Formative SAGE Assessments: Proposal

Jairo Pava

COMS E6901, Section 14

Spring 2016

1.0 Introduction

The motivations behind this project are to automate teacher evaluation of SAGE (Bender, 2015) assignment submissions and improve students' understanding of computational thinking (Barr & Stephenson, 2011) concepts via personalized feedback.

Computational thinking is an approach to solving problems using concepts of abstraction, recursion, and iteration to process. Mastery of these concepts fosters critical thinking skills that are not just important in computer science but are crucial for the academic and personal development of young children through their adult lives. As such, this problem solving methodology can be transferred and applied across subjects in the classroom. Teaching computational thinking, however, has its challenges. Many teachers do not have the training to lead classroom discussions or prepare assignments to build this skill in their students. And even when teachers are motivated, sometimes it's a lot harder to keep students engaged especially since these skills can be difficult and frustrating to grasp at first.

Computational thinking is not computer science. But research in computer science, education, and the marriage between the two have led to software tools that address the challenges of teaching computational thinking in the classroom. The Lifelong Kindergarten Group at the Massachusetts Institute of Technology provides a great example of this with Scratch (Scratch - Imagine, Program, Share, 2016). Scratch is a programming language and online community where anyone can program and share interactive media such as stories, games, and animation with people from all over the world. The programming language is drag-and-drop based and programs, like the one shown in Figure 1, consist of two-dimensional, interactive animations. As of January 2016, more than 12 million projects have been shared on the Scratch

website by about 9 million registered users. The majority of those users are between the ages of 8 and 16.

Figure 1 Scratch Project

Millions of people are creating Scratch projects in a wide variety of settings including homes, schools, museums, libraries and community centers. It is easy to learn. It engages children. And it's a powerful learning tool.

SAGE extends Scratch by empowering teachers to play the role of game designers. This enables them to create games for students to play as they learn computational thinking concepts. Since games are fun and engaging for young students, SAGE leverages game-based learning techniques to instill intrinsic motivation for computational thinking at an early age. The teacher designs games and students play the game by building Scratch programs to solve objectives.

SAGE takes advantage of the collaborative nature of the online Scratch community and traditional classroom environments to encourage students to compete with each other while

playing the games and, most importantly, learning important concepts. However, it is ultimately up to the teacher to provide a final evaluation of the student's completion of the game.

This evaluation is time consuming and does not scale well. Evaluating student submissions of completed games is both subjective and objective. Objective criteria include whether the student completed specific tasks laid out in the game objectives. Subjective criteria, on the other hand, include the aggregate attention to details by students in their work that only a human can perceive but makes the clear difference between students that have understood a concept versus students that have *mastered* a concept. Evaluating subjective criteria is the most creative aspect of a teacher's work and meaningful feedback in this aspect will immensely benefit a student's progress.

The proposed project will therefore enable teachers to automate objective evaluations of student submissions. These evaluations will automatically provide real-time feedback to students as they work on their SAGE game. As a result, students will know how close they are to completing their game and whether they are on the right path, very much like popular games today track progress to completion. These automated evaluations will also free up teacher time to focus on the subjective evaluation of student submissions.

2.0 Related Work

Prior related work exists to provide students with real-time feedback as they complete programming assignments. Prior work also exists to perform automated analysis of Scratch project files. But no prior work was found to merge both those ideas, like it is proposed in this project. The sections below list the related work and discuss how those ideas can be extended.

2.1 Retina

Retina (Murphy, Kaiser, Loveland, & Hasan, 2009) is a tool that collects information about students as they complete computer programming assignments. This information includes time spent on the assignment, number of successful and failed compilations, and other related data. The course professor and student are then provided with reports by Retina based on the aggregation of that information across all students in the course. An example of this report is illustrated in Figure 2.

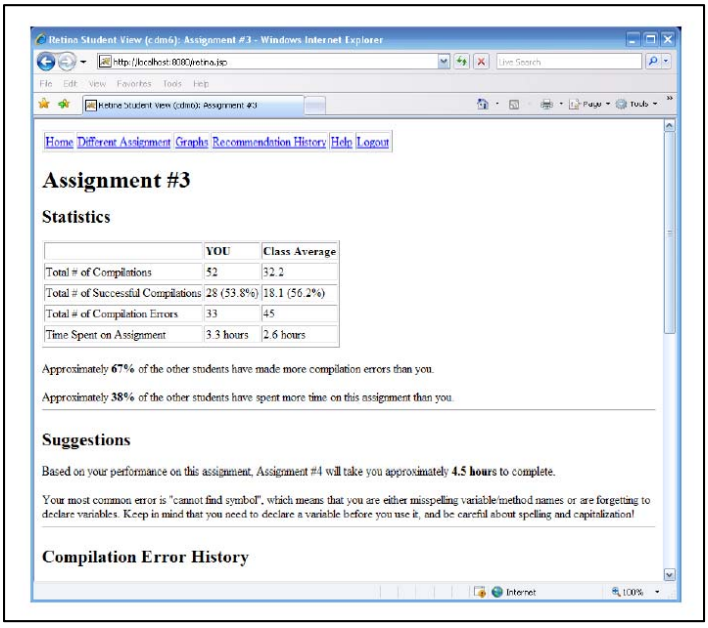

Figure 2 Retina Assignment Feedback

More importantly, Retina is able to generate personalized suggestions to students in real-time as they complete their programming assignments to guide them towards successful completion. Anecdotal evidence demonstrated Retina's utility in helping course instructors better plan class and one-on-one discussions with students.

The proposed project can benefit from the lessons learned by the team that worked on the Retina tool. For example, Retina uses rules to generate personalized feedback for students. These rules take into consideration the kinds of compilation errors that a student encounters, their rate

of occurrence, and how long it takes for a student to solve the errors. The proposed project can leverage the research behind these rules to generate feedback for students working on Scratch programs based on their compilation errors *and* automated evaluation results.

2.2 Hairball

Hairball (Boe, et al., 2013) is a static analysis tool for Scratch project files. The tool receives saved Scratch project files as input and outputs an analysis of the project. The analysis is performed by a collection of Hairball plugins that work independently to analyze the project on different criteria. Currently, Hairball has plugins that analyze the implementation of Scratch programs for competence in initialization, broadcast and receive, say and sound synchronization, and animation.

Hairball allows for the easy development of plugins using Python to add to the analysis of Scratch project files. Figure 3 illustrates an example Hairball plugin that keeps track of the number of unique blocks in a project.

```
class BlockCounts(HairballPlugin):
  def analyze(self, scratch):
    blocks = Counter()
    for block, _, _ in iter_blocks(scratch):
      blocks.update({block: 1})
    return blocks
```

Figure 3 Hairball Plugin Example

The proposed project can leverage the Python API exposed by Hairball to conduct its own analysis of student Scratch project files. The Python API provides facilities used to programmatically extract and retrieve meta-data about the blocks in a project file. This data can be used to perform the automated test evaluations as the students work on their assignment.

2.3 Dr. Scratch

Dr. Scratch (Moreno-Leon, Robles, & Roman-Gonzalez, 2015) is a web application where Scratch project files can be uploaded for automated analysis. After a project file is

analyzed, a score card is presented that indicates how well the project uses a variety of computational thinking concepts. An example of this score card is illustrated in Figure 4. Dr. Scratch has two goals: to support educators in the assessment of student projects and to encourage students to keep improving their programming skills. Feedback from Dr. Scratch enables students to understand that successful completion of an assignment includes more than just completing a set of tasks. Successful completion also includes mastering computational thinking concepts that improve their ability to complete similar assignments in the future even if they are not in the same domain.

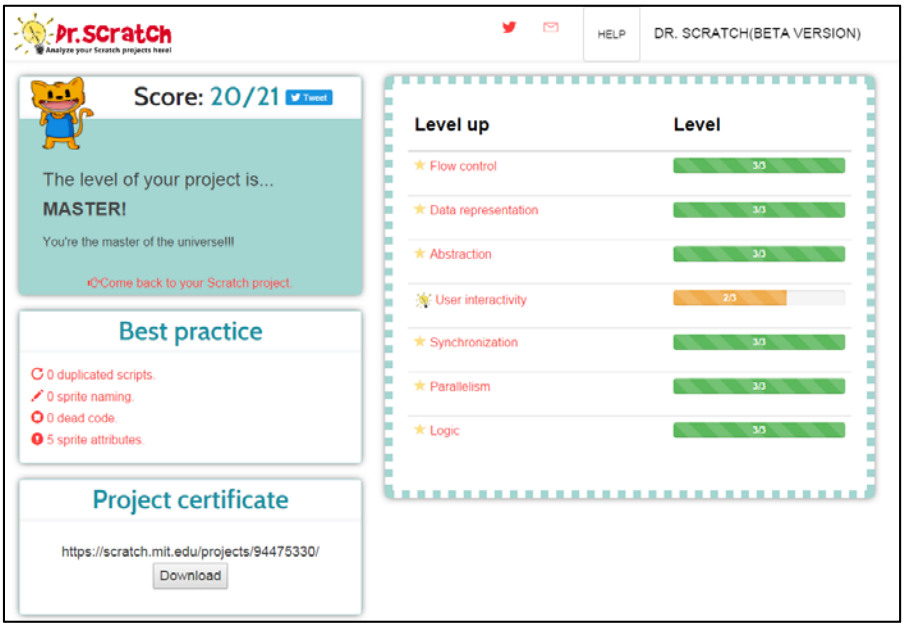

Figure 4 Dr. Scratch Project Evaluation

The proposed project can benefit from Dr. Scratch by taking inspiration from the approach of using a score card to present students with visual feedback on the progress they are making on the assignment. The score card is concise and clear. Students immediately see where they have mastered a concept and where they need to improve. Badges and scores in Dr. Scratch helps keep students engaged because it introduces an added incentive of social competitiveness

among classmates. And the progress bars allow students to quickly understand their progress towards completing their assignment the way it was really meant to be done by their teacher.

3.0 Proposal

The project will include the development of three independent components that together aim to accomplish the goals set forth by this proposal. The components are 1) a drag-and-drop based testing language for automatic evaluation of student SAGE assignments, 2) a test server where tests and scratch projects files are to be uploaded for automated evaluation, and 3) an updated Scratch editor which periodically saves and uploads project files to the test server and displays results to the student via a dashboard. The three components are illustrated in Figure 5 and are further described in the sub sections below.

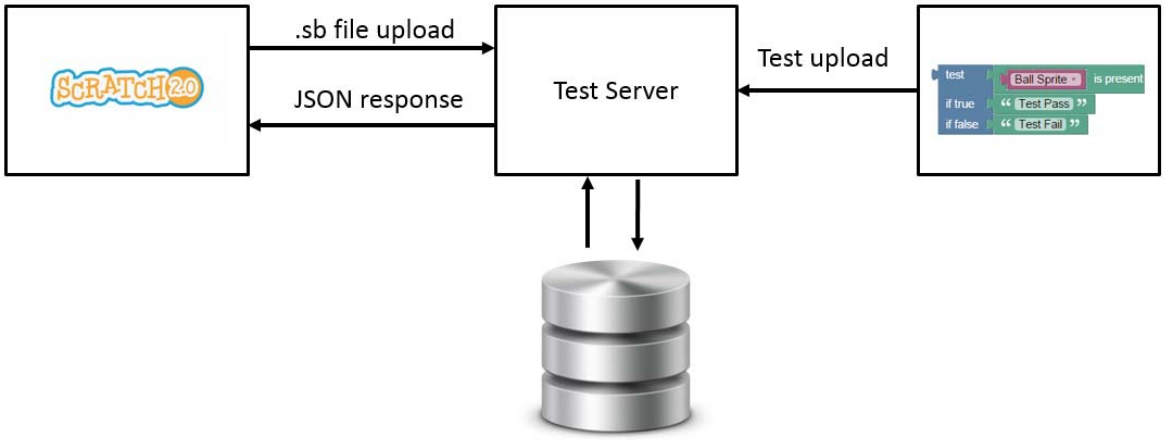

Figure 5 Proposed Architecture

3.1 Testing Language

Teachers will be provided with a block-based language, just like the one students use in Scratch, to build automated tests that evaluate student assignments. Blockly (Blockly, 2016), a library for building visual programming editors, will be used to implement the testing language. The tests that teachers will be able to write using this language are largely inspired from Harvard Graduate School of Education's Creative Computing Scratch Curriculum (Brennan, Balch, &

Chung, 2014). The curriculum defines a large variety of programming assignments that are recommended for middle school students. It also defines criteria that teachers can use to evaluate student assignments. These criteria include asserting the presence of sprites on the screen, response to user input, and interaction between sprites, among others. The proposed testing language will enable teachers to automate the evaluation of student submissions based on these criteria. Figure 6 illustrates an example of the language. Blocks are drag and dropped to connect and form test cases, just like students connect blocks to build games in Scratch.

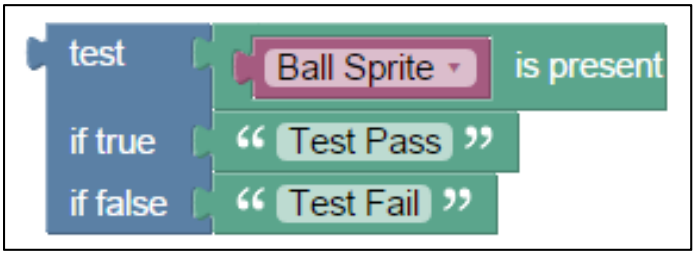

Figure 6 Proposed Testing Language

3.2 Test Server

The test server will expose an HTTP API that allows for the upload of tests and Scratch project files. When a project file is uploaded to the server, it will be evaluated using the automated tests and will return a detailed report. The server will leverage Hairball by running customized plugins that parse through the Scratch project files and perform the evaluations defined in the test language. A database will be used to persist the results of an assignment's evaluation over time to enable the ability to mine the data for further insight as part of future work.

3.3 Scratch Editor

The Scratch editor will be extended to periodically upload Scratch project files as a student works on an assignment. These files will be uploaded to the test server and evaluated based on the automated evaluations that have been created by the teacher. When the Scratch

editor receives the results from the test server, a new dashboard will display the results to the student. This process will occur in the background without disrupting the student's work on the assignment. The results on the dashboard will indicate to the student which evaluations have passed successfully and which evaluations are not passing. For those evaluations that are not passing, the student will need to continue to work. The dashboard will, in cases where it is appropriate, display suggestions that will help the student work towards successful evaluations.

4.0 Milestones

Table 1 outlines the various milestones and their estimated completion dates. Dates are also defined for midterm and final project deliverables.

Milestone	Date
Visual testing language	
Design	February 8, 2016
Implementation	February 22, 2016
Test server	
HTTP API	February 29, 2016
Execute tests against scratch project files	March 14, 2016
Scratch editor	
Periodic upload of projects to test server	April 4, 2016
Dashboard	April 18, 2016
Midterm written progress report	March 23, 2016
Final written report and oral presentation	May 11, 2016

Table 1 Proposed milestone completion dates

5.0 Future Work

The design of the proposed system will provide the flexibility to collect and analyze data for future study. For example, the test server will maintain a history of assignment evaluations.

This data may provide useful insights that can be mined to provide students with personalized recommendations. For example, consider a student that is currently struggling on an assignment. This student may exhibit a pattern of passed and failed automated evaluations that is similar to other students in the past who have worked on the same assignment. A recommendation can then be made to the student based on the path to success that past students have taken. The mining strategies, automated evaluations, and kinds of recommendations may be the subject of study for future work.

Another topic of interest for further study relates to the timing of the automatic evaluations. Automatic evaluations that are performed too frequently may overwhelm and frustrate a student that is working on an assignment. Evaluations that are not performed frequently enough may leave the student feeling isolated. The frequency with which data is collected will also impact the kinds of patterns and insights that may be mined from the data. Better understanding these tradeoffs will be an interesting future point of study.

References

(2016, January 25). Retrieved from Scratch - Imagine, Program, Share: https://scratch.mit.edu/

Barr, V., & Stephenson, C. (2011, March). Bringing Computational Thinking to K-12: What is Involved and What is the Role of the Computer Science Education Community? *ACM InRoads*, pp. 48-54.

Bender, J. (2015). *Developing a Collaborative Game-Based Learning System to Infuse Computational Thinking within Grade 6-8 Curricula.*

Boe, B., Hill, C., Len, M., Dreschler, G., Conrad, P., & Franklin, D. (2013). Hairball: Lint-inspired Static Analysis of Scratch Projects. *SIGCSE.* Denver: ACM.

Brennan, K., Balch, C., & Chung, M. (2014). *Creative Computing.* Harvard Graduate School of Education.

Moreno-Leon, J., Robles, G., & Roman-Gonzalez, M. (2015, September). Dr. Scratch: Automatic Analysis of Scratch Projects to Assess and Foster Computational Thinking. *Revista de Educación a Distancia*.

Murphy, C., Kaiser, G., Loveland, K., & Hasan, S. (2009). Retina: Helping Students and Instructors Based on Observed Programming Activities. *SIGCSE* (pp. 178-182). Chattanooga: ACM.