Foundations for Gamification of Formative SAGE Assessments

Final Report

COMS 6732 Section 01

December 21, 2016

Iris Zhang

Yu-Chun (Julie) Chien

Tsung-Jui (Ray) Tsai

Table of Contents

Abstract -- **Interest in computer science as an educational field has seen explosive growth in the past decade [16]. Thus researchers and educators have developed tools like Scratch to teach computational thinking concepts to a young audience [5]. Scratch allows for exploratory play but educators have made recent headway allowing it to become a more guided teaching tool. This paper outlines a semester-long project aimed at adding gamification elements to SAGE, a Scratch extension that empowers teachers to play the role of game designers. We outline motivation and implementation details on two gamification elements and an overall architecture improvement. The first is an addition to the Scratch Editor with point values to motivate students to explore with Scratch blocks. The second offers better extrinsic motivation towards passing assessment criteria on top of the SAGE Visual Language Assessment Editor. These feature additions are made possible by a back end server overhaul to make communication between different front end clients more flexible and better targeted. Overall the project lays strong foundations for a more guided student experience of Scratch project gameplay, more robust teacher tools for developing Scratch projects, and more structured future development work.**

1. Introduction

In 2002 MIT came out with their first iteration of Scratch. Since its release, Scratch has become popular among educators as a stepping stone towards teaching advanced computer programming concepts. It boasts 11 million unique visitors monthly, a testament to its ease of use and low entry barriers [5]. Scratch allows users to program interactive stories, games, and animations using a simple drag and drop interface [5]. Alone, Scratch encourages free form exploration but has very little in the way of guidance. As a result, educational researchers sought ways to improve for the platform as a tool for both students and teachers.

The Scratch Editor

Social Addictive Gameful Engineering (SAGE), first imagined by Jeff Bender in early 2015, sought to "tooling Scratch with a collaborative game-based learning system that catalyzes teachers and (grade 6-8) students to immerse in computational thinking" [19]. By the end of Spring of 2016, SAGE had been implemented with a tool for teachers to more closely guide student learning through Scratch [18]. While SAGE had improved vastly upon the open-ended tool that is Scratch, we sought to improve upon its goal of teaching computational thinking.

Why is teaching computational thinking desireable? Computational thinking is a way of thinking that allows for better problem solving and system design by drawing on fundamental Computer Science concepts (Wing, 2006). It involves using a logical and systematic engineer's approach to solve problems. When confronted with a problem, a computational thinker may first take a high-level assessment of the problem, pattern-match with other problems for possible solution

ideas, break the problem down into smaller subproblems, prototype solution ideas, test solutions in a systematic way, and iterate on the results to make improvements. We believe this type of analytical thinking is useful for everyone, not just programmers and engineers. Teaching students computational thinking from a young age may empower them with creative and logical problem solving skills that will be useful throughout their lives. The SAGE system brings students closer to this way of thinking by teaching students programming in a controlled environment with increased guidance than just open-ended play.

To improve upon teaching computational thinking through Scratch, we propose our project, Gamification of SAGE. Gamification of SAGE is partly a continuation and partly an expansion of Formative SAGE Assessments, adding game design elements and reward mechanisms to motivate students' learning through gamification, while overall vastly improving upon the server architecture and infrastructure. We propose that increased gamification of SAGE will

1) Drive learning by increasing the learner's motivation to engage with the system
2) Make SAGE more of a constructionist video game, increasing learning by increasing opportunity for exploration.
3) Lay the foundations of a better development framework with a new server endpoints

For one, making learning more game-like should hopefully increase students motivation to engage more with the system. Through elements such as points, rewards, trophies, or leaderboards, students will be motivated to increase their score, unlock new achievements, and compete with their peers [11]. While this doesn't guarantee increased learning outcomes, we hope that increased engagement with the system will lead towards actual learning.

Gamification will also make SAGE more constructionist. Constructionist video games are video games in which the primary gameplay consists of the construction of in-game artifacts . The principles of constructionist video games are 1) construction games must be sufficiently expressive, and 2) in-game goals should encourage exploration [3]. We believe that gamification will help make SAGE more of a constructionist video game by making the environment more "discovery rich" [1].

All of this would be enabled by an overhaul of the back end system which drives the SAGE server. The unnecessary pushing of chunks of that was currently being done by client-facing side of the code would be wrapped into resource endpoints that only deliver and serve structured data

that will actually be used by students and teachers [25]. This will also lay the foundations for future development work that will need less set up and is more structured.

This report describes the features we planned to add to the SAGE system in Fall 2016 and how they tie into our goals of improving teaching of computational thinking skills. It will formally describe the technical details of features added, as well as some documentation and the planning process along the way. The reader will understand the state of the project at the beginning and end of Fall 2016, some limitations we faced along the way, and proposals for future work.

2. Background and Relevant Works

We reviewed relevant work in three stages. The first part involved assessing the current stage of the SAGE project from a technical standpoint. To get started, each project member took a look at the existing Scratch tool available publicly for students to get a sense of the tool we aim to improve. Then we viewed existing Scratch projects to understand how educators may possibly use Scratch as a teaching tool [15]. This process helped guide our understanding of the motivation for the creation of SAGE, as well as form our first impressions of the technological possibilities. The paper "Formative SAGE Assessments" informed us of the state of the current SAGE project, outlining specifics of what technologies have been developed as well as suggestions for further expansions [18].

The later stage involved reading more theoretical papers about learning theory and computational thinking to inform our understanding of the foundations for SAGE. "Gamification in Education" described certain aspects of games that make them fun, in particular engaging players on a cognitive, emotional, and social level [22]. Gamification bridges the gap between game elements with education by providing students with "opportunity to experiment with rules, emotions, and social roles." The paper later goes on to give various examples of how mundane school tasks can be reframed in a fun way. In a midterm paper that formed the stage for SAGE, "Tooling Scratch" tied all this together by identifying "four learning principles which guide the formulation of design tactics fit for tooling Scratch with a collaborative game-based learning system that catalyzes teachers and students to immerse in computational thinking" [19]. Finally, "Micro Adaptivity" provided some insight into how to balance gaming and learning, while "Native Programmer Confusion Achievement" emphasized the critical need to correct confusion immediately after discovery [26] [27]. Both informed our design of Scratch enhancements which we hope addresses students' learning needs as soon as possible without distracting them from the overall "game" experience.

Finally, in the last technical stage of exploring related works, each of us were given a walkthrough of SAGE as well as how to set up Scratch running on our local environments. In addition, each of us had to separately become familiar with the codebase of the particular area we were working on. As the Scratch Editor codebase itself is tens of thousands lines of code, this was no trivial task [5].

This next section outlines individual related works that we used to support and inform our motivations for this project.

"Intelligent Tutoring and Games" (McNamara, Jackson, & Graesser, 2010)
Describes how adding game-based features to Intelligent Tutoring Systems(ITSs) may improve their effectiveness. An ITS is educational software that works like a teacher, providing feedback and guidance during the teaching. While students learn from these systems, they often dislike interacting with them. The paper proposes that adding video game features to ITSs may increase students' motivation to engage with the system and thus increase learning. It describes different types of motivation that may be increased by gamification, organizes these motivations into categories, and explains how gamification may improve each of these different types of motivation. SAGE is arguably an ITS, and we add game features to SAGE to increase students' desire to engage with the system. We hope that the game features we add will serve as extrinsic motivators that can serve as a springboard for interest in the topic and deeper learning.

"Computational Thinking" (Wing, 2006)
Defines Computational Thinking. Computational Thinking is a form of problem solving that draws upon Computer Science concepts. The paper gives several examples of Computational Thinking, such as transforming a difficult problem into a simpler one, separating concerns, and considering tradeoffs in resources. The paper outlines how computational thinking is advantageous for many disciplines, not just computer science. It stresses that Computational Thinking is not merely a mechanical skill, but an abstract and high-level one. SAGE's goal is to facilitate the learning of computational thinking by having students solve programming problems in a guided environment. Our new features will get us closer to this goal by increasing students' engagement with SAGE, which will drive learning.

"Constructionist Video Games" (Weintrop, Holbert, Wilensky, & Horn, 2012)
This paper introduces constructionist video games as a new class of constructionist learning environments. In a constructionist video game, the core gameplay consists of building things in-game. It empower players to be architects of their own learning. The paper provide a definition of constructionist video games, offering several examples of what is and isn't a constructionist video game. The paper also proposes two principles of successful constructionist games: sufficiently expressive building blocks and in-game goals that encourage exploration. The block points feature we add will encourage exploration by rewarding different solutions, making SAGE more constructionist.

3. Strategy and Planning

3.1 Gamification Features

As of Fall 2016, the SAGE project could be broken down into three main client-facing areas which we could make enhancements to the student and teacher Scratch experience:

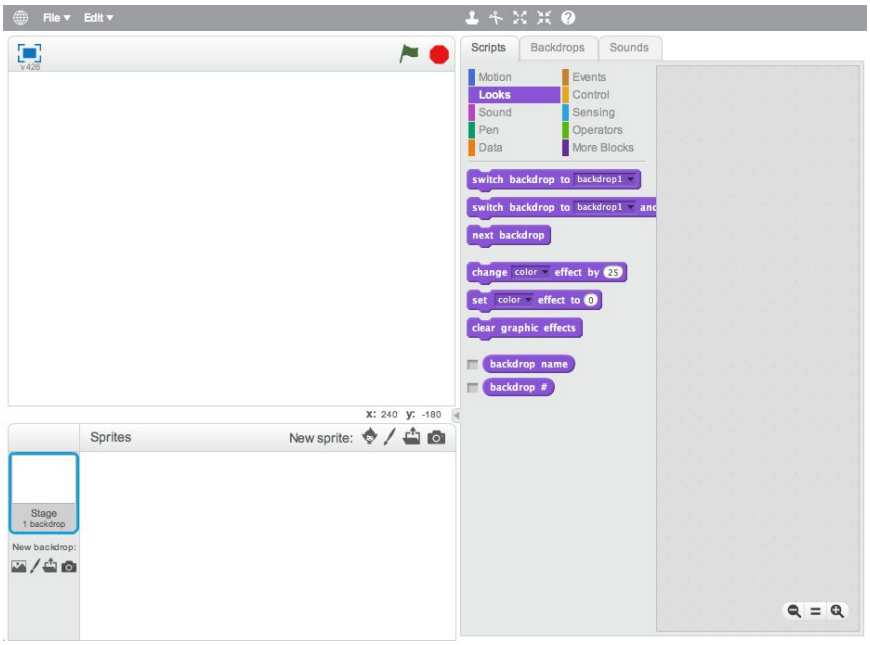


Figure 2: Student view of Scratch Editor

1. **Scratch Editor**: The Scratch Editor is where students build Scratch projects. SAGE made additions to the Scratch Editor that allows students to work on Scratch programming assignments created by teachers [13].
2. **Visual Assessment Language Editor:** Written using Blockly, SAGE added a whole grammar for describing Scratch blocks and Scratch elements that could be placed by a student while they "play" a Scratch game [23]. This allows teachers to test students for particular criteria when completing a Scratch-based assignment (figure on next page)
3. **Dashboard:** This is a browser-based prompt that opens in addition to the Scratch Editor from the student perspective. Once a student has loaded a SAGE Scratch assignment, the Dashboard will update with colors, visuals, and text messages without any prompt from

the student as they interact with the Scratch Editor. It was designed to feel unobtrusive and to run with minimal prompting [24]

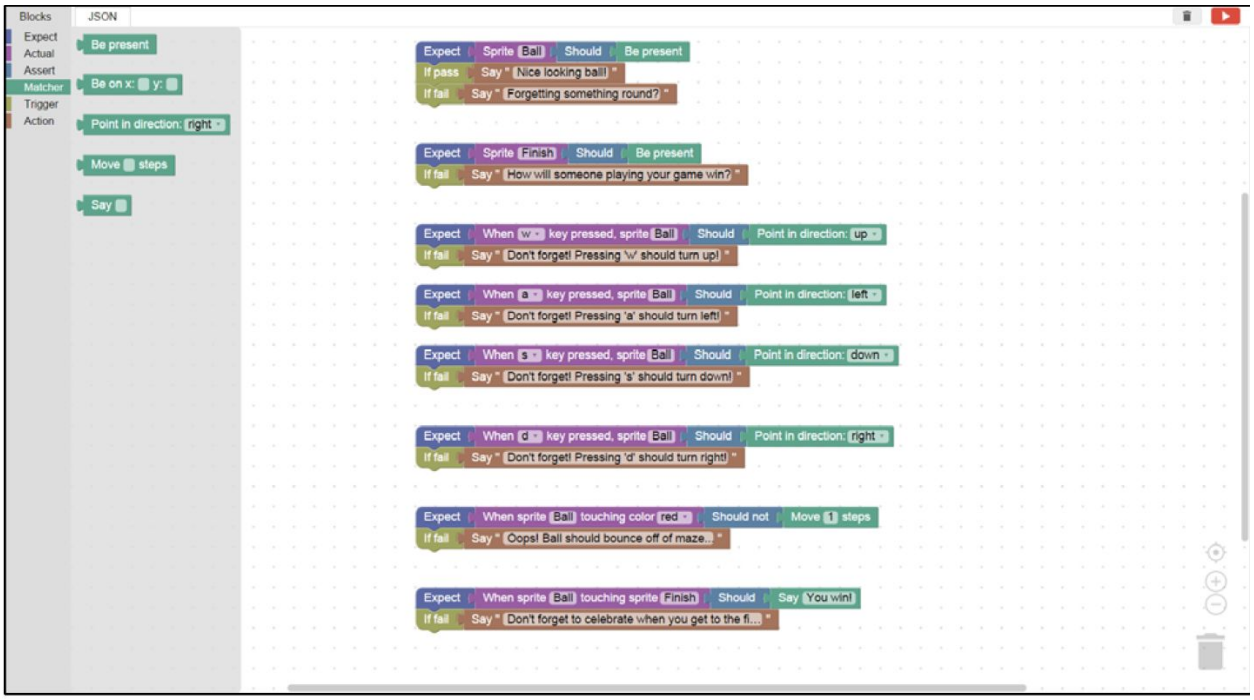


Figure 3: Teacher view of Visual Assessment Language Editor

Our plan for features were mostly twofold: First, we wanted to improve upon the front-end client facing aspects of SAGE that the teacher and student would work with directly. This would be adding gamification features that students engaged with directly from the Scratch Editor and/or Dashboard and tools the teacher would use, such as the Visual Assessment Language, designed to make assessments.

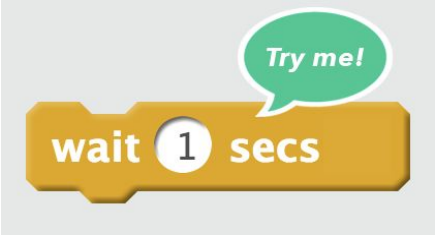


Figure 4: Example of a visual prompt, something that we planned but didn't make it to production

In early October, we started with the basic idea of adding point values, visual prompts, and reward mechanisms in the Scratch editor to increase student engagement with the system and to

encourage students to achieve teacher-defined milestones. Later as the project progressed, we divided this idea into two places where we injected points.

3.1.1 Points in Scratch Editor Blocks

In the SAGE editor, each block is now associated with a point value. These point values are called Block Points. When a student is building a project in the Scratch editor, these point values appear as numbers displayed on the block inside a star. Blocks used in the project will have their point values totaled in the editor to give the student a certain score for the project. Teachers can assign weights to blocks by using SAGE's edit mode, called SAGE Design Mode.

Figure 5: Example of Block Points in Play Mode. The "wait 1 secs" block is worth 1 point. The "repeat 10" block is worth 2 points.

Block Points allow for the automatic quantification of different problem-solving approaches. There may be several ways to solve a particular programming problem, and Block Points assign a numeric measure of "success" to each of these solutions. For example, blocks that may represent more complex programming concepts may be worth more points that blocks that represent simple concepts. In this case, teachers may encourage students to get more points by using blocks with greater point values. Alternatively, perhaps a teacher wants students to program concisely and elegantly, and she does so by encouraging students to complete an assignment with the lowest point value possible. Either of these setups would be useful for games such as Parson's programming puzzles: there may be several ways to solve a puzzle, but "better" solutions will be worth better points.

There are several ways that Block Points may help SAGE become a better conduit for Computational Thinking. They will serve as extrinsic motivation that can act as a springboard for intrinsic motivation and deeper learning. First, these points are a game-based feature that may improve effectiveness of SAGE as an educational software system by increasing the student's

motivation to engage with the system [4]. McNamara et al. list several types of motivation that may be improved by points:

- *Self-regulation* is the student's process of seeking to understand a topic by monitoring and adapting their own cognitive processes [7]. Having to think about how to obtain an optimal number of points will encourage students to set goals and be creative in order to come up with solutions. When a student knows little about the topic at hand, it is more reasonable to expect students to set and achieve tangible game-based goals as opposed to topics-based goals that require a deeper understanding of the material.
- *Self-efficacy* is the student's sense of success and achievement [8]. Having a point value provides an indicator of success to the user. The more empowered and successful the learner feels, the more likely they may be to engage with the system.
- *Interest* is the student's interest in the subject material. Framing the content within a game can be used to capture the interest of the student [4]. Block Points embed the project creation process in a game-like context that may increase interaction with the system, leading to interest in the topic.
- *Engagement* is the student's desire to interact with the system. The opposite of engagement is boredom. Bored learners think less and learn less [9]. Gamification through Block Points can help stave off boredom and increase engagement.

Block Points will also push SAGE further in the direction of a constructionist video game. In constructionist video games such as Minecraft and Roller Coaster Tycoon, the construction of in-game artifacts is the primary gameplay [3]. Weintrop et al. propose that an important principle for designing a constructionist video game is that in-game goals should encourage exploration. Block Points can reward exploration by making more interesting blocks worth more points. It'll help make the environment more discovery rich and not limit the player to a single winning strategy. With the points system, all correct solutions are rewarded with points, but the player can always try to find a better solution with a better point value. Players will be incentivized to discover different solutions to receive better points. This encourages self-regulation and creative thinking, leading to further understanding of the topic and deeper learning.

3.1.2 Points in Action Block of Visual Assessment Language and Quests

In addition to gamifying the Scratch Editor itself, we have designed and planned a way to integrate points that aligned much closer to teacher-defined assessment criteria. The motivations

for doing so align closely with reasons for adding points to blocks in the Scratch Editor. However, there are several motivations for doing so on their own. For a glance at the existing stage of the project before we started, Formative SAGE Assessments formally describes the Visual Assessment Language (VAL).

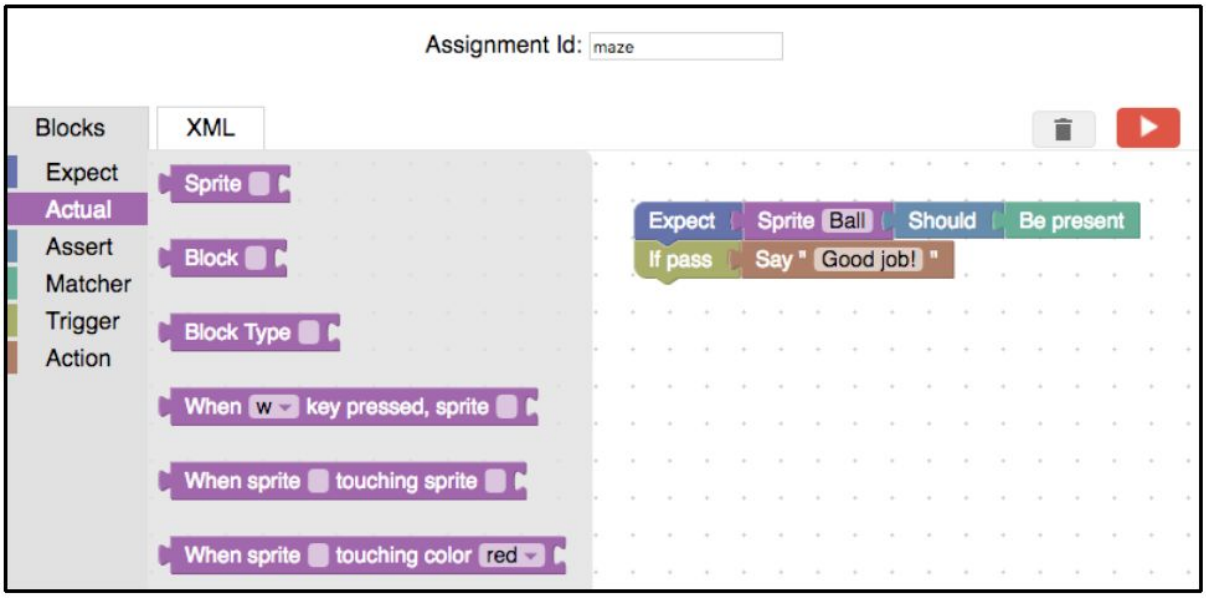


Figure 6: Visual Assessment Language (VAL) Editor

VAL is a client front-end of SAGE that was designed for teachers to use. More specifically, it allowed teachers to define the assignment that students are to complete and a list of assessment criteria that the teacher may use to evaluate student submissions of that assignment. It was purposely designed to look and feel much like the blocks that students would play with in the Scratch Editor Palette [18]. These assignments are uploaded to a server which queries for a student's progress on these criteria in real time, and actively provides them feedback with minimal prompting.

Assignments were stored in a persistent server, but the functionality was extremely limited. A teacher could only type in assignment number to save it into the database. There was no way of editing assignments post assignment uploading. The XML tab was a way of quickly converting from the Blockly visuals to raw text data. However to upload an old assignment this would have to be done by the teacher manually in a series of steps:

1. Do something in the VAL block editor
2. Tab over to "XML"

3. Copy and paste the XML into a text document somewhere, and name and save the file locally.
4. Open that file again
5. Tab over to XML and paste the contents of the text file into the VAL XML editor
6. Tab back over to “Blocks.”

It became apparent that improving upon the structural architecture of assignments would have to be done in tandem to allowing points for particular actions.

These assessments are also aligned with the Progression of Early Computational Thinking (PECT) model of learning, which maps computational thinking concepts to Design Pattern Variables such as

- animate looks
- motion
- collide
- conversate
- user interaction
- scoring

These concepts have certain levels of difficulty ranged from least to most, with animate looks corresponding to the lowest level and scoring associated with higher levels of computational thinking [17]. Although there exists a way of assessing for this criteria, currently there exists no way for teachers to easily define progression through these concepts. The teacher could in theory publish a numbered order in which students should complete assignments, but again this would be incumbent on both teacher and students to remember the order, type them in one by one by name or number, and finally load them in the Scratch Editor. This process is lengthy, cumbersome, and not easy to set up or remember. We can do better by automating this process.

Building points into the VAL and formally encoding them with other assessment criteria would strengthen directly student understanding and engagement in two ways:

1. Students can work towards concrete “goals” associated with points. These goals are aligned with and increase in difficulty with various computational thinking skills.
2. Students have a sense of progression as they can see the points they’ve accumulated in the past, and how much further they must achieve in order to achieve the next “goal,” thus engaging them with gamification tactics.

For teachers, points the VLA Editor improvements allow them to

1. Load a previous version of an assignment they had worked on

2. Save an assignment from in VLA to the SAGE server, which stores assessment points they have designated from the editor directly

The hope is that once teachers have automated tools for designing assignments and assigning points to various criteria, it wouldn't be an afterthought for them to design meta-projects that can group assignments by difficulty category or assign a series of assignments that slowly release students to learning deeper concepts. They would have all the tools they need as soon as they've finished designing just a few assignments using VAL.

With points enabled in VLA, we decided sometime in mid-November to begin designing a system for teachers to also group assignments and allow for unlocking of assignments. This premise lead to the idea of Quests in addition to the existing Assignments. The basic idea is that as students play and achieve points on assignments, this would unlock harder and more complex assignments that are designed to teach higher level computational thinking skills. We needed a system which would enable teachers to automatically assign point unlock values to each assignment and also group assignments under a single Quest. Naturally this would be implemented in a teacher dashboard-like environment similar to the one existing for students. It would take data both from the VLA editor and additional inputs from the teacher on the dashboard to make this a reality.

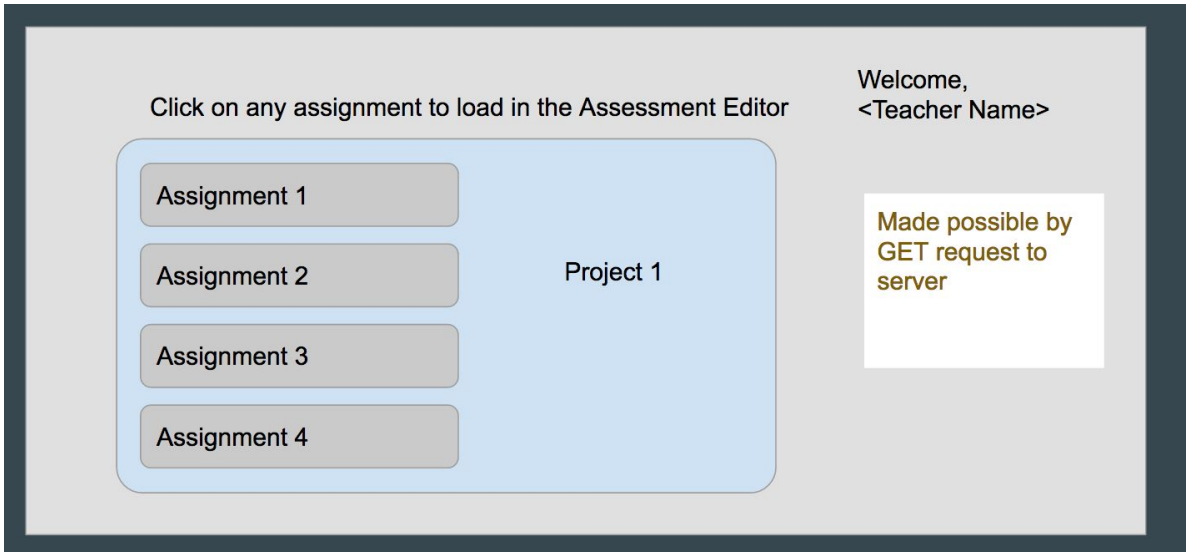


Figure 7: Design plan for teacher front-end for grouping assignments by project (November 2016)

3.2 Architecture/SAGE Assessment Server

The SAGE assessment server is the main point of communication between the SAGE front-end server, the Scratch editor, and Scratch projects. It is the only server that is directly connected to the database, so all applications must communicate with the SAGE server in order to save and retrieve data from the database. It is also the layer that processes data and returns data in a form that other applications can ingest.

A diagram of the SAGE server along with the applications that use it is shown below. All of the arrows represent HTTP requests except for the arrow between the SAGE assessment server and MongoDB.

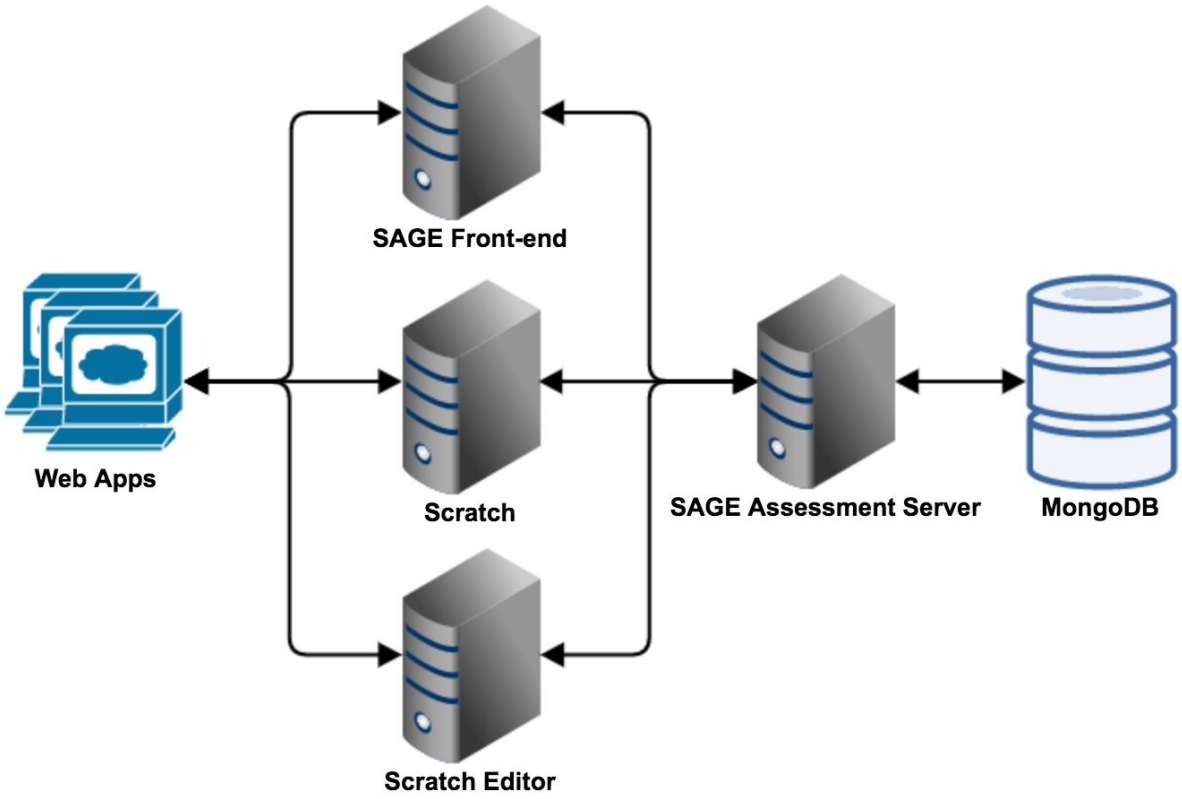


Figure 8: SAGE Architecture

At the beginning of this project, the SAGE server had only a basic set of communication endpoints for all these different applications. The server was able to store data from the Scratch

projects and Scratch editor but provided little functionality to make updates or create connections between related pieces of data. For example, every time project data was sent from Scratch to the SAGE server, it would create a new document was being stored in the database rather than checking if the project had already been stored. Also, the SAGE server had an endpoint for teachers to store their assignments but no endpoint to retrieve the assignment after, forcing the teachers to store their files locally.

The improvements to the SAGE assessment server had two main goals.

1. The first goal was to build new endpoints or refine existing endpoints to add functionality accommodating the new features being built by the research teams this semester.
2. The second goal was to improve the infrastructure of the SAGE server to make it easier to maintain and setup for new developers.

Overall, we worked towards improving the SAGE ecosystem for teachers and students to easily access tools for learning and assessment. This is very important foundational work for future development as we can more easily access and pass around only the data we need. Better organization and documentation means further development work can be more guided and have clearer objectives. As more people work on the project it can be subdivided more cleanly.

3.3 Roles

3.3.1 Julie Chien

Julie is working on her M.S. in computer science in the Vision and Graphics track. Previously she worked with Iris at InteractAble, a small educational technology company. At InteractAble she worked on a mobile game geared towards social skills learning for kids on the autism spectrum. Julie worked on the Block Points feature in the SAGE-Scratch editor.

3.3.2 Iris Zhang

Iris is pursuing her M.S. in computer science in the Software Systems track. Previously she had worked at an educational games start up where she designed and developed iOS and cross-platform mobile games to teach children social skills. She also had a brief stint as a high school teacher. She hoped to use her background in both game development and education to add

to an open source project in gamifying teaching tools. She worked on improving the Visual Assessment Language tool for teachers to better integrate it into the SAGE architecture. She also designed wireframes for the Teacher Dashboard for making Quests and Assignments, and helped Ray with designing the API endpoints needed for her portion of the project.

3.3.3 Ray Tsai

Ray is a master's student studying computer science with a concentration in Computer Graphics and Vision. He has worked as a full-stack web developer at multiple companies ranging from brand new startups to international media companies. Ray specializes in creating consumer-facing web experiences and has recently decided to use his powers for good, helping build open-source tools. He worked on building the architecture side of SAGE including expanding the API endpoints and improving its infrastructure.

3.4 Timeline

We had a timeline submitted with our project proposals mid-October with milestones to benchmark our progress. Because the project was open ended, the goals we set out at the beginning and the actual milestones we accomplished ended up looking quite different.

3.4.1 Gamification Features Milestones (Projected)

Date	**Milestone (Projected)**
Oct 7, 2016	Development environment setup
Oct 14, 2016	Design of game elements
Oct 28, 2016	Implementation of real-time Scratch editor feedback
Nov 11, 2016	Implementation of point/trophy system Communication with backend architecture
Nov 25, 2016	Implementation of leaderboards
Dec 9, 2016	Improvements

3.4.2 Architecture Milestones (Projected)

Date	**Milestone (Projected)**
Oct 7, 2016	Development environment setup
Oct 14, 2016	Design of API endpoints
Oct 28, 2016	Implementation of API endpoints
Nov 11, 2016	Implementation of Scratch editor communication blocks
Nov 25, 2016	Analysis of data to be logged
Dec 16, 2016	Implementation of database and logging

3.4.3 Gamification Features Milestones (Actual)

Date	**Milestone**
Oct 7, 2016	Development environment setup
Oct 14, 2016	Design of Scratch Editor point blocks
Nov 14, 2016	Scratch Editor block points implemented
Nov 19, 2016	Scratch Editor loading and saving points locally
Dec 15, 2016	Block Points playing well with other Scratch features
	Design of Visual Assessment Language Block Points
Oct ?, 2016	Visual Assessment Language Add Points Action implemented
	Design of Quest and Assignments Teacher Dashboard
Dec 9, 2016	Loading and saving assignments from SAGE servers

3.4.4 Architecture Milestones (Actual)

Date	**Milestone**
Oct 7, 2016	Development environment setup
Oct 28, 2016	Migration to Node.js and MongoDB
Nov 11, 2016	Design of Teacher and Student API endpoints
Nov 18, 016	Implementation of Teacher and Student API endpoints
Nov 25, 2016	Design of Quest and Assignments API Endpoints
Dec 9, 2016	Implementation of Quest and Assignments API Endpoints
Dec 16, 2016	Documentation and Code migration to public Github repo

4. Implementation / Features

For the implementation aspects of this report, we will go into specific technical details about our project, including technical specifications, architectural diagrams, and actual code written. The various repositories we worked on were:

Project Area	**Repository Location**	**Party Responsible**
4.1.1 Points in Scratch Blocks	https://github.com/cu-sage/sage-scratch	Julie
4.1.2 Points in Visual Assessment Language Blocks	https://github.com/cu-sage/sage-editor	Iris
4.2 Architecture	https://github.com/cu-sage/sage-node	Ray

4.1 Gamification Elements

4.1.1 Points in Scratch Blocks

This section describes usage, implementation, and technical details of Block Points.

4.1.1.1 Background Technologies

Since Block Points are displayed and modified in the Scratch Editor, all of the implementation took place in the SAGE-Scratch code repository, which is an extension of the Scratch Editor code repository. Scratch is an Adobe Flash project and the SAGE-Scratch Editor executable is a Flash (.swc) file. Adobe Flash, Apache Ant, and Apache Flex are required to build and run the executable. The code is written in Adobe ActionScript 3, the primary language used to develop Flash applications.

4.1.1.2 Block Points in SAGE Play Mode

Students will build Scratch projects in SAGE Play Mode. In this mode students can build Scratch projects and submit assignments. Block Points in Play Mode are simply numbers displayed in a star in the block they're associated with. As students build projects, the point values of the blocks used to make their project are summed and displayed on the Scratch Editor. The feedback is live; students can see their point progress as they build projects.

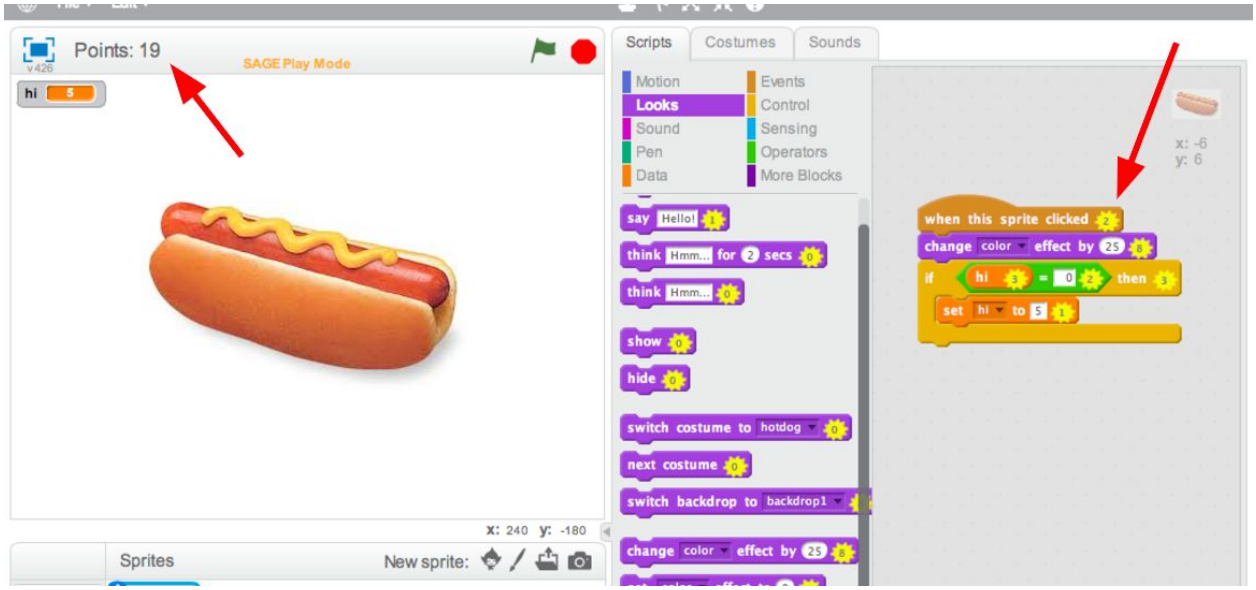


Figure 9: Points are displayed on blocks in Play Mode. Total points are displayed in the upper left corner of the editor.

The total score is incremented when a student drags a block from the palette (the middle section above that contains the block "bank") to the stage (the rightmost section above). The score is decremented when a block or stack of blocks is dragged from the stage to the palette. The total point value is simply a sum of the current blocks in the stage.

4.1.1.3 Editing Block Points in SAGE Design Mode

Teachers can configure Block Point values in SAGE Design Mode. This is the "editing" mode that allows teachers to set up SAGE assignments.

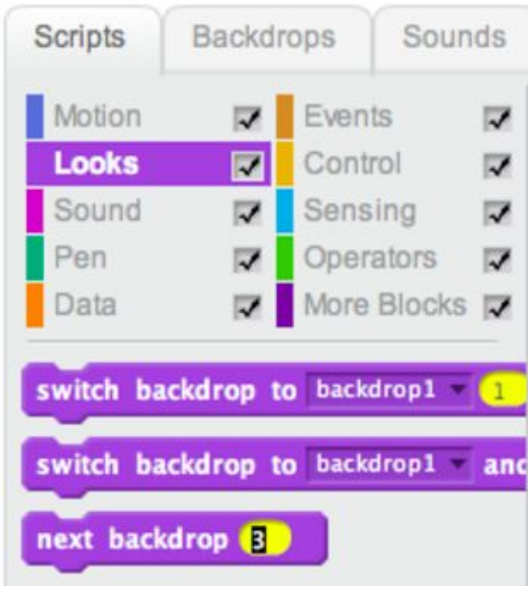


Figure 10: Points are editable in SAGE Design Mode

In SAGE Design Mode, the points are shown as arguments on top of the block. They are editable and changes are auto-saved while the teacher configures the points. To modify a point value for a particular block, simply click on the yellow oval on the block and enter a new numeric value.

The UI design is intentionally similar to what the student sees in Play Mode. According to the American Educational Research Association, teachers should be as familiar as possible with the topics presented to students (David & Krajcik, 2005). Having the UI be as similar as possible between Design and Play Mode will hopefully allow teachers to come up with better point configurations by having them be proximate to what the students will experience.

4.1.1.4 Saving and Loading Block Point Configurations

Point configurations can be saved to file and loaded from file. When loaded from file, the point configuration is stored as an in-memory static dictionary in the Specs class in the SAGE-Scratch codebase. The program reads and writes from this dictionary when it's executing. When a point configuration is saved, this dictionary gets serialized to a JSON file that's saved locally.

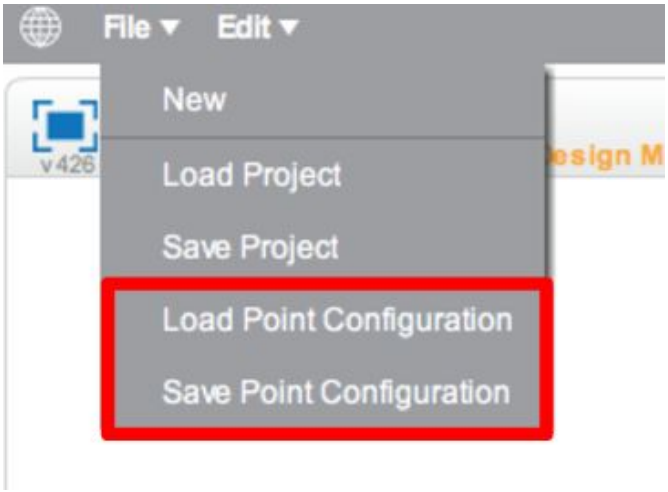


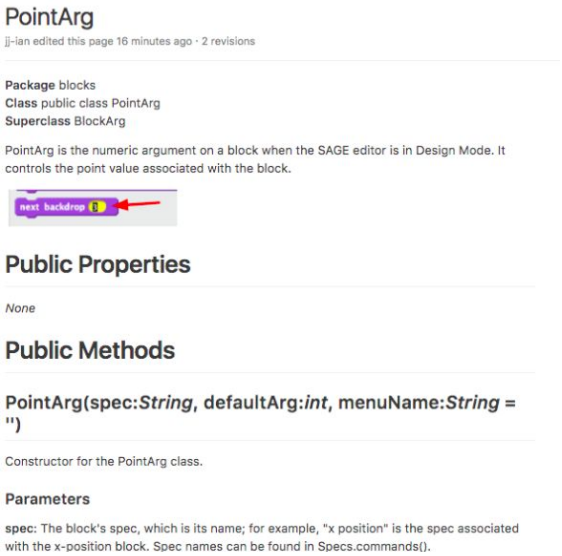
Figure 11: Loading and saving point configurations

The user must be in SAGE Design Mode to save and load point configurations. When in Design Mode, these options are available from the Edit menu.

Point configurations are also saved and loaded automatically when saving or loading a SAGE project, thanks to the Parson's programming puzzles team.

4.1.15 Documentation

Documentation for the changes made in SAGE-Scratch can be found in the SAGE-Scratch Wiki.



PointArg

jj-ian edited this page 16 minutes ago · 2 revisions

Package blocks
Class public class PointArg
Superclass BlockArg

PointArg is the numeric argument on a block when the SAGE editor is in Design Mode. It controls the point value associated with the block.

next backdrop 0

Public Properties

None

Public Methods

PointArg(spec:*String*, defaultArg:*int*, menuName:*String* = '')

Constructor for the PointArg class.

Parameters

spec: The block's spec, which is its name; for example, "x position" is the spec associated with the x-position block. Spec names can be found in Specs.commands().

Figure 12: A sample of documentation

4.1.2 Points in Visual Assessment Language Blocks

This section goes into the technical details of the VAL and how points were implemented to fit into VAL and SAGE.

4.1.2.1 Background Technologies

The Visual Assessment Language developed for SAGE was implemented using Blockly for web, a Google open source project designed for easy drag and drop visual code generation. Blockly is written in Javascript and is 100% client side, as well as easily customizable and extendable [20].

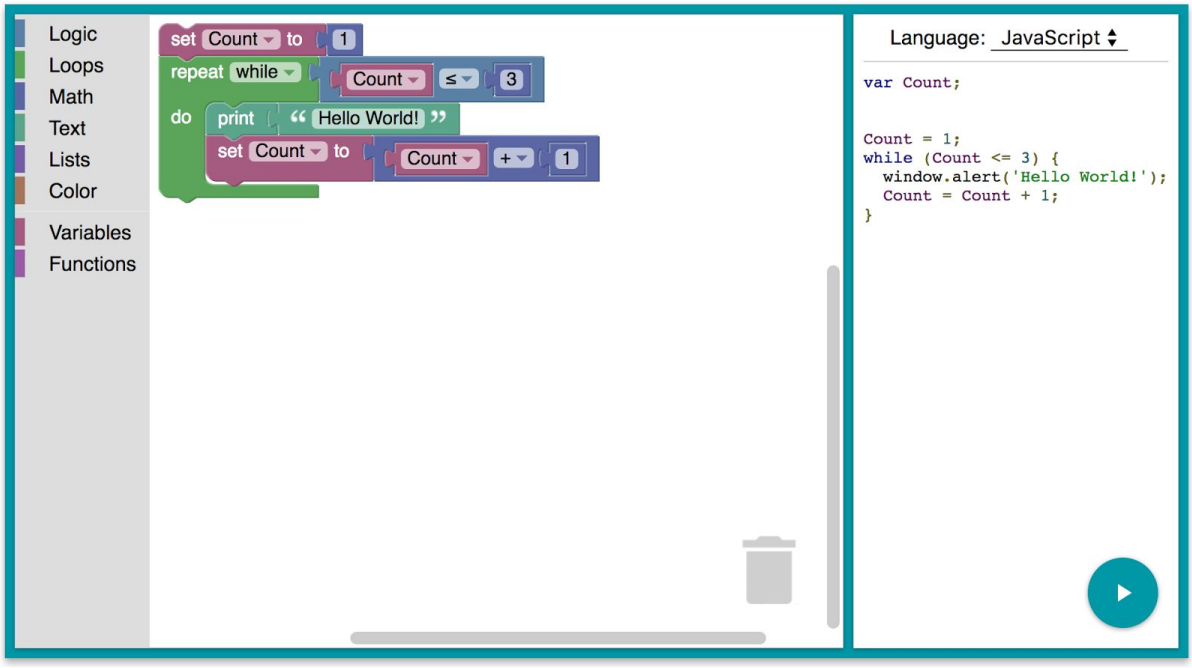


Figure 13: Blockly Web Editor, taken from their website

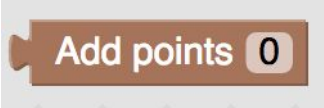
VAL was implemented on top of a Blockly web demo and allowed for a drag and drop interface where a teacher could customize the logic in which to evaluate blocks. In addition to Javascript, some Ajax technology existed to upload and download these blocks to/from server using HTTP GET and POST calls. However, only the upload functionality was implemented and you had to specify an assignment string that would then store raw XML into the database. Editing of an assignment post-uploading was not possible.

In addition to this, jQuery was added into the project as a dependency library for easier parsing of XML and easier AJAX calls.

4.1.2.2 Blockly Action "Add Points"

The point functionality would be attached to a single concept that the teacher could decide to assess. For example, in the existing framework the teacher could use Blockly to test for the presence of a sprite named "ball" and offer a message such as "great use of sprites!" to be presented to the student in the Dashboard in real time. To fit points into this, we decided to add an additional "add_point_value" action block. This would work in tandem or in lieu of the "say" and "include" action blocks. For a full description of the Visual Assessment Language and grammar, see Formative SAGE Assessments [18].

`actn-add-points-block`



The Add Points block is used to associate points with the entire block if the previous trigger is invoked. It takes one argument. The argument is of data type number, at the least 1 and at the most 500, and only whole number increments. The default argument is 0. It represents the point value that should be associated with taking a particular action. To add this particular functionality, we added code to the file `blocks/testing.js` and `app/editor/index.html` in the repository. The most relevant code is highlighted here:

```
80 ▼ Blockly.Blocks['action_add_points'] = {
81 ▼   init: function() {
82 ▼     this.appendDummyInput()
83           .appendField("Add points")
84           .appendField(new Blockly.FieldNumber("0", 0, 500, 1), "point_value");
85     this.setOutput(true, "action_add_points");
86     this.setColour(20);
87     this.setTooltip('');
88     this.setHelpUrl('http://www.example.com/');
89   }
90 };
```

Figure 14: Relevant code in `blocks/testing.js` in the SAGE-editor repository

The way the teacher would use the block is as an action that would be attached to a certain trigger. The two triggers available in VAL are `if-pass` and `if-fail`. The `actn-add-points-block` was designed to work with the `if-pass` trigger. Although a teacher could choose to associate points with failing to do something, that behavior is undefined.

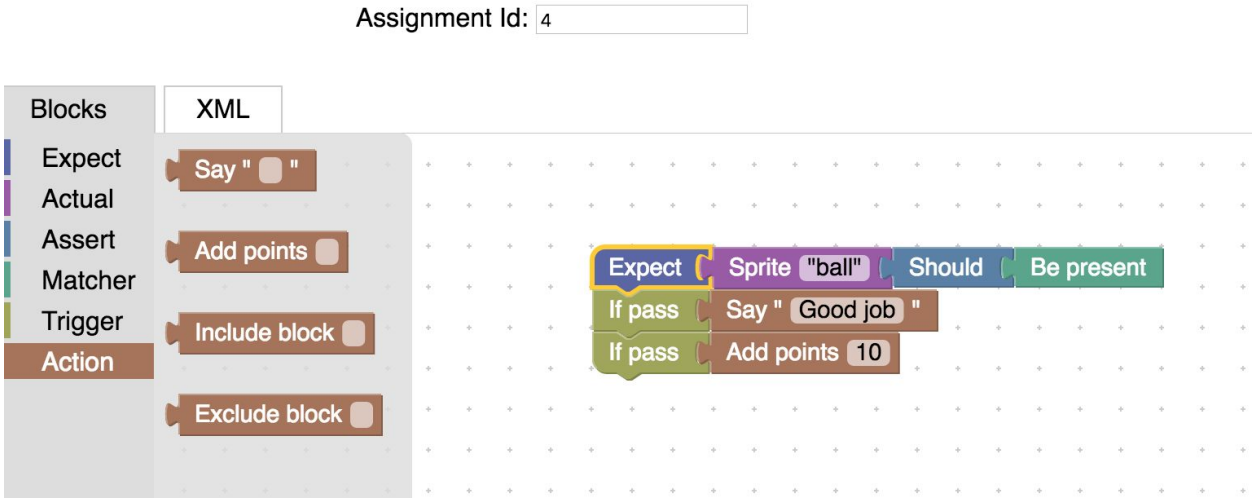


Figure 15: Example of how to use add points in an assessment criteria

In addition there was need for parsing the point values within the VAL front end code. This is because the teacher could add multiple assessment criteria all with different point associations, thus we needed a way to detect when the `actn-add-points-block` was being used and add up the points for each block. The logic for this is contained in `app/editor/code.js`. The way parsing was implemented was to use jQuery to introspect the XML generated. XML generation happened previously upon the clicking of the XML tab. However this functionality will probably become deprecated later in the future when there is no need for local storage of XML anymore. Therefore, the parsing happens only when the teacher hits the “play” (red arrow) button to the upper right corner.

4.1.2.3 Updating Blockly

An important thing to note is that the VLA editor uses a local copy of Blockly. Blockly itself is an open source project and is being constantly updated, but the VLA editor does not update automatically to the latest version of Blockly. Therefore any new features added that uses newer Blockly features must mean a migration to a new version of Blockly. The last version used of Blockly was updated sometime in April . The currently used version is from November 11, 2016:

(https://github.com/google/blockly/commit/c9ef71ecf3469c009a6e991942482219e8a92e02). To use the most current version of Blockly, we must download the latest files

- `blockly_compressed.js`
- `blocks_compressed.js`

and include them in our source repository.

In implementing this project and getting the desired feature for whole number validation in the add points block, we had to update to the latest version of Blockly. In addition the VLA Editor had used some older features of Blockly and we had to make adjustments to get the source code to compile without errors or warnings. For instance, we had to swap out the deprecated way of reloading tabs after clicking on the XML tab, as well as the way XML gets converted into Blocks and vise versa. Both of these issues were documented on Github and the suggestions were taken into account when updating Blockly for SAGE.

- Issue 1 - Workspace doesn't redraw properly any more after latest updates. #399 (https://github.com/google/blockly/issues/399)
- Issue 2 - Blockly Factory: Confirm Changes with User, Reduce Alerts, Generate Starter Code #606 (https://github.com/google/blockly/pull/606)

4.1.2.4 Loading and Saving of Assignments

In addition to the implementation of points, we had to implement a way to support teachers to easily edit and update their assignments. These HTTP endpoints enable the ability to load an assignment from a GET request and save or update an assignment via a POST request. These are documented in the wiki of the SAGE server repository.

GET assignments/:id

Returns information for a specified assignment.

Example Request

```
localhost:8081/assignments/583c9865aa877721348f427e
```

Example Response

```
{
  "id": "583c9865aa877721348f427e",
  "xml": "<xml xmlns=\"http://www.w3.org/1999/xhtml\"><block type=\"expect\"
id=\"v%kq5WgSPvUoxlI1uljg\" x=\"38\" y=\"38\"></block></xml>",
```

```
  "teacher": {
    "id": "582a1be95789252c48ada270",
    "name": "Valerie Frizzle"
  },
  "quest_id": "584c3b9f042e59a737c762a7",
  "quest_sort": 2,
  "points_total": 20,
  "points_unlock": 10
}
```


POST assignments/:id/update_xml

Updates the xml for an assignment.

Body Parameters

Name	Required	Description	Default Value	Example
xml	required	The updated xml of an assignment as a string.		""
points_total	optional	The updated total points of an assignment.		28

Example Request


```
{
  "xml": "<xml xmlns=\"http://www.w3.org/1999/xhtml\"><block type=\"expect\"
id=\"v%kq5WgSPvUoxlI1uljg\" x=\"44\" y=\"38\"></block></xml>",
  "points_total": 28
}
```


Example Response


```
{
  "teacher": {
    "id": "582a1c125789252c48ada271",
    "name": "Severus Snape"
  },
  "id": "583c9865aa877721348f427e",
```

```
  "xml": "<xml xmlns=\"http://www.w3.org/1999/xhtml\"><block type=\"expect\"
id=\"v%kq5WgSPvUoxlI1uljg\" x=\"44\" y=\"38\"></block></xml>",
  "points_total": 28
}
```


The GET and POST requests are used from the front end by the teacher in the VAL editor. When the teacher first loads the assignment, they do so by visiting a URL that in its fragment the assignment id. The loading by prompting the code to check for a fragment in the URL, and then making a GET request with that assignment id to the server. The XML in the JSON that is returned by the server is parsed and placed into the workspace and the page is forced to reload. This will upload the previous blocks designed into the palette.

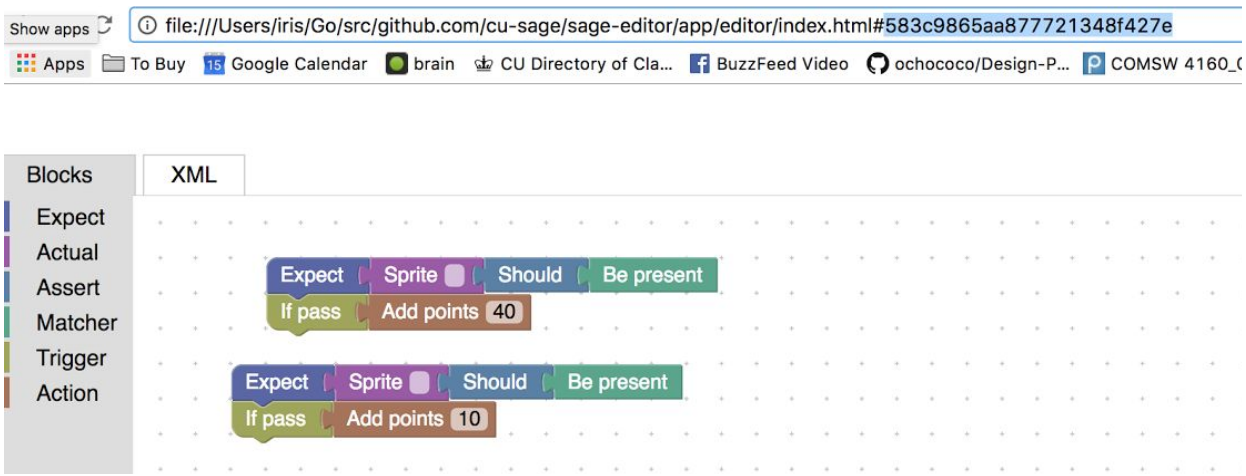


Figure 16: Loading assignment 583c9865aa877721348f427e into the editor

Once the teacher has finished updating the assignment, they can press the red “play” button in the upper right corner to upload the updated assignment into the SAGE server. This prompts an alert that contains the JSON being uploaded. The alert is a visual cue to elucidate what is happening in the back end. However, in future considerations this would probably become deprecated as this front end becomes integrated with the teacher Dashboard. Note that in this example provided, the assignment is now made with three assessment criteria, one with 40 points, one with 10 points, and one with 30 points. They add up to 80, which is the `points_total field` that is sent to the server.

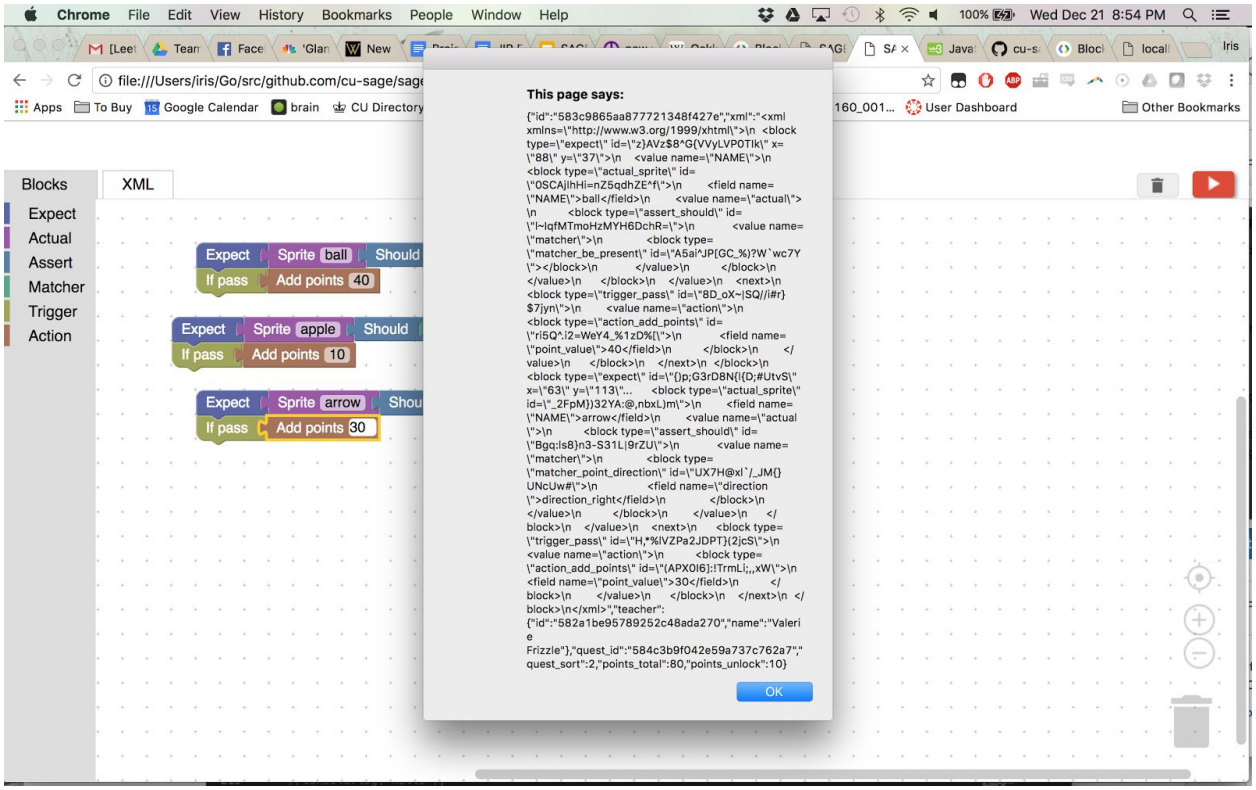


Figure 17: Uploading assignment 583c9865aa877721348f427e with additional assessment criteria worth 30 points, for a points_total of 80

4.1.2.5 Design of Teacher Dashboard to manipulate Quests and Assignments

In lieu of implementing an entire front end for this within the VLA, we decided to draw out the wireframes for how this system would work in preparation for future work. The figure below shows a web app front end designed in Sketch with the elements a teacher would need to design her Quests and Assignments. On the left pane, unsorted assignments are listed. Clicking on any of them would take her to the VLA Editor, where she could manipulate assessment criteria for students. Each assignment's name and `points_total` is shown in this pane. On the right, the Quests are listed in order of when the teacher created them. A teacher can drag and drop assignments into the Quests pane and reorder them in whatever way she wishes. She can also assign a `points_unlock` value in a field next to the assignment under Quests. This associates an unlock value to that particular assignment. This value translates to how many assessment criteria points a student must achieve on that assignment in order to unlock the next assignment in the Quest. Each assignment under a Quest will have an unlock value except for the last, and except if there is only one. Further discussion on the Teacher Dashboard Quests and Assignments view is outlined in the Looking Ahead section.

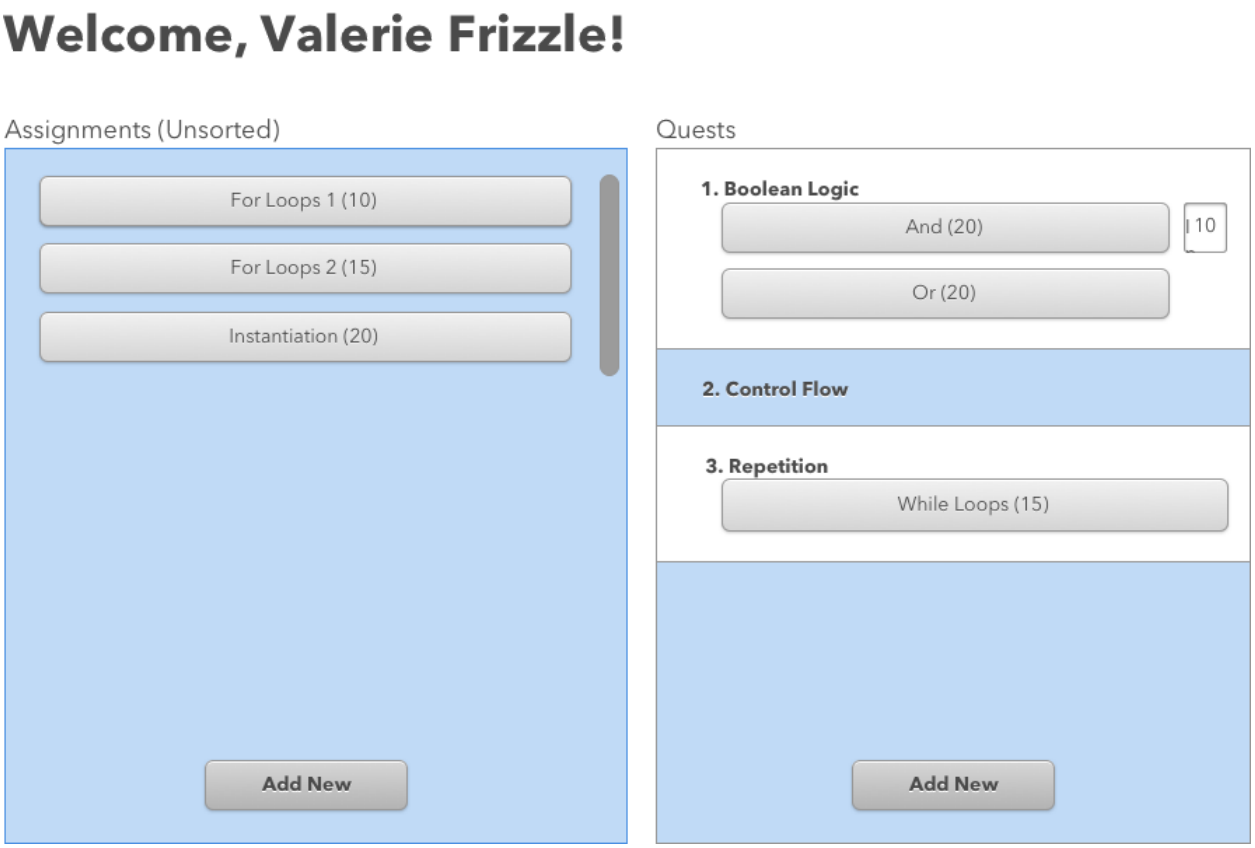


Figure 18: Teacher Dashboard for Ms. Frizzle to design her Quests and Assignments

4.2 Architecture

4.2.1 Node.js

The first major set of changes made to the SAGE assessment server was migrating all the existing functionality from Go to Node.js. This decision was made in lieu of the fact that JavaScript is a far more widely adopted language than Go (Stack Overflow 2016), so future developers for SAGE are more likely to know JavaScript than know Go. Additionally, the current developers for this project working on both the SAGE Server and the Sage front-end were more familiar with JavaScript, allowing the teams to make more progress by working with Node.js.

The Node.js server is modularized for ease of code maintenance. The code structure has layers of logic between where the server receives HTTP requests and the database in order to separate the

business logic. Each of the endpoints has its own controller, database service, and formatter. The controllers are responsible for receiving incoming HTTP requests, calling functions in lower-level modules, and combining the returned data. The database services handle all of the database transactions such as queries and updates. Finally, the formatters convert the data from the database into the format the original application is expecting.

The typical API call will then look like this:

1. An application makes an HTTP request to the SAGE server using one of the RESTful API endpoints.
2. The Node.js application routes the request to a controller.
3. The controller reads the data from the application and formats it.
4. The controller uses the database service to query or update the database.
5. The controller modifies the data such as joining additional data using the database services of other collections.
6. The controller formats the data and returns the data to the application via HTTP request.

4.2.2 Database

MongoDB was chosen as the database for the new SAGE assessment server simply for ease: Many libraries and utilities already exist for connecting Node.js to MongoDB. As MongoDB has no built-in schema mapping or validation, Mongoose ODM is used to model the data stored by the SAGE server. This library allows the server to verify the application data is stored in a consistent manner and also comes with functions to easily query and update data.

There are six collections stored in the database:

1. assignments
2. quests
3. assessments
4. students
5. teachers
6. classes

The assignments, quests, and assessments are used by the Scratch projects and Scratch editor. The students, teachers, and classes are used by the SAGE front-end server. There may be some confusion between assignment and assessment, so in further work there may need to be renaming

of one or more of these collections. Assignments are created by teachers and contain logic for assessing the projects that the students work on. Quests are groups of assignments which share a common theme. Assessments contain the results of a student's progress on a Scratch project which is created using the evaluation criteria defined by the teacher's assignment.

The students, teachers, and classes collections are used by the SAGE front-end server to store data for all the users. As their names suggest, the students collection is used to store student information, and the teachers collection is used to store teacher information. Classes are led by one teacher and contain a group of students. They also contain leaderboards that show the highscores of students enrolled in the class. The highscores are determined by students completing assignments which are analyzed by the SAGE server to create assessments.

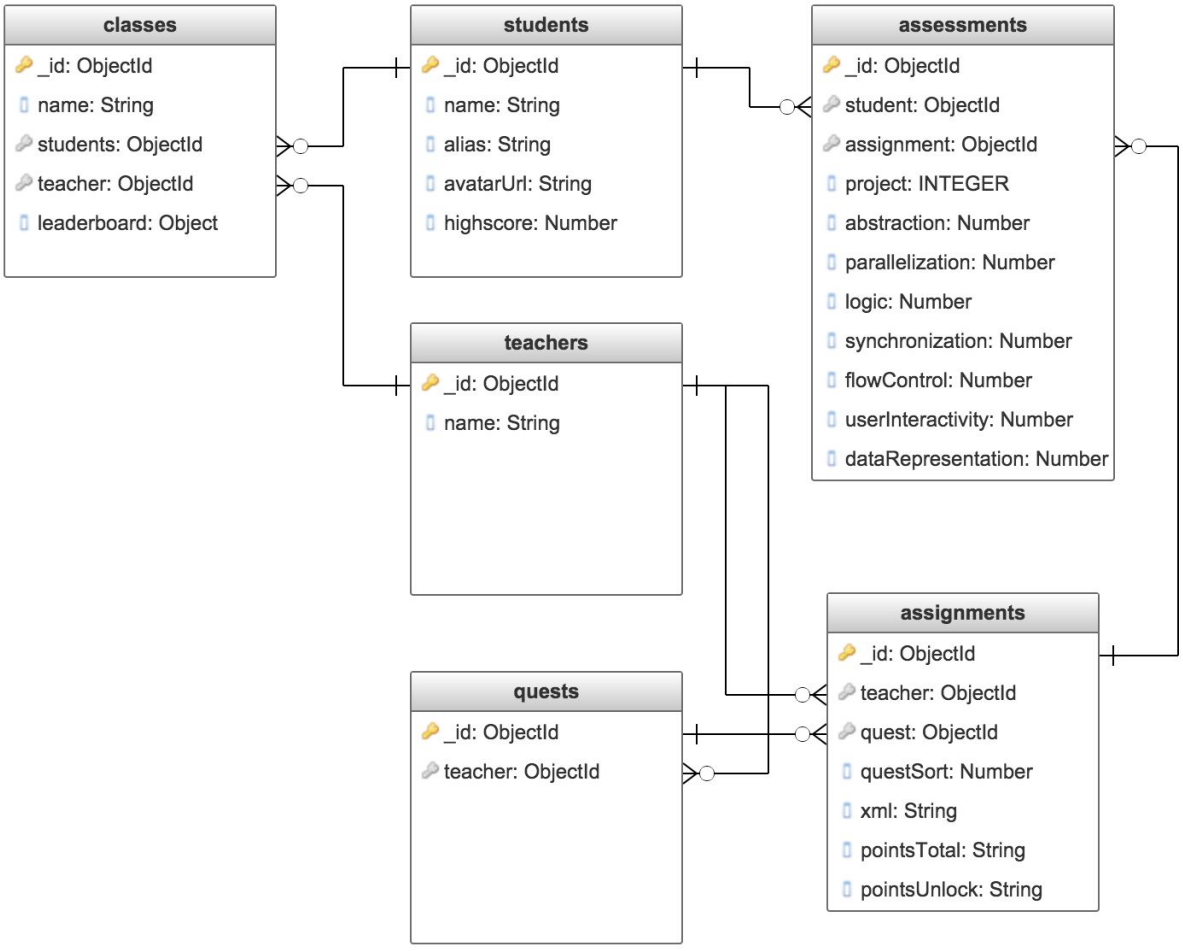


Figure 19: Database Schema for SAGE Server. The schemas are validated using Mongoose ODM.

4.2.3 RESTful API Endpoints

All available endpoints for the SAGE assessment server are listed below. The GET endpoints retrieve data without modifying the database. The POST endpoints all update or add data to the database. Each of the collections has endpoints for finding a single document by identifier, finding all documents, and creating a new document in that collection.

GET Endpoints	**POST Endpoints**
GET assessments/latest	POST assignments/:id/update_xml
GET assignments/:id	POST assignments/:id/update_quest
GET assignments/list	POST assignments/new
GET classes/:id	POST classes/:id/update_teacher
GET classes/list	POST classes/:id/remove_teacher
GET quests/:id	POST classes/:id/add_student
GET quests/list	POST classes/:id/remove_student
GET students/:id	POST classes/new
GET students/list	POST quests/new
GET teachers/:id	POST students/new
GET teachers/list	POST teachers/new

The full documentation for all the REST API endpoints can be found can be found on the Wiki of the GitHub repository. The documentation describes what the purpose of the endpoints, lists the available parameters, and give example requests and responses.

Documentation	**Link**
SAGE Assessment Server RESTful API Endpoints	https://github.com/cu-sage/sage-node/wiki/API-Documentation

For example, below is the documentation for the POST /classes/new endpoint, which creates a new assignment.

POST classes/new

Creates a new class.

Body Parameters

Name	Required	Description	Default Value	Example
teacher	optional	The identifier of the teacher leading the class.		582f7e45d35339723018e6d2
name	optional	The name of the class.		Defense Against Dark Arts

Example Request

```
{
  "name": "Defense Against Dark Arts",
  "teacher": "582f7e45d35339723018e6d1"
}
```

Example Response

```
{
  "id": "582f7e45d35339723018e6d2",
  "name": "Defense Against Dark Arts",
  "teacher": {
    "id": "582f7e45d35339723018e6d1",
    "name": "Remus Lupin"
  },
  "leadership_board": [],
  "students_enrolled": []
}
```

4.2.4 SAGE Assessment Server Startup Guide

The GitHub repository for the SAGE assessment server.

Project	**GitHub Repository**
SAGE Assessment Server	https://github.com/cu-sage/sage-node
SAGE Assessment Server (legacy)	https://github.com/cu-sage/sage

Below are instructions for installing the SAGE assessment server on a new environment.

1. Install MongoDB.
2. Install Node.JS.
3. Start the MongoDB daemon: `mongod`.
4. Use Git to clone the SAGE Assessment Server.
5. Import mock data into the database: `npm run mocks`.
6. Start the Node.js server: `npm run gulp`.

Once the steps above have been completed, the SAGE assessment server will be running locally and ready for development. The "gulp" library will automatically restart the server when any changes are detected in the code.

4.2.5 Code Maintenance and Stability

As mentioned previously, one of the other major goals for the SAGE assessment server infrastructure was to improve easy of code maintenance and ensure code stability. These changes come in three forms: adding utilities to write clean code with consistent style, and writing integration tests which test all the server's functionalities.

The utilities have been incorporated to the Node.js server to help developers write code faster and better:

1. ESLint
2. gulp.js
3. Express
4. Mocha/Chai

ESLint is a linting tool that checks code for errors and enforces style guidelines. It is particularly useful for JavaScript which is a scripting language and loosely-typed. Without ESLint, many syntax errors would not be found until the code is executed. Developers can run ESLint to check for warning and errors by executing the command `npm run lint`. Most text editors such as Sublime and Atom also have plugins which highlight errors immediately as code is written.

Gulp.js is a task runner that makes it easy to run repeated tasks. A server can use gulp.js to automatically restart the SAGE server upon file change by running `npm run gulp`. Express is a library for Node.js that eliminates a lot of boilerplate code for setting configurations and defining API endpoints.

Mocha and Chai are testing frameworks that a developer can use to run integration tests for the SAGE server. To do so, the developer must first start the server in test mode by running `npm run start:test`. The integration tests can then be run by executing `npm run test:integration`.

5. Limitations and Looking Ahead

5.1 Gamification Elements

5.1.1 Block Points

It is important to note that Block Points are a toolkit that must be used within some system and serve some purpose. By themselves, they add little value. Example systems that can use points are Parson's programming puzzles, leaderboards, or unlockable rewards. Adding game-like features isolation without context provide no benefits to either learning or motivation [4]. Future work can embed Block Points in useful contexts and examine whether or not it has an effect on motivation.

Another question is: how do you quantify individual blocks? It's difficult to quantify the worth of individual programming constructs since their value depends on how they are used in combination.

Lastly, the primary technical challenge we ran into regarding the Scratch Editor was the lack of documentation in the source code. We added documentation where we could, but development time was slowed down significantly by the lack of documentation.

5.1.2 Visual Language Editor Integration with SAGE

Working with Blockly was quite the challenge because it exists as its own open source project alongside Scratch and our SAGE project. Much development time was spent reconciling using the most currently documented features with an older version of Blockly. When we updated to a new version of Blockly, features that were written with the old version would break and had to be fixed.

Much time was spent trying to reconcile the many different front ends and servers that existed and many opportunities for future work should stem from this difficulty. Because the VLA was its own front end, some time was lost trying to add front-end features to the codebase before we came to the realization it would fit better with the Dashboard. Further work can be done to better integrate the VLA Editor with the rest of SAGE. In particular, there's no reason for there to be separate front ends for the VLA Editor and the Teacher dashboard. Once the VLA Editor is integrated with the rest of the Dashboard code, implementation for the Quests and Assignments (outlined in section 4.1.2.5) on this combined client-side code should come easily, although more design thought could be put into how to structure unlocking points for each assignment of a Quest.

5.1.3 General

The hope is that gamification of SAGE will increase student motivation, drive engagement with the material, and facilitate learning. Future work can work on answering the question: "Does gamified SAGE work better than regular SAGE?" We can try to answer this question by having users play Parson's programming puzzles with Block Points vs. without points. We can test users via Amazon Mechanical Paper as outlined in Lee, Ko, and Kwan's In-Game Assessments paper [11]. When testing, observe which aspects of motivation are enhanced, if any, and examine what the points make them do to see what the learning benefits are [4].

5.2 SAGE Assessment Server

One of the biggest challenges when implementing new features in the SAGE assessment server was designing the database schema. In a NoSQL database, data is stored as documents, and documents are stored inside collections, similar to how rows are stored in tables in SQL databases. The difficulty in designing a NoSQL database schema arises when documents have one-to-many relationships with other documents. The relationships can either be stored as an array inside the document referenced by many documents or as a property of the many documents which reference one document. Two examples of this type of relationship are the class-to-student relationship and the assignment-to-assessment relationship. Classes contain multiple students and quests contain multiple assignments.

The heuristic used to solve this problem is to design the schemas based on the use cases. In other words, the location of the relationship data depends on the endpoint used when creating the relationship. For example, when a student is added to a class, the "/class/:id/add_student endpoint" is used to update the data. Therefore the relationship data should be stored as an array of student identifiers inside the class schema. On the other hand, when an assignment is added to a quest, the "/assignments/:id/update_quest" endpoint is used, so the relationship data is stored as a property in the assignment schema. This heuristic then raises the question of how to design the endpoints. The answer will require a discussion of the use cases in the front-end portions of the web application.

5.3 Trophies

Trophies may be awarded after every level for using a certain number of points, using blocks that demonstrate important concepts, or achieving other milestones. We did not get the chance to fit this into any aspects of the project, but they could easily be displayed in the Student Dashboard using the data gathered either from the assessments criteria or Scratch Editor Block points.

5.4 Leaderboard

As a follow up to trophies and awards, future work can be done on a leaderboard that shows students with top scores. The motivation for building such a tool stems from studies that show that competition is a gamification element that adds a social and emotional element to learning. Although this is a reach goal for us, we have put thought into designing this feature to mitigate

risks of negative student outcome [15]. The leaderboards can anonymize personal information, leading to students only knowing how they individually achieved relative to the class.

6. Conclusion

Gamification of the educational system is at the intersection of computer science, education, and video games. At the heart of this system is Scratch, a free visual programming language developed by the Lifelong Kindergarten Group at MIT. Scratch alone is very free form and allows for a wide range of exploration with little guidance. SAGE brings Scratch into a more structured environment by organizing programming projects into different assignments; providing scaffolding, guidance, and feedback for students as they work; and including an assessment platform for automatic evaluation of assignments for educators. Our work lays the foundations for improving formative SAGE assessments by adding gamification elements like point values to student-used blocks in the Scratch Editor and tying points to assessment criteria designed by teachers. It also vastly overhauls the architecture, migrating the back end server-side code to Node.js and improving the way the SAGE server talks to the various client-facing aspects of SAGE by sending and exposing only the necessary resources. With a fresh new repository and plenty of additional documentation, our feature and infrastructure additions pave the way for robust future development work.

References

[1] Papert, S. (1980). Mindstorms. New York: Basic Books.

[2] Wing, Jeannette M. "Computational Thinking." Communications of the ACM 49.3 (2006): 33.

[3] Weintrop, David, Nathan Holbert, Uri Wilensky, and Michael S. Horn. "Redefining Constructionist Video Games: Marrying Constructionism and Video Game Design." Constructionism 2012 (2012): 645-649.

[4] Mcnamara, Danielle S., G. Tanner Jackson, and Art Graesser. "Intelligent Tutoring and Games (ITaG)." Gaming for Classroom-Based Learning (2010): 44-65.

[5] "About Scratch." Scratch - Imagine, Program, Share. Web. 20 Dec. 2016.

[6] "Stack Overflow Developer Survey 2016 Results." Stack Overflow. Web. 21 Dec. 2016.

[7] Greene, Jeffrey, and Roger Azevedo. "Adolescents' Use of Self-Regulatory Processes and Their Relation to Qualitative Mental Model Shifts While Using Hypermedia." Journal of Educational Computing Research 36.2 (2007): 125-48.

[8] Bandura, A. (2000). Self-efficacy: The foundation of agency. Control of human behavior, mental processes, and consciousness: Essays in honor of the 60th birthday of August Flammer. Mahwah, NJ: Erlbaum.

[9] Craig, Scotty D., Arthur C. Graesser, Jeremiah Sullins, and Barry Gholson. "Affect and Learning: An Exploratory Look into the Role of Affect in Learning with AutoTutor." Journal of Educational Media, 01 Oct. 2004. Web. 21 Dec. 2016.

[10] Davis, E. A., and J. S. Krajcik. "Designing Educative Curriculum Materials to Promote Teacher Learning." Educational Researcher 34.3 (2005): 3-14.

[11] Lee, Michael J., Andrew J. Ko, and Irwin Kwan. "In-game Assessments Increase Novice Programmers' Engagement and Level Completion Speed." Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research - ICER '13 (2013)

[12] Scratch 2.0 Project Editor Repository (2016, October). https://github.com/LLK/scratch-flash

[13] SAGE-Scratch Code Repository (2016, December). https://github.com/cu-sage/sage-scratch

[14] SAGE-Scratch Wiki (2016, December). https://github.com/cu-sage/sage-scratch/wiki

[15] A. Geoffrey Swab. “The Effects of Cooperative and Individualistic Learning Structures on Achievement in a College-level Computer-aided Drafting Course.” Dissertation Submitted to the Faculty of the Virginia Polytechnic Institute and State University in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Curriculum and Instruction. June 26, 2012.

[16] Lytle, Ryan. “Computer Science Continues Growth on College Campuses.” 12 July 2012. [Web.] http://www.usnews.com/education/best-colleges/articles/2012/07/12/computer-science-continues-growth-on-college-campuses. Visited 21 Dec 2016.

[17] Seiter, Linda and Foreman, Brendan. “Modeling the Learning Progressions of Computational Thinking of Primary Grade Students.” John Carroll University, Department of Mathematics and Computer Science. Undated.

[18] Pava, Jairo. “Formative SAGE Assessments.” Columbia University. COMS E6901 Final Paper. Spring 2016.

[19] Bender, Jeff. “Tooling Scratch: Designing a Collaborative Game-Based Learning System to Infuse Computational Thinking within Grade 6-8 Curricula.” Columbia University. COMS E6123 Midterm Paper. 7 March 2015.

[20] Google. (2016, May). Blockly Demos Code Editor. Retrieved from Blockly: https://blocklydemo.
appspot.com/static/demos/code/index.html

[21] Google. (2016, May). Google Developers. Retrieved from Blockly: https://developers.google.com/blockly/

[22] Lee, J. J. & Hammer, J. (2011). "Gamification in Education: What, How, Why Bother?" Academic Exchange Quarterly, 15(2).

[23] SAGE-Editor Code Repository (2016, December). https://github.com/cu-sage/sage-editor

[24] SAGE-Frontend Code Repository (2016, December). https://github.com/cu-sage/sage-frontend

[25] SAGE Code Repository (2016, December). https://github.com/cu-sage/sage

[26] S. D´Mello et al. "Exploring the Relationship between Novice Programmer Confusion and Achievement." (Eds.): ACII 2011, Part I, LNCS 6974, pp. 175–184, 2011.

[27] Kickmeier-Rust, M.D. and Albert, D. "Micro-adaptivity: protecting immersion in didactically adaptive digital educational games." Department of Psychology, University of Graz, Graz, Austria. Journal of Computer Assisted Learning (2010), 26, 95–105