Programming Behavior Detection 1.1

Final Report

Chao-Yang Lo

COMS E6901, Section 28, Spring 2018

Abstract

Intelligent Tutoring system of Social Addictive Gameful Engineering (SAGE) project has been partially implemented. However, the Scratch Analyzer, the main software to detect and collect user behavior data, has not yet been fully integrated to current SAGE node. Therefore, all detection data is stored and analysis on local machine. The objective of this project is to integrate and deploy the current Scratch Analyzer to SAGE system so that not only the whole Intelligent Tutoring system can be run on server but also it provides a better workflow for the Hinting system. Also, the selected technical tools, implementation details and the reminders for future developers are also described in this report.

1. Introduction

This report introduces my contribution to this project through enhancing the Intelligent Tutoring system of Social Addictive Gameful Engineering (SAGE) [1] project. The main objective is to develop an infrastructure and workflow of student behavior detection related modules.

As of 2006, Prof. Jeannette Wing has addressed the importance of computational thinking [2]. Computational thinking is considered as a fundamental skill which involves solving problems, designing systems. Therefore, computational thinking should be required as one of every child's analytical abilities. A visual programming language Scratch [3] is developed by MIT Media Lab which aims for helping 8 to 18-year-old children to develop computational thinking and programming skill with interactive stories, games, and animations. The graphical interface of Scratch provides a set of figures, actions, and sound effects. It encourages children to share, reuse and combine different elements to create their own stories and develop the ability of computational thinking at the same time.

SAGE is designed and developed to provide a game-like environment for students at grade 6-8 to develop computational thinking skills [1]. The adaptive features and feedback system can help students optimize their learning experience and hence get "addicted" as other non-educational video game. Also, SAGE also provides a set of functionalities for teachers and researchers. Therefore SAGE equips teachers with the resources to help students succeed. SAGE is a web application implemented in Node.js. SAGE Assessment Server, or SAGE Node, is the main part of SAGE functionality engine. It provides the assessment of student performance and displays learning related statistics.

To analyze the student behavior, Scratch Analyzer is built by Jeffrey Bender as another major component of SAGE project. It analyzes the log file recorded by SAGE node in a JSON-like format called se file. It is implemented with Java.

In this project, we will use the game data which records the process when a student plays a game and implement the functionalities for the system to directly interact with the server in the whole analysis process.

Chapter 2 will introduce the related works and literatures about the project. The overview and architecture of the data stream and workflow will be mentioned in Chapter 3. The assumptions we made in this project and the limitation we are confronting are described in Chapter 4, while Chapter 5 gives details of technology selection and implementation with advice for future developers in this project. Potential future works that extend this project will be in Chapter 6 and Chapter 7 concludes this report.

2. Related Works

2.1 Modeling How Students Learn to Program

Piech et al. [4] proposed a series of program distance metric of program snapshots from different students according to natural language processing based features, API call patterns and abstract syntax tree of compiled code. A modeling progress is also proposed to provide a Hidden Markov Model (HMM) assumption when analyzing data. With machine learning algorithm, they were able to recognize the students' learning patterns with clustering and HMM estimation with EM algorithm.

2.2 Using learning analytics to assess students' behavior

Blikstein [5] analyzed students' coding behavior by processing programming logs in XML, which is produced by the modified IDE provided for studying to code a programming task. The log contains information about users' actions including key presses, button click, compilation attempts etc. He proposed a canonical coding strategies by observing action patterns and therefore was able to summarize students' programming profile. Also, he analyzed the relation of compilation attempts and code size is a inverse parabolic shape.

2.3 Gameplay as Assessment, Analyzing Event-stream Player Data

Owen et al. [6] proposed a game-based assessment model, which might be used as any game-based learning research project. This includes two parts: Game design process and Assessment process. Game design process defines the main component and game-flow of their game about regenerative medicine. Assessment process defines the indicators of real learning and the analysis of players' profile. Also, a pre-post assessment test is the main evaluation tool for quantifying the gain of knowledge learned.

2.4 Conditions of Learning in Novice Programmers

Perkins et al. [7] analyzed the learning pattern of youngsters when learning to programme in BASIC or LOGO. A list of learner types is also proposed in this paper. Movers, stoppers, extreme movers and tinkerers are the four typical types of learner. Examples of dialogs when student programming was given to demonstrate the behavior of each type of students. Besides, an analysis of codes generated by students is also conducted to give the methodology of

classification of different students. This research helps the team developing machine learning modules to mock student behavior and use the mocked student behavior to develop classifier, which can later to be used on real student data and annotate student learning types.

3. Architecture

In this chapter, we explain the architecture of the whole project, including a brief description of technical tools and functionality of each model.

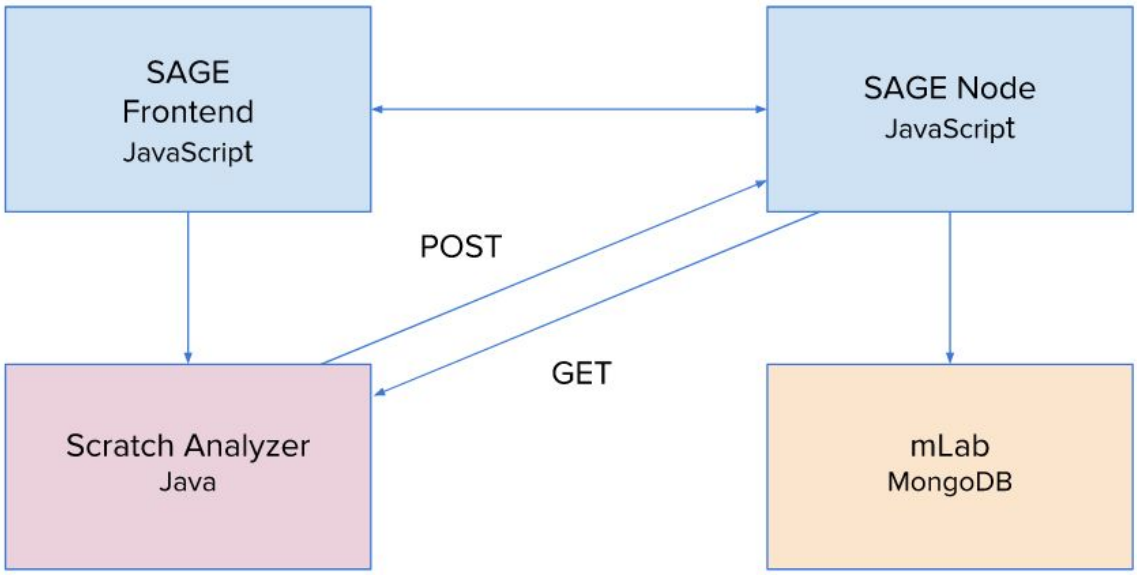


Fig. 1 System architecture with implemented

Fig. 1 shows the structure of the project. SAGE Frontend is the integration of Scratch learning environment that is used by real students to learn to programme by playing games. It is an important module in the application which provides the direct contact with users. However, this is not the module we focus on in this project. SAGE-Node, as known as SAGE Assessment

Server interacts with a MongoDB server called mLab using Node.JS interfaces. Also, SAGE Node provides RESTful APIs using express.JS which allows the users to update the database with standard HTTP requests. It is the main backend engine of SAGE project. mLab is the MongoDB based database server deployed in external hosting server. It contains several databases and collections. In this project, we will only focus on the 'sage-node' database. Scratch Analyzer is a tool to analyze the archived data of game record. There are three components in the analyzer: extractor, dispatcher, and traverser. The most important component is the extractor. It extracts useful operations for machine learning modules from the user history on a line-by-line basis. We switch the source of input and output game files by building the RESTful consuming functionalities using Java libraries and replacing the file names with Http GET or POST APIs.

4. Assumptions and Limitations

In the project, all the modifications are done on the database “sage-node” in mLab. This is not the active database in use in the running system. We work on this database because only the connections to sage-node database are implemented in the current version of SAGE-Node, whereas the sage-login is the active database used by students. However, as a proof of concept, we still decide to build the system on sage-node because not only it does not affect the current online system but the connection is relatively independent of the RESTful functionality. Even if we want to put the system online in the future, the major code can still be reused with modification of the connection related codes.

Also, the current scratch analyzer works only on some formats of the game records and the game format is yet well-defined. Therefore, it is necessary to generate some virtual games in the database to allow the whole process works.

5. Implementations

In this chapter, we will introduce the implementation of each user story in details, including the programming language, libraries and sample implementation code snippets.

5.1 Scratch Analyzer to Assessment Server Connectivity

The first step in this user story is to define the data we will be using. As the game data that the current server collects is not yet has a standard form, we use the sample game data of Scratch Analyzer as our game source data. Therefore, we started with a python code which reads the

sample game file and create a sample game into MongoDB game collection with this JSON file, represented as "gameJSON" field.

```
{
        "objName": "Stage",
        "variables": [{
                        "name": "CLONE COUNT",
                        "value": 0,
                        "isPersistent": false
                },
                {
                        "name": "MODE",
                        "value": "auto",
                        "isPersistent": false
                },
                {
                        "name": "DRAWING",
                        "value": "false",
                        "isPersistent": false
                },
                {
                        "name": "MANUAL PHASE",
                        "value": "TRUNK",
                        "isPersistent": false
                },
                {
                        "name": "Happy with trunk",
                        "value": "false",
                        "isPersistent": false
                },
```

Fig. 2 Example of input game JSON file

```
import json
from pymongo import *
import bson
from bson import ObjectId
from bson import *
from bson.json_util import *

gameID = '5accd7feec89a87424c43abd'
client = MongoClient('mongodb://sage-node:sage-node@ds161210.mlab.com:61210/sage-node')
db = client['sage-node']
collection = db['games']

game1 = open('game1.json').read()
game2 = open('game2.json').read()

testg = dict()
testg['gameID'] = ObjectId(gameID)
testg['Text'] = 'Test game'
testg['gameJSON'] = [game1, game2]

result = collection.insert(testg)
print(result)
```

Fig. 3 Python script to create a new game with designated game JSON file

Notice that although the game is named as “gameJSON,” it is conceptually not JSON file at all. Although the format is indeed compatible with JSON, it is conceptually not key-value pair data structure. Instead, the order of the context lines is important as it reflects the time series of user actions during the game. Also, the indentation should also be preserved for Scratch Analyzer to parse it. Therefore, instead of using a JSON parsing library, the game file should be treated as a pure text file with all spaces and tabs preserved in its original order.

The next step is to update related RESTful APIs within SAGE-Node. Current SAGE-Node is implemented in NodeJS. Especially, the RESTful APIs are powered by express.js and the connection between MongoDB server is based on mongoose library. In this architecture, it is necessary to redefine the document schema in mongoose, for mongoose needs an explicit definition of a data schema to recognize all fields when interacting with MongoDB server.

Fig 4. shows the updated schema. Text and extractedGameJSON are the two fields we added. Text is the field for testing purpose as we can easily search the document in mLab and extractedGameJSON is the field we plan to store the game data processed by Scratch Analyzer, especially the extractor component. The modification of the schema is defined in games.js in sage-node/app/models.

```
var GameSchema = new Schema({
  gameID: {
    type: ObjectId
  },

  studentID: {
    type: ObjectId
  },

  objectiveID: {
    type: ObjectId
  },

  Text: String,

  extractedGameJSON : [],

  sprites : {
    type : []
  },
  gameJSON : [],

}, {
  toObject: {
    virtuals: true
  },
  toJson: {
    virtuals: true
  }
});
```

Fig. 4 The new Schema for the game collection

After the schema is redefined, we want to build a series of RESTful APIs on SAGE-Node side. All the RESTful APIs we built are listed in the following and the usage, functionality and implementation details are summarized in tables.

Usage	`/games/:gameID`
Example	`http://localhost:8081/games/5accd7feec89a87424c43abd`
Type	`GET`
Functionality	Send a GET request with gameID and fetch the whole game document
Return type	String representing a game is returned with the response
Implementation	Using findOne with gameID as matching parameter

Table 1 Summary of GET interface to get game File

The first one is the GET API. When SAGE-Node receives this request, it calls the function `fetchGame` in Game controller. The corresponding `fetchGame` function in Game service will be evoked to get a document with the designated gameID, which is the form of MongoDB ID and it returns the document in the BSON format represented as a string.

Usage	`/games/updategame/:gameID` with `extractedJson` field in data section
Example	`http://localhost:8081/games/updategame/5accd7feec89a87424c43aac`
Type	`POST`
Functionality	Send a POST request with gameID and update extractedGameJSON field
Return type	String representing the gameID which is successfully updated
Implementation	Using findOneUpdate with gameID as matching parameter. Updated content located in extractedGameJSON field of the document

Table 2 Summary of POST interface which updates

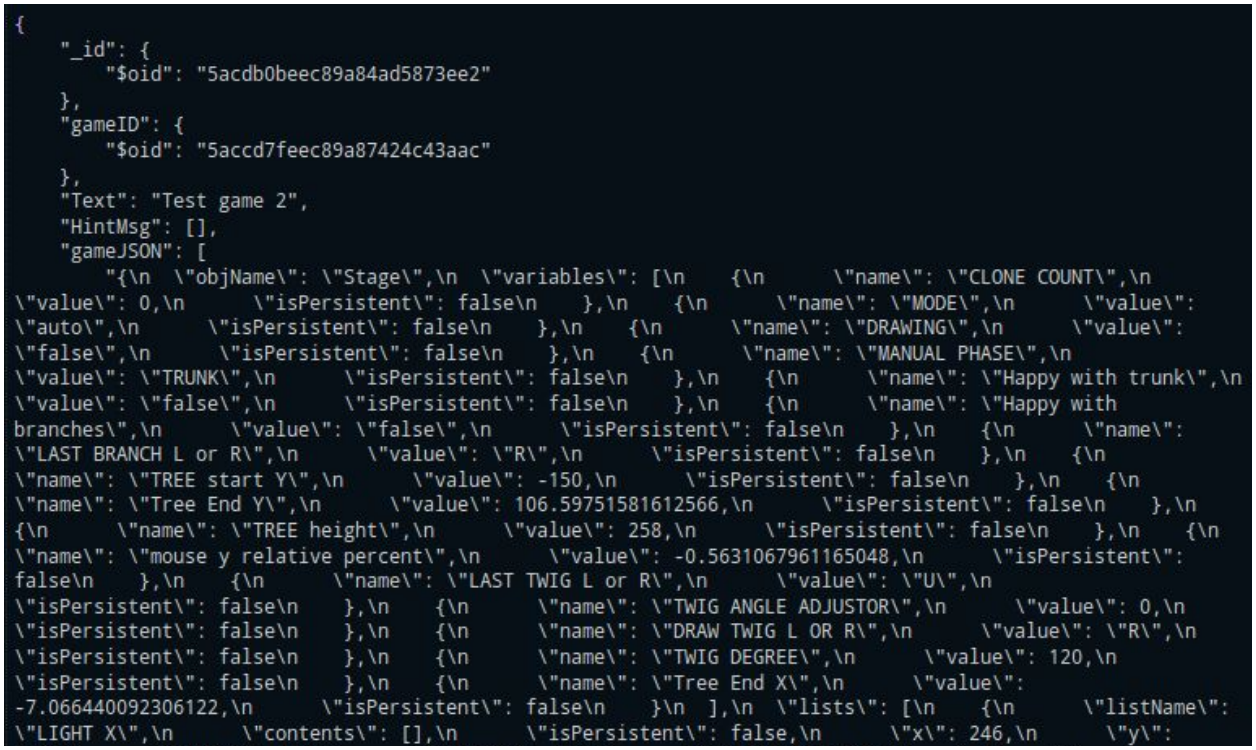
The second one is the POST interface which allows users to update the `extractedGameJson` field in the document with designated gameID in the game collection. When SAGE-Node receives this request, `updateGameSE` in the controller will collect the gameID and the `extractedJSON` field in the request body and calls the `updateGame` function in game service. MongoDB driver `findOneAndUpdate` then appends the `extractedJSON` file to the document with this gameID. If the update is successfully conducted, the gameID will be returned as a response to the request.

```
import requests
import json

data = {"extractedJSON": "<EXTRACTED JSON FILE>"}
data_json = json.dumps(data)
headers = {'Content-type': 'application/json'}
url = 'http://localhost:8081/games/updategameinmemory/5accd7feec89a87424c43aac'

response = requests.post(url, data=data_json, headers=headers)
print(response.text)
```

Fig. 5 Simple Python script consuming POST API

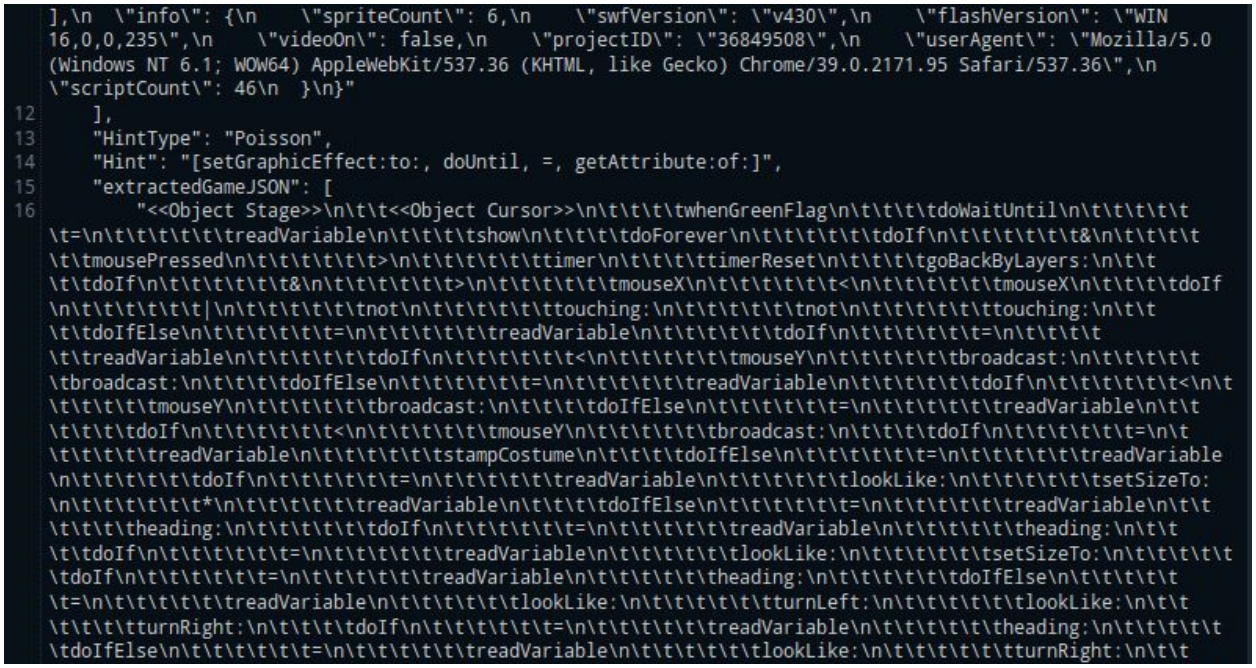
To demonstrate the usage of the POST API, we attached a simple python script, as shown in Fig. 5, which consumes the POST API with designated field assigned in the data section of a POST request.

Also, to help understand the results of applying such APIs. We first show the before and after the Scratch Analyzer directly consumes the GET and POST APIs. The detail of such implementation will be described in later section. Fig. 6 shows the original game document structure with "gameJSON" field. After running the Scratch Analyzer with RESTful API access, the extracted game file is successfully attached to the "extractedGameJson" field, as shown in Fig. 7.



```
{
    "_id": {
        "$oid": "5acdb0beec89a84ad5873ee2"
    },
    "gameID": {
        "$oid": "5accd7feec89a87424c43aac"
    },
    "Text": "Test game 2",
    "HintMsg": [],
    "gameJSON": [
        "{\n  \"objName\": \"Stage\",\n  \"variables\": [\n    {\n      \"name\": \"CLONE COUNT\",\n
\"value\": 0,\n      \"isPersistent\": false\n    },\n    {\n      \"name\": \"MODE\",\n      \"value\":
\"auto\",\n      \"isPersistent\": false\n    },\n    {\n      \"name\": \"DRAWING\",\n      \"value\":
\"false\",\n      \"isPersistent\": false\n    },\n    {\n      \"name\": \"MANUAL PHASE\",\n
\"value\": \"TRUNK\",\n      \"isPersistent\": false\n    },\n    {\n      \"name\": \"Happy with trunk\",\n
\"value\": \"false\",\n      \"isPersistent\": false\n    },\n    {\n      \"name\": \"Happy with
branches\",\n      \"value\": \"false\",\n      \"isPersistent\": false\n    },\n    {\n      \"name\":
\"LAST BRANCH L or R\",\n      \"value\": \"R\",\n      \"isPersistent\": false\n    },\n    {\n
\"name\": \"TREE start Y\",\n      \"value\": -150,\n      \"isPersistent\": false\n    },\n    {\n
\"name\": \"Tree End Y\",\n      \"value\": 106.59751581612566,\n      \"isPersistent\": false\n    },\n
{\n      \"name\": \"TREE height\",\n      \"value\": 258,\n      \"isPersistent\": false\n    },\n    {\n
\"name\": \"mouse y relative percent\",\n      \"value\": -0.5631067961165048,\n      \"isPersistent\":
false\n    },\n    {\n      \"name\": \"LAST TWIG L or R\",\n      \"value\": \"U\",\n
\"isPersistent\": false\n    },\n    {\n      \"name\": \"TWIG ANGLE ADJUSTOR\",\n      \"value\": 0,\n
\"isPersistent\": false\n    },\n    {\n      \"name\": \"DRAW TWIG L OR R\",\n      \"value\": \"R\",\n
\"isPersistent\": false\n    },\n    {\n      \"name\": \"TWIG DEGREE\",\n      \"value\": 120,\n
\"isPersistent\": false\n    },\n    {\n      \"name\": \"Tree End X\",\n      \"value\":
-7.066440092306122,\n      \"isPersistent\": false\n    }\n  ],\n  \"lists\": [\n    {\n      \"listName\":
\"LIGHT X\",\n      \"contents\": [],\n      \"isPersistent\": false,\n      \"x\": 246,\n      \"y\":
```


Fig. 6 The overview of a game with original gameJSON



```
],\n  \"info\": {\n    \"spriteCount\": 6,\n    \"swfVersion\": \"v430\",\n    \"flashVersion\": \"WIN
16,0,0,235\",\n    \"videoOn\": false,\n    \"projectID\": \"36849508\",\n    \"userAgent\": \"Mozilla/5.0
(Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171.95 Safari/537.36\",\n
\"scriptCount\": 46\n  }\n}"
    ],
    "HintType": "Poisson",
    "Hint": "[setGraphicEffect:to:, doUntil, =, getAttribute:of:]",
    "extractedGameJSON": [
        "<<Object Stage>>\n\t\t<<Object Cursor>>\n\t\t\t\twhenGreenFlag\n\t\t\t\tdoWaitUntil\n\t\t\t\t\t\t
\t=\n\t\t\t\t\t\treadVariable\n\t\t\t\tshow\n\t\t\t\tdoForever\n\t\t\t\t\t\tdoIf\n\t\t\t\t\t\t&\n\t\t\t\t
\tmousePressed\n\t\t\t\t\t\t>\n\t\t\t\t\t\ttimer\n\t\t\t\ttimerReset\n\t\t\t\tgoBackByLayers:\n\t\t
\tdoIf\n\t\t\t\t\t\t&\n\t\t\t\t\t\t>\n\t\t\t\t\t\tmouseX\n\t\t\t\t\t\t<\n\t\t\t\t\t\tmouseX\n\t\t\t\tdoIf
\n\t\t\t\t|\n\t\t\t\tnot\n\t\t\t\ttouching:\n\t\t\t\tnot\n\t\t\t\ttouching:\n\t\t
\tdoIfElse\n\t\t\t\t\t=\n\t\t\t\treadVariable\n\t\t\t\tdoIf\n\t\t\t\t=\n\t\t\t
\treadVariable\n\t\t\t\tdoIf\n\t\t\t\t<\n\t\t\t\tmouseY\n\t\t\t\tbroadcast:\n\t\t\t\tbroadcast:\n\t\t\t\tdoIfElse\n\t\t\t\t=\n\t\t\t\treadVariable\n\t\t\t\tdoIf\n\t\t\t\t<\n\t\t\t\tmouseY\n\t\t\t\tbroadcast:\n\t\t\t\tdoIfElse\n\t\t\t\t=\n\t\t\t\treadVariable\n\t\t\t\tdoIf\n\t\t\t\t<\n\t\t\t\tmouseY\n\t\t\t\tbroadcast:\n\t\t\t\tdoIf\n\t\t\t\t=\n\t\t
\t\treadVariable\n\t\t\t\tstampCostume\n\t\t\t\tdoIfElse\n\t\t\t\t=\n\t\t\t\treadVariable
\n\t\t\t\tdoIf\n\t\t\t\t=\n\t\t\t\treadVariable\n\t\t\t\tlookLike:\n\t\t\t\tsetSizeTo:
\n\t\t\t\t*\n\t\t\t\treadVariable\n\t\t\t\tdoIfElse\n\t\t\t\t=\n\t\t\t\treadVariable\n\t\t
\theading:\n\t\t\t\tdoIf\n\t\t\t\t=\n\t\t\t\treadVariable\n\t\t\t\theading:\n\t\t
\tdoIf\n\t\t\t\t=\n\t\t\t\treadVariable\n\t\t\t\tlookLike:\n\t\t\t\tsetSizeTo:\n\t\t\t\t
\tdoIf\n\t\t\t\t=\n\t\t\t\treadVariable\n\t\t\t\theading:\n\t\t\t\tdoIfElse\n\t\t\t\t
\t=\n\t\t\t\treadVariable\n\t\t\t\tlookLike:\n\t\t\t\tturnLeft:\n\t\t\t\tlookLike:\n\t\t
\tturnRight:\n\t\t\t\tdoIf\n\t\t\t\t=\n\t\t\t\treadVariable\n\t\t\t\theading:\n\t\t\t\t
\tdoIfElse\n\t\t\t\t=\n\t\t\t\treadVariable\n\t\t\t\tlookLike:\n\t\t\t\tturnRight:\n\t\t
```


Fig. 7 Example of extractedGameJSon field after updated

The last step is to extend the functionality of Scratch Analyzer by implementing the RESTful API consuming blocks. Since Scratch Analyzer is written in Java, we decided to use Spring Framework [8] as the main framework to support RESTful consumption. After updating the dependency, we have a nice implemented "RestTemplate" class which supports various RESTful API consumption. By refactoring the "extract" and implementing the "extractFileFromString" function in ScratchExtractor, we are able to switch the source of the input file from the local file system to the response of GET request. The modification is shown in Fig. 8 and Fig. 9

```
/*
RESTFul Style GET
*/
RestTemplate restTemplate = new RestTemplate();
String url = "http://localhost:8081/games/5accd7feec89a87424c43abd";
String res = restTemplate.getForObject(url, String.class);

JSONObject jj = new JSONObject(res);
JSONArray games = jj.getJSONArray("gameJSON");
for (int i=0; i<games.length(); i++) {
        System.out.println("Game: " + i);
        String game1str = games.getString(i);
        extractFileFromString(game1str, 99990 + i, ""+(99990+i));
}
writeProjectsInMemory(howToExtract);
```

Fig. 8 Modification in `extract` function of Scratch Extractor

```
private void writeProjectsInMemory(boolean howToPrint) {
        if(!howToPrint) {
                Writer writer = null;
                Set<Entry<Integer, ArrayList<Tree<Block>>>> set = userProjects.entrySet();
                Iterator<Map.Entry<Integer, ArrayList<Tree<Block>>>> iterator = set.iterator();
                while (iterator.hasNext())
                {
                        Map.Entry<Integer, ArrayList<Tree<Block>>> me = (Map.Entry<Integer, ArrayList<Tree<Block>>>)iterator.next();
                        Path userPath = Paths.get(outputDir.toString(), "/", me.getKey().toString());
                        try {
                                Files.createDirectories(userPath);
                        }
                        catch(IOException e) {

                        }
                        ArrayList<Tree<Block>> al = me.getValue();
                        for (Tree<Block> project : al)
                        {
                                String extractedTree = project.printTree(1);
                                //Restful POST URL
                                RestTemplate restTemplate = new RestTemplate();
                                //System.out.println("TEST POST");
                                String url = "http://localhost:8081/games/updategameinmemory/5accd7feec89a87424c43aac";

                                String modifiedExtractedTree = "";
                                for (char ch: extractedTree.toCharArray()) {
                                        if(ch == '\n') {
                                                modifiedExtractedTree += "\\n";
                                        }
                                        else if(ch == '\t') {
                                                modifiedExtractedTree += "\\t";
                                        }
                                        else {
                                                modifiedExtractedTree += ch;
                                        }
                                }

                                String requestJson = "{\"extractedJSON\":\"" + modifiedExtractedTree +"\"}";
                                HttpHeaders headers = new HttpHeaders();
                                headers.setContentType(MediaType.APPLICATION_JSON);

                                HttpEntity<String> entity = new HttpEntity<String>(requestJson, headers);
                                String answer = restTemplate.postForObject(url, entity, String.class);
                                System.out.println(answer);
                        }
                }
        }
}
```

Fig. 9 Re-implemented `writeProjectInMemory` function

As shown in Fig. 9, the usage of POST API is rather complicated. The first thing to notice is the special handling of the escape character. The backslash character needs to be explicitly preserved so that they can be saved in MongoDB. Also, the user needs to assign the HTTP header in order to successfully call the POST API.

5.2 Programming Behavior Persistence

In this section, we are using a similar concept of developing GET and POST APIs for students collection. Although the project will be run on the 'sage-login' database in the future, it still worths to temporarily implement the functionality in 'sage-node' database, as the only part to replace in the future is the connection to another database. Here are the summaries of the two interfaces implemented in students.js in student controller.

Usage	`/students/:studentID`
Example	`http://localhost:8081/students/582f7c1c1489e46faecd12dc`
Type	`GET`
Functionality	Send a GET request with studentID and fetch the whole student document
Return type	String representing a student is returned with the response
Implementation	Using findOne with studentID as matching parameter

Table 3 Summary of GET interface to get student data

Usage	`/students/updatestudents/:studentID`
Example	`http://localhost:8081/students/updatestudent/582f7c1c1489e46faecd12dc`
Type	`POST`
Functionality	Send a POST request with student and update behaviorType field
Return type	String representing the studentID which is successfully updated
Implementation	Using findOneUpdate with studentID as matching parameter. Updated content located in behaviorType field of the document

Table 4 Summary of POST interface which updates student type

Similarly, we need to update the Student Schema as we did in game Schema. As shown in Fig. 10, the `behaviorType` field is added here to store the programming behavior type, which will later be categorized by the machine learning modules.

```
var StudentSchema = new Schema({
  name: {
    type: String,
    trim: true,
    required: true
  },
  avatarUrl: {
    type: String,
    trim: true
  },
  alias: {
    type: String,
    trim: true
  },
  highscore: {
    type: Number
  },
  behaviorType: {
    type: String
  }
});
```

Fig .10

Again, as a demonstration of consuming these two APIs, we attach simple Python scripts, as shown in Fig. 10 and Fig. 11.

```
import requests
import json

req = requests.get('http://localhost:8081/students/582f7c1c1489e46faecd12dc')

student = req.json()
print(student)
print('Student ID:', student['_id'])
print('Student name:', student['name'])
```

Fig. 11 Consuming student collection GET API

```python
import requests
import json

data = {"behaviorType" : "Mover"}
data_json = json.dumps(data)
headers = {'Content-type': 'application/json'}

url = 'http://localhost:8081/students/updatestudent/582f7c1c1489e46faecd12dc'

response = requests.post(url, data=data_json, headers=headers)
print(response.text)
```

Fig. 12 Consuming student collection POST API

And Fig. 13 demonstrates the result in student collection after running the script in Fig. 11

Fig. 13 Student document after updating via POST API

6. Future Works

There are some possible extensions that can be built based on this project. First, the integration of sage-node and machine learning is the direct possible application. As the RESTful APIs are already available, the machine learning module can communicate with SAGE-Node. After the integration for machine learning module is finished, we hope the system can be used to process and analyze real data. The final goal is to power the hint generating system.

Besides, there is a possibility to develop a researcher interface in affinity space, which provides some flexibilities to manipulate the system and thus allows a researcher to modify the behavior detection system without writing the code.

7. Conclusion

In this project, we built a connection and workflow to integrate the current Scratch Analyzer and SAGE system. This provides an experimental proof of concept to migrate the hinting system completely to work on the server. Also, it provides the flexibility to both upload and download data from applications in different programming languages by unifying the communication to SAGE-Node via RESTful APIs.

References

1. J. Bender, *"Tooling Scratch: Designing a Collaborative Game-Based Learning System to Infuse Computational Thinking within Grade 6-8 Curricula"*, Columbia University, 2015
2. J. M. Wing, "*Computational Thinking"*, Communications of the ACM, 2015
3. Scratch. Retrieved from https://scratch.mit.edu/
4. Piech, Chris, et al. *"Modeling how students learn to program."* Proceedings of the 43rd ACM technical symposium on Computer Science Education. ACM, 2012.
5. Blikstein, Paulo. *"Using learning analytics to assess students' behavior in open-ended programming tasks."* Proceedings of the 1st international conference on learning analytics and knowledge. ACM, 2011.
6. V. E. Owen et al. "*Gameplay as Assessment: Analyzing Event-Stream Player Data and Learning Using GBA (a Game-Based Assessment Model).*" International Conference on Computer Supported Collaborative Learning, 2017.
7. Perkins, David N., et al. "Conditions of learning in novice programmers." *Journal of Educational Computing Research* 2.1 (1986): 37-55.
8. Spring. Retrieved from https://spring.io/