# Formative SAGE Assessments: Midterm Progress Report

Jairo Pava

COMS E6901, Section 14

Spring 2016

## 1.0 Introduction

This document will describe the technical work completed up to this point for the Formative SAGE Assessment project. Section two of this document will describe the design and implementation of the assessment language. Section three will describe the assessment server that is used to parse Scratch project and assessment files to evaluate student SAGE submissions. Section four will describe remaining work.

## 2.0 Assessment Language

The assessment language is to be used by teachers to build automated assessments of student SAGE submissions. The language will be block-based, just like the one students use in Scratch. The types of assessments that teachers will automate are largely inspired from Harvard Graduate School of Education's Creative Computing Scratch Curriculum. The curriculum defines Scratch projects for middle school students and assessment guidance for teachers that grade student submissions.

### 2.1 Design

Figure 1 illustrates the general structure of an assessment using the assessment language. An assessment consists of an expect block that connects horizontally to an actual block. The actual block is a block that describes an action to be verified, like when a user pressed a key on the keyboard. The actual block is also horizontally connected to an assert block. The assert block describes the type of assertion that will be made on the action. The assert block is horizontally connected to the matcher block. The matcher block describes the expectation that is being evaluated (ie. a sprite moves 10 steps).

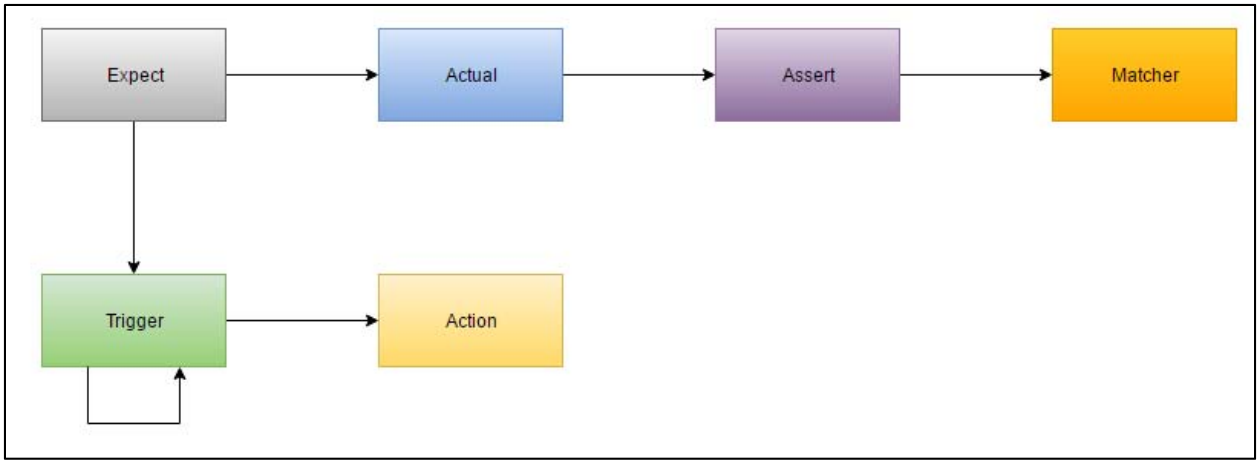


*Figure 1 Assessment Language Structure*

The expect block is vertically connected to one or more triggers. These are tasks that should be invoked when a test either passes or fails. The triggers are horizontally connected to an action block that describes the action to be taken when the trigger is invoked.

**2.2 Implementation**

Figure 2 illustrates the blocks implemented for expect, assert, trigger, and action block types. The colors of the blocks in Figure 2 correspond to the colors of the block types in Figure 1. The assert blocks allow the teacher to specify whether an action should or should not happen.

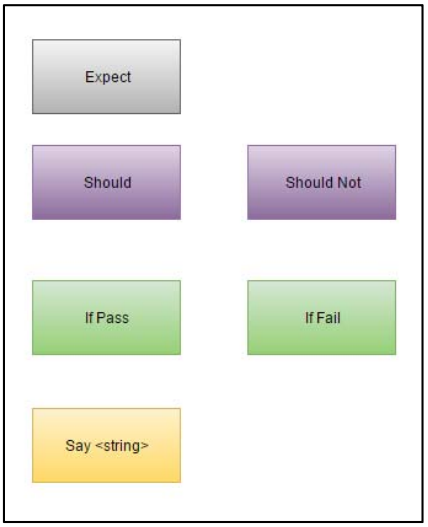


*Figure 2 Block Instances*

The triggers can be used to take action when an assessment passes or fails. And the action that may be taken when a trigger is invoked, is to say a message to the dashboard that the student can view while working on their project.

Figure 3 illustrates the actual and matcher blocks. These blocks must be grouped together as illustrated by the inner rectangles in the figure. Otherwise, there will be an error while executing the test. The topmost inner rectangle in the figure illustrates a sprite block that can be used to point out a specific block in the student's assessment. The respective matcher blocks allow an assessment to assert the presence and location of the sprite.

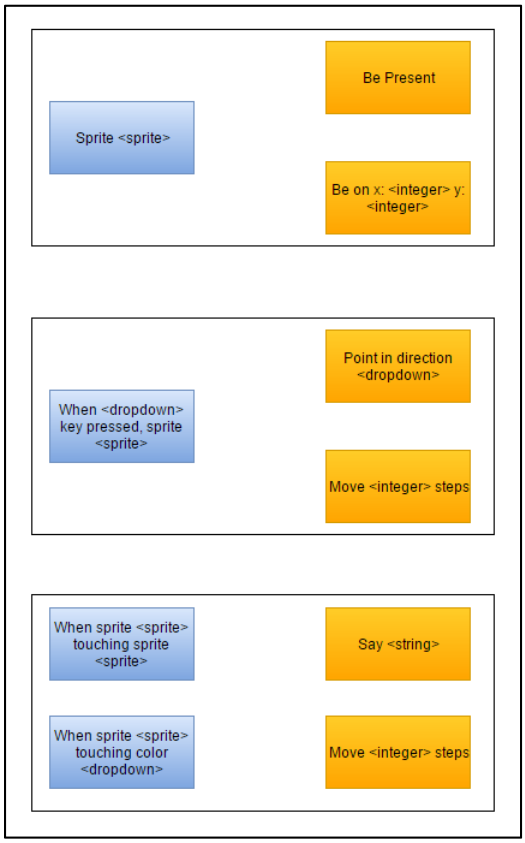


*Figure 3 Actual and Matcher Blocks*

The inner rectangle in the middle of the figure illustrates a block that may be used to describe keyboard presses. The respective matcher blocks describe actions that the sprite will be expected to perform as a response to those key presses.

The last inner rectangle in the figure illustrates blocks that describe the interaction of sprites with other sprites or with other colors on the Scratch stage. The respective matchers illustrate blocks that may be used to describe actions that the sprite will be expected to perform.

### 2.3 Editor

Figure 4 demonstrates a screen capture of the editor that was created to allow teachers to connect blocks that represent the automated assessments. The editor consists of two columns. The leftmost column includes links that describe the block types. When a user clicks on those links, a pop-up is displayed which allows the user to select a block that is of the type that was described by the link. In the figure, the user has clicked on the matcher block type and is presented with matcher blocks.

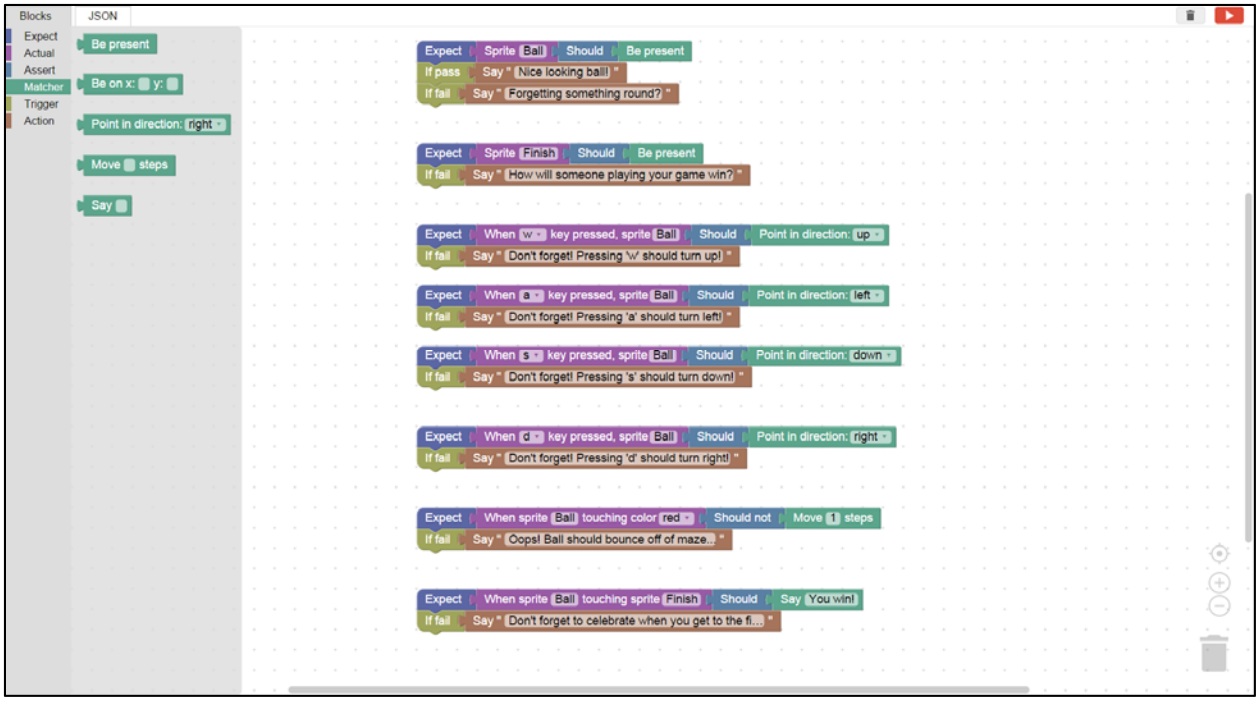


*Figure 4 Assessment Editor*

The rightmost column may be used to place and connect blocks. The editor will only allow the correct connection of two blocks. For example, only matcher blocks may be connected to assertion blocks. Some of the blocks contain textboxes that the user must fill out or drop down

menus that must have a pre-created menu item selected before the assessment automation is completed. The figure illustrates automation of all assessments suggested by the Creative Computing Curriculum for the Maze Starter game.

Once the assessments are automated, the user will be able to upload them to the assessment server. This part will be completed as part of the pending work. In the meantime, a user will be able to click on the tab mislabeled “JSON” to view the XML that represents the blocks and upload that manually to the assessment server.

## 3.0 Assessment Server

An HTTP API was created using GoLang that allows uploads of Scratch projects and assessment files. The API may also be used to execute assessments against uploaded files. Uploaded items and assessment results are stored in a database, powered by MongoDB, for later retrieval and analysis.

### 3.1 Scratch Project Parser

Figure 5 illustrates a screen capture of the Postman tool interacting with the API. Postman is a plugin for the Chrome web browser that enables a user to execute HTTP requests against an HTTP API. The screen capture demonstrates an HTTP POST request to the /project endpoint. The body of the request includes the JSON representation of the student’s assignment. This JSON is automatically generated by the scratch editor. The API responds with a response code of HTTP 202 to indicate that the JSON was successfully uploaded. It also responds with an ID in the HTTP response body that indicates a unique ID that was generated for the project.

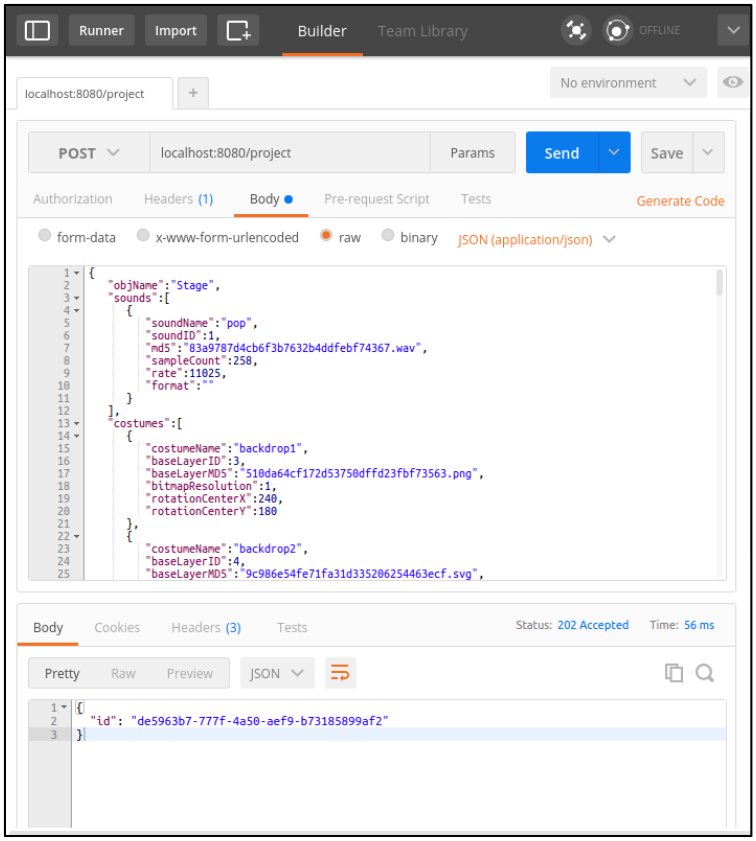


*Figure 5 Project Endpoint*

When a Scratch JSON document is uploaded, the Assessment server parses the document, extracts the blocks associated to sprites and stores them in string data form. For example, the series of Scratch blocks illustrated in Figure 6 would be parsed into the following string:

*whenGreenFlag doWaitUntil touching: Ball say:duration:elapsed:from: You win! 2*


*Figure 6 Scratch Example*

### 3.2 Assessment Parser

Figure 7 provides a screen capture of the Postman tool being used to interact with the assessment endpoint of the API. An HTTP POST request is made to the assessment endpoint

with an XML representation of the assessment. The XML is automatically generated by the Assessment editor described in a previous section.

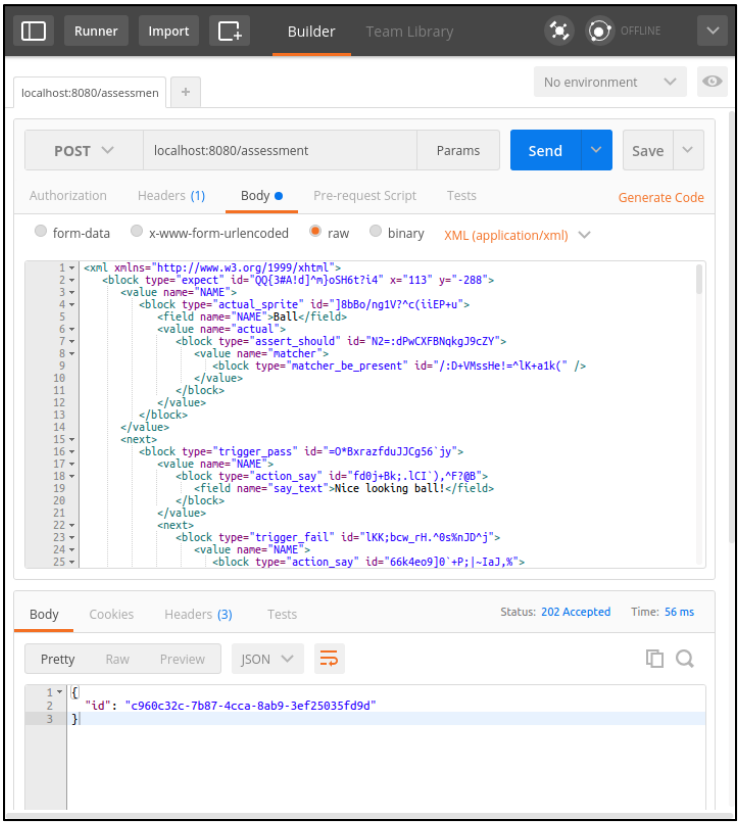


*Figure 7 Assessment Endpoint*

The API responds with an HTTP 202 response code to indicate that the assessment has been successfully uploaded. It also responds with an ID in the HTTP response body that is assigned to the assessment.

Once the assessment is uploaded, the assessment server parses the XML and stores it in a format that it can later read to perform the assessments. Like the Scratch projects, assessments are also stored in the database for later retrieval and analysis.

### 3.3 Assessment Runner

Figure 8 demonstrates a screen capture of the Postman tool being used to execute an assessment against a Scratch project file. An HTTP GET request is used to execute the

assessment. The URI of the GET request takes the following form: /project/{id}/assessment/{id}. The first id corresponds to the ID that the API returns when a project is uploaded. The second id corresponds to the ID that the API returns when an assessment is uploaded.

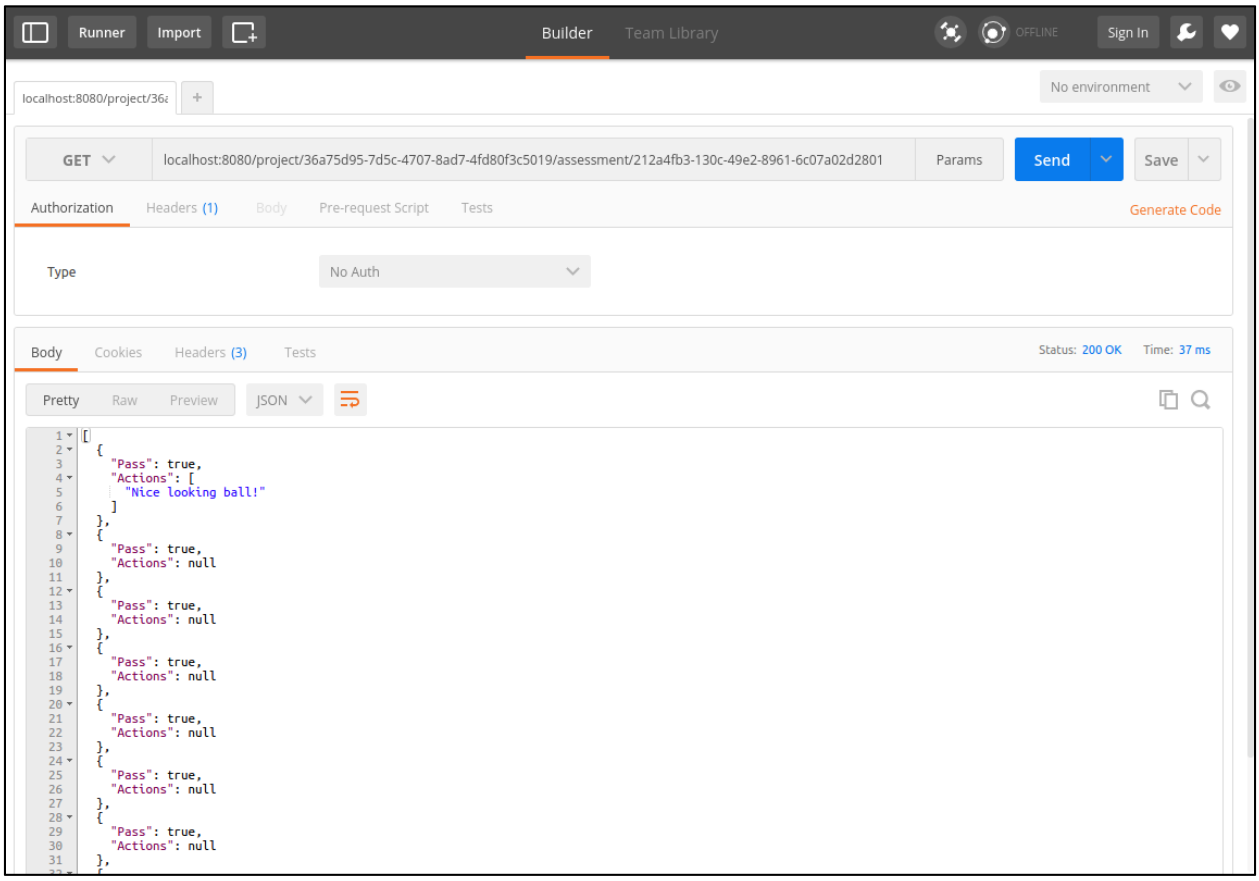


*Figure 8 Execute Assessment Endpoint*

The API returns an HTTP 200 response code to indicate the project and assessment were found by their IDs and that the test was executed successfully. The HTTP response body includes the Pass or Fail results of the assessment.

The assessment server currently performs the execution of the test by pattern matching against the scripts from the scratch project file to find blocks that would lead to the behavior asserted by the assessments.

## 4.0 Pending Work

The following section describes pending technical work.

### 4.1 Assessment Evaluation

Assessment evaluation currently takes place by performing pattern matching against the scripts in the Scratch project file that were stored as strings. However, by storing the Scratch project files as strings, it is difficult to interpret the scripts to find semantic meaning that would enrich the kinds of assessment that can be written. An alternative that is currently being explored is defining a lexicon and grammar that represents the scripts. The scripts can then be stored in a tree-like format that may be walked during the assessment evaluation process to perform better pattern matching.

### 4.2 Scratch Editor

The Scratch editor needs to be updated to periodically upload Scratch project files to the assessment server and retrieve the assessment results. Once the results are retrieved, they should be displayed to the user in a dashboard. The dashboard will help guide the student towards the completion of the assignment based on the results of the assessments that the teacher has provided.

### 4.3 Assessment Language and Editor

Additional blocks must be introduced to the assessment language that enable a teacher to write assessments to evaluate specific computational thinking concepts. Currently, the assessment language only provides means to test functional aspects of student submissions. But teachers will also be interested in automatic assessment of computational thinking evidence in student submissions such as the use of abstract thinking, data representation, and user interactivity among others.