scratch 1.0 website provided by colleagues from MIT, which contains data from over 9,000,000 projects. The data is in the form of a .csv file and contains project ID, block names, and the number of blocks that exist in each project (Figure 5).

The scoring process is then identical to SAGE data, using a lookup table to map block types to evidence variable scores and using a special process for parallelization and initialization.

The results of the process is stored in a CSV file called ev_output.csv, which is formatted slightly differently in comparison with SAGE results (see Figure 6).

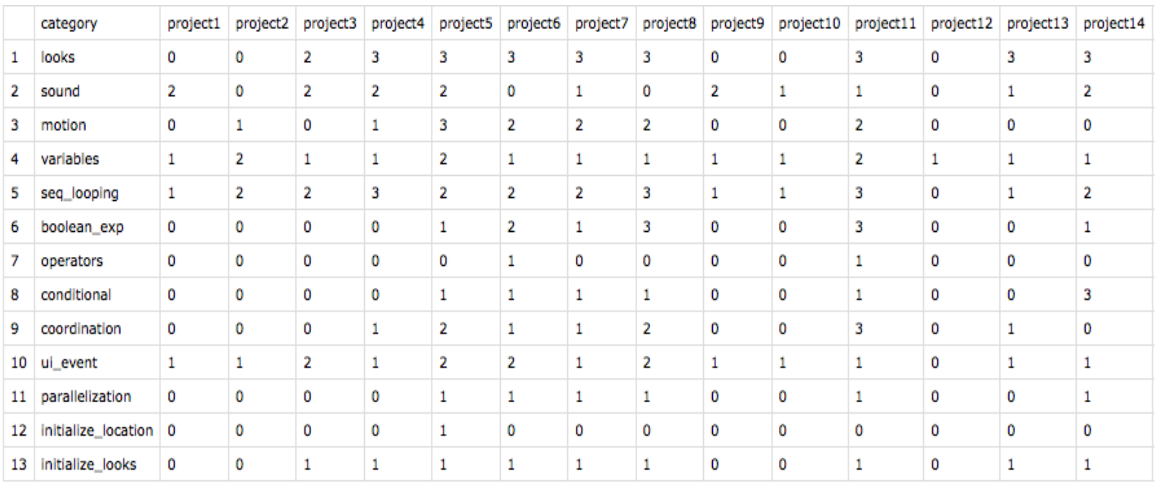


	project_id	scratchcomment	keyeventhatmorph	eventhatmorph_startclicked	eventhatmorph	mouseclickeventhatmorph	whenhatblockmorph	and_operator	multiply_operator	add_operator
1	2437817	0	0	1	0	0	0	0	0	0
2	2437816	0	0	1	0	0	0	0	0	0
3	2437815	1	1	1	0	0	0	0	0	0
4	2437814	0	0	1	0	0	0	0	0	0
5	2437813	0	1	14	2	4	0	0	0	0
6	2437812	0	14	7	0	0	0	0	0	0
7	2437811	0	0	2	0	0	0	0	0	0
8	2437810	0	0	7	7	4	0	0	0	0
9	2437809	0	0	1	0	0	0	0	0	0
10	2437808	0	0	1	0	0	0	0	0	0
11	2437807	0	0	48	0	0	0	0	0	0
12	2437806	0	0	0	0	0	0	0	0	0
13	2437805	0	0	1	0	0	0	0	0	0
14	2437804	0	0	4	0	0	0	0	0	0
15	2437803	0	0	1	0	0	0	0	0	0

Figure 5: Input data for Wild data analysis. Data contains information from over 9,000,000 projects from the Scratch website, including project ID, block names, and the number of blocks contained in each project.

4.5 Data Visualization

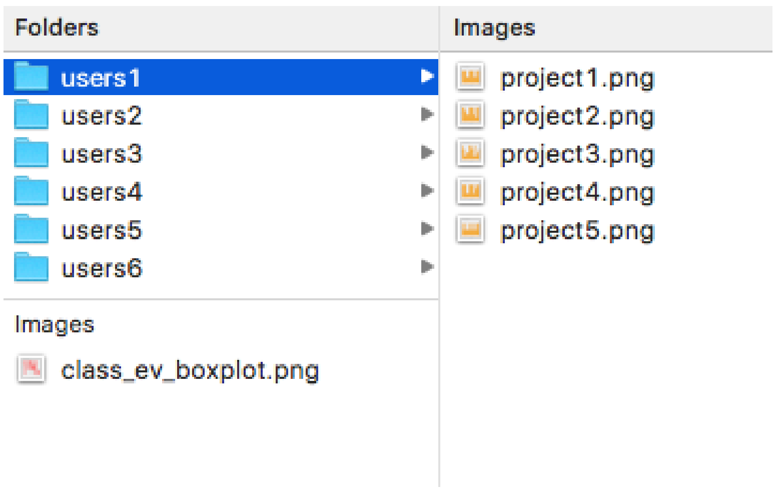
Visualizations are provided automatically as evidence variables are mapped. For SAGE data, bar plots of project scores are organized by student ID subdirectories, as shown in Figure 7. The bar plots are in the form of Figure 8. Furthermore, a box plot representing the distribution of evidence variable scores of the entire class is also provided, in purpose



	category	project1	project2	project3	project4	project5	project6	project7	project8	project9	project10	project11	project12	project13	project14
1	looks	0	0	2	3	3	3	3	3	0	0	3	0	3	3
2	sound	2	0	2	2	2	0	1	0	2	1	1	0	1	2
3	motion	0	1	0	1	3	2	2	2	0	0	2	0	0	0
4	variables	1	2	1	1	2	1	1	1	1	1	2	1	1	1
5	seq_looping	1	2	2	3	2	2	2	3	1	1	3	0	1	2
6	boolean_exp	0	0	0	0	1	2	1	3	0	0	3	0	0	1
7	operators	0	0	0	0	0	1	0	0	0	0	1	0	0	0
8	conditional	0	0	0	0	1	1	1	1	0	0	1	0	0	3
9	coordination	0	0	0	1	2	1	1	2	0	0	3	0	1	0
10	ui_event	1	1	2	1	2	2	1	2	1	1	1	0	1	1
11	parallelization	0	0	0	0	1	1	1	1	0	0	1	0	0	1
12	initialize_location	0	0	0	0	1	0	0	0	0	0	0	0	0	0
13	initialize_looks	0	0	1	1	1	1	1	1	0	0	1	0	1	1

Figure 6: Output results for Wild data analysis, containing evidence variable scores of each project.

of aiding teachers and researchers in determining how the entire class is performing (see Figure 9).



Folders
users1
users2
users3
users4
users5
users6

Images
class_ev_boxplot.png

Images
project1.png
project2.png
project3.png
project4.png
project5.png

Figure 7: Directory structure of the graphical output for each student's evidence variable scores. Each student folder contains bar plots of their project scores.

For Wild data, only a box plot of the entire data set is produced, showing the distribution of evidence variables scores as a whole. The box plot format is identical to that of SAGE data (Figure 9). Individual bar plots for each project are not produced due to the sheer number of projects.

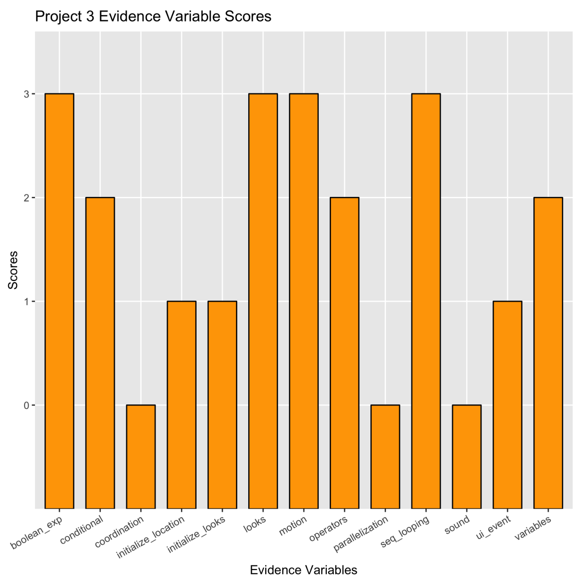


Figure 8: Bar plot of a student's project evidence variable scores.

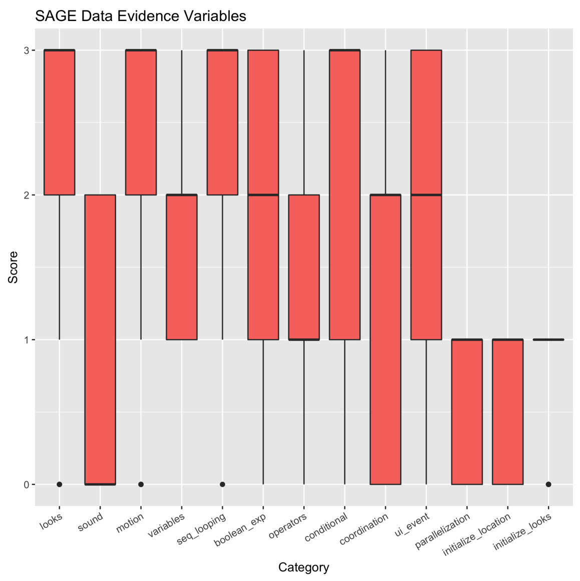
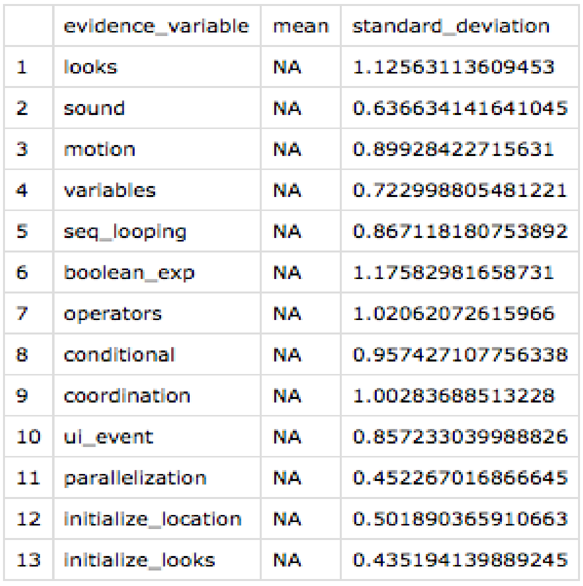


Figure 9: Box plot showing the distribution of evidence variable scores of the whole class.

4.6 Statistical Results

Descriptive statistics of aggregate evidence variable scores are automatically provided in the scoring process. The resulting CSV file is shown in Figure 10, by the name of ev_statistics. These statistics are produced for intuitive inferences in a class' CT proficiency.



	evidence_variable	mean	standard_deviation
1	looks	NA	1.12563113609453
2	sound	NA	0.636634141641045
3	motion	NA	0.89928422715631
4	variables	NA	0.722998805481221
5	seq_looping	NA	0.867118180753892
6	boolean_exp	NA	1.17582981658731
7	operators	NA	1.02062072615966
8	conditional	NA	0.957427107756338
9	coordination	NA	1.00283688513228
10	ui_event	NA	0.857233039988826
11	parallelization	NA	0.452267016866645
12	initialize_location	NA	0.501890365910663
13	initialize_looks	NA	0.435194139889245

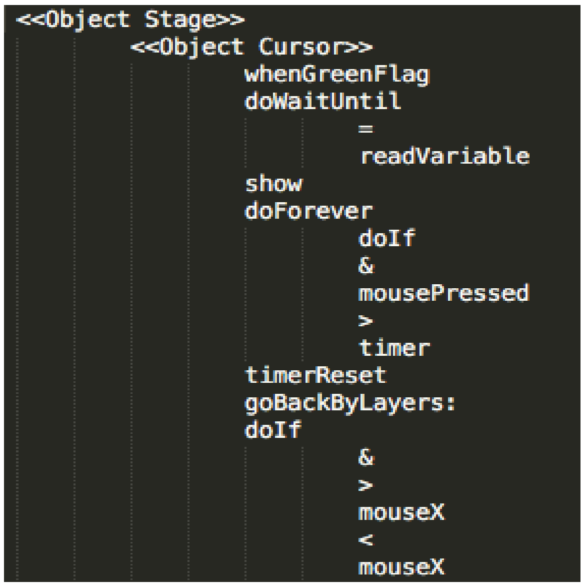
Figure 10: Mean and standard deviation statistics of the evidence variable scores from the entire class.

5 Limitations and Assumptions

The process above was done with some limitations and assumptions, due to the format of the data sources as well as the given version of Scratch.

5.1 Exclusion of Block Tree Structure in Assessment

The scoring process for both SAGE and Wild data assumes that the blocks present in a project are placed in a cohesive manner. Consequently, there are always a possibility that the blocks present in the projects are placed randomly without any logical sequence. The process ignores any kind of block structure due to the format of the Wild data (see Figure 5). The Wild data contains only information regarding how many blocks are present in a given project, but none regarding how they are connected with each other. For SAGE data, Scratch Extractor provides output that maintains block tree structure (see Figure 11). However, this structure is intentionally not included in the scoring process to maintain an identical assessment process with Wild data, so that proper comparison can be made. Therefore, some assessments, such as those concerning nested if-else blocks, may not be accurate .



```
<<Object Stage>>
        <<Object Cursor>>
                whenGreenFlag
                doWaitUntil
                        =
                        readVariable
                show
                doForever
                        doIf
                        &
                        mousePressed
                        >
                        timer
                timerReset
                goBackByLayers:
                doIf
                        &
                        >
                        mouseX
                        <
                        mouseX
```

Figure 11: Output of Scratch Extractor, which is a .se file that contains the type of blocks present in a project along with its tree structure.

5.2 Differing Block Names by Scratch Version

This limitation only affects the scoring process for SAGE data. Each version of Scratch has different names for each block. For example, in Scratch 1.0, the addition operator block has the name of "add_operator," while in Scratch 2.0 its variable name is "+". Furthermore, there are differences in syntax. In Scratch 1.0, the underscore symbol is used to represent spaces, while in Scratch 2.0, the colon symbol replaces the underscore symbol. These differences in block names directly impacts the lookup table, eventually causing difficulty when projects are created with a newer version of Scratch.

Some work has been done to deal with these differences. First, the scoring process ignores all non-numeric and non-character symbols. Second, there functionality in the

R script that takes two CSV files, one containing all the block names in the old version of Scratch (blockListWild.csv) and one with block names of the current version used in SAGE (blockListSAGE.csv), and outputs a CSV file containing block names that are exclusive to the newer version (notInclude.csv). The lookup table can then be manually revised to include the newer block names.

6 Future Work

6.1 Mapping to Design Pattern Variables and CT Concepts

Due to time constraints, this project was only able to map and visualize evidence variables scores for both SAGE and Wild data. Mapping to Design Pattern Variables then CT thinking will be necessary to provide better assessment of students proficiency in programming. Even so, the code that generates data visualization and descriptive statistics can be easily transferable when mapping Design Pattern and CT concepts.

6.2 Utilizing Tree Structure for Enhanced SAGE Data Scoring

In future projects, SAGE data assessment can be enhanced by including block tree structure. More accurate inferences to categories such as Parallelization can be made, providing better CT scoring. This enhanced assessment can aid teachers in determining the proficiency of students and overall proficiency of a class. Scratch Traverser can be utilized in this future work, as it is able to perform functions that leverages the hierarchical structure of the .se files, preserved by Scratch Extractor.

6.3 Clarification of Project ID

The current Scratch Dispatcher organizes block count by project ID and student ID but does not give a clear distinction on which project ID is associated with which project name. Enhancements could be added to Scratch Dispatcher to create a lookup table that determines which pairs project ID with their respective names.

6.4 Integration to Scratch Analyzer

Currently, the scoring process still requires manual execution via the command line. This process can be streamlined further if integrated to Scratch Analyzer. Using Java to R API such as the JRI, the scoring process can be automated within Scratch Dispatcher for easier use.

References

[1] Michal Armoni, Orni Meerbaum-Salant, and Mordechai Ben-Ari. From scratch to "real" programming. *ACM Transactions on Computing Education (TOCE)*, 14(4):25, 2015.

[2] Computer Science Teachers Association. Computational thinking. http://csta.acm.org/Curriculum/sub/CompThinking.html, 2012.

[3] Jeff Bender. Scratch analyzer: Transforming scratch projects into inputs fit for educational data mining and learning analytics. Columbia University, New York, NY, 2015.

[4] Jeff Bender. Tooling scratch: Designing a collaborative game-based learning system to infuse computational thinking within grade 6-8 curricula. Columbia University, New York, NY, 2015.

[5] Bryce Boe, Charlotte Hill, Michelle Len, Greg Dreschler, Phillip Conrad, and Diana Franklin. Hairball: Lint-inspired static analysis of scratch projects. In *Proceeding of the 44th ACM technical symposium on Computer science education*, pages 215–220. ACM, 2013.

[6] Ann L Brown and Joseph C Campione. *Guided discovery in a community of learners.* The MIT Press, 1994.

[7] Shuchi Grover, Roy Pea, and Stephen Cooper. Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education*, 25(2):199–237, 2015.

[8] Orni Meerbaum-Salant, Michal Armoni, and Mordechai Ben-Ari. Habits of programming in scratch. In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*, pages 168–172. ACM, 2011.

[9] Jesús Moreno-León and Gregorio Robles. Dr. scratch: A web tool to automatically evaluate scratch projects. In *Proceedings of the workshop in primary and secondary computing education*, pages 132–133. ACM, 2015.

[10] Daniel L Schwartz, John D Bransford, David Sears, et al. Efficiency and innovation in transfer. *Transfer of learning from a modern multidisciplinary perspective*, pages 1–51, 2005.

[11] Linda Seiter and Brendan Foreman. Modeling the learning progressions of computational thinking of primary grade students. In *Proceedings of the ninth annual international ACM conference on International computing education research*, pages 59–66. ACM, 2013.