Music Blocks JavaScript Export

Sugar Labs

GSoC 2020 Project Proposal

INTRODUCTORY DETAILS

Full Name

Anindya Kundu

University

I am pursuing *BTech* in *Information Technology* from *Indian Institute of Engineering Science and Technology, Shibpur*, in the academic term *2017-21*.

Email

anindyaak007@gmail.com, anindya.k22@outlook.com

GitHub Username

meganindya

IRC nicknames

meganindya1, pluto

Languages

My first language is *Bengali*. However, I am proficient in *English* and comfortable to communicate in the same.

Location

I am located in *Kolkata, West Bengal, India*. My timezone is *Indian Standard Time* or *UTC+5:30*.

Previous Open-source Experience

I have a brief exposure to open-source development through *Hacktoberfest (2019)*, and an educational open-source program called *Kharagpur Winter of Code (2019)* conducted by *Kharagpur Open Source Society: KOSS*, the open-source club of *Indian Institute of Technology Kharagpur*.

Through *Hacktoberfest*, I learnt the basics of *Git* and *GitHub* while contributing solutions for a few competitive programming problems, in *Java*. During *Kharagpur Winter of Code*, I further improved upon my hold on version control and contributed to three repositories:

1. Competitive Programming Data Structures and Algorithms
2. Moviepedia
3. Home's Magic

I contributed five programs to the *Competitive Programming* repository based on linked lists, binary tree, mathematics, and dynamic programming, all in *Java*.

The second is a movie catalog website in development that uses *themoviedb* API, and is developed in Angular CLI. I fixed a design issue, and restructured and restyled some elements to improve the appearance. I contributed in *HTML* and *CSS* (plus *Bootstrap*).

The third is the one I've contributed most to. It is a home-cooked food delivery website in development, written in *HTML*, *CSS*, *PHP*, *MySQL*. My work included restructuring the *HTML* layout, restyling by improving the *CSS* and improving the design language, updating the CDN versions, restructuring the *PHP* code and files, and refactoring existing *PHP* and *MySQL* code.

Here is a link to my final project report: *KWoC 2019 Project Report - Anindya Kundu*.

Technical Knowledge, Interests, Previous Work
In terms of *Programming Languages*, I am introduced to *Java*, *C*, *C++*, *JavaScript*, *PHP*, *Python*, *SQL*, *Assembly*, *MATLAB*, and *C#*, of which I am most comfortable in *Java*, *C*, and *JavaScript*. I am also briefly introduced to *JQuery*, *React*, and *Angular* frameworks.

I am ambitious about creating/contributing to technologies that make the process of education more efficient by doing away or cutting down tedious processes. As such, I am quite interested in *Web Development* and have been briefly doing so since the last 4 years. Interestingly, my motivation is perfectly in line with this particular project in particular, and Sugar Labs in general.

I have built quite a handful of simple projects, mostly based on *Web Development*. Very recently, as part of my *5th semester Database Management* project, I built a mock *College Database* web application that interacts with a database of students, instructors, courses, and departments. Features included personal information view/edit, and recording attendance and marks per student per course. I used *HTML*, *CSS*, *PHP*, and *MySQL* in the project.

Also to showcase, I built a few *front-end board games*: *Minesweeper*, *Snakes and Ladders*, and *Sudoku*. They are written in *HTML*, *CSS*, and *vanilla JavaScript*, and correspond to *Sugarizer* activities to some extent.

Apart from these, I've built a few simple utility apps like *Scientific Calculator*, *Expenditure Notebook*, *Dice Game*, etc. that aren't listed on *GitHub*. My notable non-web works include *Radar Game* written in *Processing*, *Dinosaur Game* written in *C* (*OpenGL*), an image processing *Route Plotter* in *Python*, and a few *Deep Learning* projects in *Python*.

SUGAR LABS AND ME

I came across *Sugar Labs* from a batchmate of mine, who is a GSoC 2019 participant (with *mlpack*). After *Sugar Labs* was announced as a participating organization in GSoC 2020, I started playing around *Music Blocks* and *Sugarizer* (*desktop* and *web*). Unfortunately, I was a bit stuck with academic engagements. Around mid-March onwards, I started reading the *Music Blocks guide* between the lines while trying them out on the hosted web app, so that I could understand all functionalities of the application. I have started contributing from around the last week of March.

Pull Requests

PR Name, ID	Description	Status
Improve appearance of Wheel Menu (#2175)	General UI enhancement.	Merged
Generate notes to play/save on the fly (#2176)	Fixed issue #2165 (updated pitch values not reflected in grid playback). Created a function to generate the list of notes to play, live, from the state of the blocks (highlighted) in the Phrase Maker.	Merged
Fixed issue of notes being inside repeat block in phrase maker (#2178)	Fixed issue #2051 (Repeats Skipped in Phrase Maker). I figured out there was an ID comparison mismatch; I fixed the comparison. In addition, I fixed an	Merged

	invalid access condition, which was throwing an exception.	
Fix re-sorting issue (#2180)	Fixed issue #2179 (Sorting works only once in Phrase Maker). There is a flag variable whose value I reset on encountering the problem condition.	Merged

Issues Raised

Issue Name, ID	Description	Status
Tour window imperfect positioning (#2172)	The tour window appears at different positions. The window position can be made useful if it hovers around the corresponding button it is describing.	Open
Tour window fullscreen problems (#2173)	UI problems with the tour window in fullscreen mode.	Open
Block Pie Menu position doesn't change with scrolling (#2174)	If the pie (wheel) menu of a block is opened and the canvas is scrolled, the blocks scroll while the wheel doesn't.	Open
Sorting works only once in Phrase Maker (#2179)	In the Phrase Maker, if pitches are modified after sorting such that the sorting order is lost, re-sorting wasn't being allowed.	Closed (by #2180)
Notes improperly restored in Phrase Maker (#2185)	Pointed out by walterbender while discussing PR #2178. In the phrase maker, if the matrix is generated from a block stack containing note blocks or repeat, if new notes are added they are erroneously restored if reopened.	Open

Communication

Since I've been around for only a brief period so far, I don't have extensive communication experience with the community. However, in this shorter span, I've reached out to the *mailing list* a few times: a couple times to clarify about the project and received replies from *James Cameron* and *Sumit Srivastava*, and other times to clarify doubts while trying to solve an issue and received replies from *Walter Bender*. On *GitHub* I've mostly communicated with *Walter Bender* regarding issues and pull requests.

Experience so far

In this brief period, I've mostly been concerning myself with issues with the *Phrase Maker* widget. I've fixed three issues related to it, and encountered one of them in between. Also, on fixing one of them, another issue popped up, pointed out by *Walter Bender*. My impression has been that an open-source job is tedious and involves much more reading and debugging than writing new code. I'd been scrolling through thousands of lines of one of the JavaScript files for hours at a stretch trying to identify the cause of the bugs. However, after hours of effort, when I could resolve the issue, it was very rewarding. I did realize the fact that communication is key. On two occasions I'd a doubt regarding certain functions; after reaching out to the mailing list, I was able to find answers. Also, regarding GSoC projects, I've found that the mailing lists in the community are quite free and helpful. I myself benefitted from two such discussions.

I intend to continue contributing to this community actively, now onwards. In the next few weeks I intend to tackle more bugs, specifically with blocks, after I've resolved a few more visible issues with the *phrase maker*.

PROJECT DETAILS

Introduction

My idea is based on the description provided for "*Export Music Blocks code to JavaScript*". Since a similar project titled "*Turtle Blocks Python Export*" was done by *Marion Zepf* in *GSoC 2013* on *Turtle Blocks* (on which *Music Blocks* is based on), I'd like to take some inspiration for it, while being careful not to loosely copy things. As I understand, the goal of this project is to help users eventually graduate from blocks to a text-based language. Hence, the project essentially requires generation of JavaScript code which corresponds to the block stacks in the canvas, and possibly run the code too.

I'd break down my project idea to have three properties:

1. Generate an independent HTML file that displays the block stacks.
2. Generate an independent and valid JavaScript file whose tokens and structuring correspond to the block stacks.
3. The possibility to run the JavaScript file that would have the same effect as clicking the 'play' button on the HTML file.

In words, when a user exports a project, an *HTML* and a *JavaScript* file (and few other dependent files) will be downloaded. The *HTML* file will display the canvas and the blocks. Beside it, the generated *JavaScript* code will be shown. The canvas will have a *play* button to play the *start* blocks, and another to run the *JavaScript* which should produce the same effect (if the *JavaScript* code is unaltered).

Impact on the Sugar Labs community

The mission of *Sugar Labs* is to stimulate learning. Since *Music Blocks* is a tool to teach the basic concepts of programming to beginners, this project would take the community to a higher ground - beginners can go beyond getting introduced to programming, by 'graduating' to conventional and more expressive text-based languages (like *JavaScript*). The project would give the users a template (with familiar behaviour) to experiment on; users would be able to play around the JavaScript code and make valuable observations, as opposed to having to learn the language by writing programs from scratch.

Primary Objectives

Preparation

- Strip down the existing code to the essential ones for rendering, playing blocks, etc.
- Figure out details about the stored parameters for different types of blocks.
- Implement a way to re-render and play the blocks outside of *Music Blocks*, independently.
- Devise an object-oriented structure to store information about blocks (e.g. behaviour, output token type, possible identifier name, etc.).
- Study libraries that generate *Abstract Syntax Tree* (*AST*) from the *JavaScript* code (for running the code).

Bringing things together

- Create a template *HTML* file and a script to augment additional (block stack) information into the *HTML* file.

- Create scripts to implement the object-oriented structures for the blocks.
- Create a script to convert the *Music Blocks* (inner) block structure to an *AST*, and a script to generate output code from it.
- Create a script to execute the above three and other behaviour like *play*/*stop*.

Finalizing

- Create a script to restructure the files for output.
- Garbage collection, if any.

Optional Features

- Display a guide about basics of programming (e.g. variables, functions, etc.).

Scope of the Project

I want to keep the scope of the project within limits so that I can successfully complete it in time. Therefore, I want to limit my project only to the blocks in the beginner mode. Also, I'd primarily work only on the non-widget blocks. On successful completion of them will only I work on the widgets.

Tech Stack

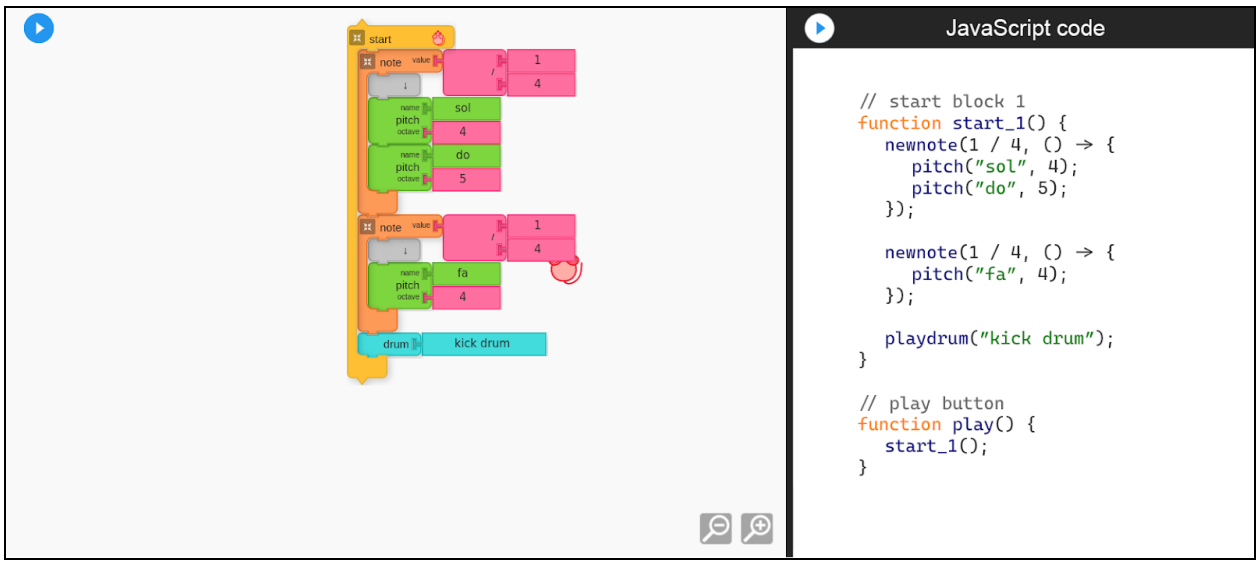
- I intend to develop the project primarily with *JavaScript* (*ES6* or *ECMAScript 2015*). In addition, I'd require the use of *HTML5* and *CSS3* for the basic layout.
- As of now, I intend to use the *Esprima* library to generate *Abstract Syntax Trees* (*AST*) following the *ESTree ES2015* specification.
- To generate code from *ASTs*, I intend to use the *Escodegen* library.

I want to mention that my choice of libraries is not fixated. I've picked the two since they can be run on the browser directly. However, I might look for other options like *Meriyah* and *Astring* too. This shouldn't be a big issue as they all essentially do the same thing.

Project Description and Mockup

User Interface

The basic interface this project would provide is a *Music Blocks canvas*, with only the *play* button and no *bar* or *palette*. The *canvas* would display the block stacks as they were while exporting. Users wouldn't be able to make changes in the structure once exported. This is so because that'd require much more dependencies and would partly become *Music Blocks* itself on the desktop, which is beyond the scope of this project.



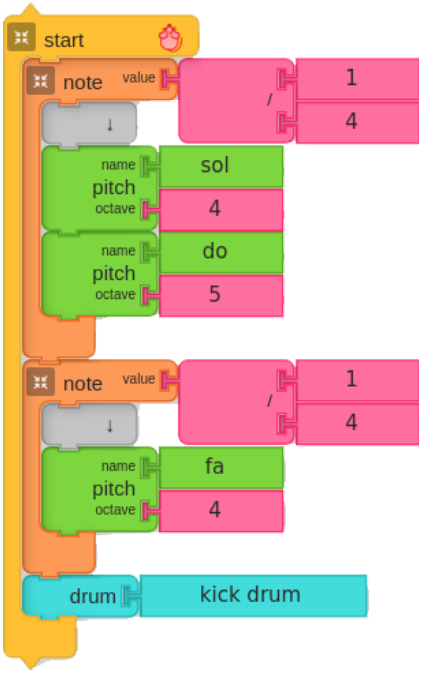
(mock layout)

The project's interface would look somewhat similar to this; also the code for the above representation of the block stack would look like the sample on the right (However I want to safely mention that these are subject to changes during the course of development).

Block Structure

Inherently *Music Blocks* stores the block structures as multidimensional arrays. Each block has an array with five arguments: *ID*, *properties* (array), *X-coord*, *Y-coord*, *IDs of linked blocks* (array).

For example, for the block structure



the inner structure is

```
[
    [
        0, ["start",{"collapsed":false,"xcor":0,"ycor":0,"heading":0,"color":10,"shade":60,"pensize":
5,"grey":185.71428571428572}],
        640,158,[null,1,null]
    ],
    [ 1,  ["newnote",{"collapsed":false}], 654, 199, [0,2,5,9] ],
    [ 2,  "divide", 756, 199, [1,3,4] ],
    [ 3,  ["number", {"value":1}], 842, 199, [2] ],
    [ 4,  ["number", {"value":4}], 842, 231, [2] ],
    [ 5,  "vspace", 668, 231, [1,6] ],
    [ 6,  "pitch", 668, 263, [5,7,8,19] ],
    [ 7,  ["solfege", {"value":"sol"}], 742, 263, [6] ],
    [ 8,  ["number", {"value":4}], 742, 295, [6] ],
    [ 9,  "hidden", 654, 420, [1,10] ],
    [ 10, ["newnote", {"collapsed":false}], 654, 420, [9,11,14,18] ],
    [ 11, "divide", 756, 420, [10,12,13] ],
    [ 12, ["number",{"value":1}], 842, 420, [11] ],
    [ 13, ["number",{"value":4}], 842, 452, [11] ],
    [ 14, "vspace", 668, 452, [10,15] ],
    [ 15, "pitch", 668, 484, [14,16,17,null] ],
    [ 16, ["solfege",{"value":"fa"}], 742, 484,[15] ],
    [ 17, ["number",{"value":4}], 742, 516, [15] ],
    [ 18, "hidden", 654, 578, [10,22] ],
    [ 19, "pitch", 668, 326, [6,20,21,null] ],
    [ 20, ["solfege",{"value":"do"}], 742, 326, [19] ],
    [ 21, ["number",{"value":5}], 742, 358, [19] ],
    [ 22, "playdrum", 654, 578, [18,23,null] ],
    [ 23, ["drumname",{"value":"kick drum"}], 728, 578, [22] ]
]
```

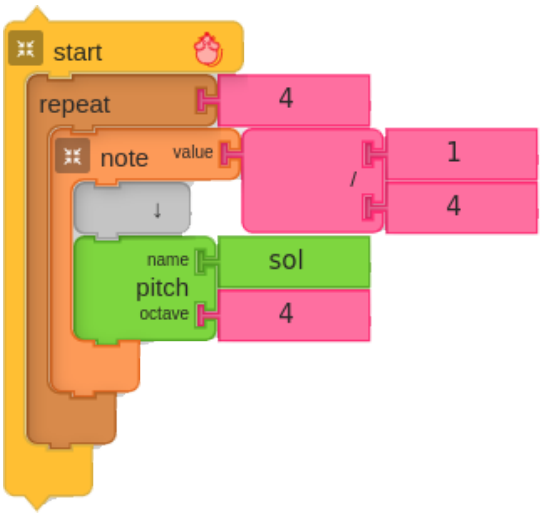
An *Abstract Syntax Tree* (*AST*) will be created from this representation in the *ESTree* format specification which can then be converted to the output *JavaScript* (*ES6*), using the serialization library *Escodegen* (or *Astring*, etc.).

Object Structure

All the blocks in *Music Blocks* can be represented in *JavaScript* by *functions*, *variables*, *constants*, *operators*, *if-else blocks*, *while loops*, and *inline statements*. Hence, there is a need to bother about only a small subset of *ESTree* specifications while creating the *AST*.

I intend to create an object-oriented representation of the blocks by designing a custom *class* that will store all information about each block, and link implementation code (rendering, behaviour, etc.) with output *JavaScript* code (token type, identifier name, arguments, etc.). I'll then parse the original block structure and convert the blocks into objects of the said custom *class*. I'll then create a custom *AST* using the objects only. I'll then parse this *AST* and convert it to the *ESTree* specified form.

Moving back to the blocks, they can have different behaviour. Some can be parameterized functions, some can be functions just to call a list of other functions, while others might be statements. For example, consider the block stack



Here, the *start* block can be represented by a *function* which calls other *functions* one by one as in the block stack:

```
function start_1() {
    callee_1(...);
    callee_2(...);
}
```

The *repeat* block can be represented by a *while loop* which could output like:

```
var n = 4;
while ((n--) > 0) {
    ....
}
```

The *note* block can be represented by a *function* that takes two *arguments* - a *floating point value*, and a list of *callback functions*. The example above could look like:

```
function note(value, fns) {
    var freq = retrieve_pitch_sum_in_hertz(fns);
    play_pitch(freq, value);
}
```

The *pitch* block can be represented by a function with a return value, like:

```
function pitch(name, octave) {
    // calculate frequency value
    var freq = ...;
    return freq;
}
```

Overall, the block stack becomes:

```
// function declarations
function start_1() {
    var n = 4;
```

```
    while ((n--) > 0) {
        note(
            1 / 4,
            [
                pitch("sol", 4);
            ]
        );
    }
}

// for example only
function play(start) {
    start();
}

// call the function
play(start_1);
```

As evident, a good solution would be an *object-oriented* approach: each block would be an *object* (or *instance*) of a *class*, say 'Block', with the *instance variables* to store information about the respective block, and *instance methods* to define the behaviour (e.g. the *pitch function* represented above). The *class* declaration could look like:

```
constructor(bid, bparams, X, Y, lid) {
   this.block_id = bid;
   this.block_params = bparams;
   this.xcoord = X;
   this.ycoord = Y;
   this.linked_ids = lid;
   this.block_type = this.get_block_type(bparams);
}

get_block_type(bparams) {
   return typeof bparams == "String" ? bparams : bparams[0];
}

output_js_token_type() {
   switch(this.block_type) {
       ...
   }
}

output_js_arguments() {
   switch(this.block_type) {
       ...
   }
}

// other instance methods
...

...
```

While parsing blocks, we can fetch all information about the blocks from inherent details used to *render*, *play*, etc. to details about the output code to generate that will be used to create the *AST* mentioned earlier. After the *AST* is created it can easily be converted to the output code. It is important to note that some blocks (e.g. *set master volume*) will be represented by inline statements, e.g. *variable assignment*. Hence, harmony should be maintained between the code to keep it valid (e.g. assignment statements should work on the variable in scope only).

Executing the Generated Code

There is the requirement of mapping the effect producing (e.g. sound playing) *functions* in the output code to their respective behaviour in *Music Blocks*. To solve this a *library of functions* would be created and their implementation will be hidden from the user in an effort to preserve abstraction.

The *AST libraries* (*Esprima*, *Meriyah*) can take care of the *Lexical*, *Syntax*, and *Semantic Analysis*. Thus, if the generated code is altered, users can know if errors have been made. If clear, the code can be executed as is.

PLAN OF ACTION

I plan to actively work on this project four days a week from Thursday - Sunday between IST 12:00 and IST 24:00. I don't yet have a planned activity to do besides this, during the span of the event. However, due to critical conditions arising throughout the globe due to CoVID-19 this year, my schedule might need to move slightly back and forth a little bit. Due to the said cause, my semester has been suspended for the time being, and I'm not sure how things will go about after April 15th. So, considering the conditions, I intend to do things ahead of time to make adjustments for academic engagements if they coincide. I'll regularly keep my mentors informed about such situations.

Preparation Phase

Due to the uncertainty of events this year, I'd start right off with the preparation right from 1st April. Initially I'll start by exploring what the different *69 JavaScript files* in the existing codebase do, and which of them are relevant to my project. Since my project is centred around block stacks and their behaviour, I'll be continuing my contributions especially around them by fixing bugs, in an effort to understand the code related to their behaviour.

If I am selected, I'll then start communicating with my mentors about how certain things work and how they can be extracted to make them work independently outside *Music Blocks*.

Timeline

As mentioned before, since a similar project has been successfully completed in GSoC 2013, I'd take inspiration from the said, and begin my timeline on the lines of it, while deciding activities specific to my project.

June 1 - June 14 (*2 weeks*)	Figure out what code needs to be shared by *Music Blocks* and the exported dependencies to locally achieve the required behaviour, i.e. *rendering* and *playing*. Only cover the modules required to run the blocks. Restructure the modules so as to easily retrieve the required functions when having to execute a block.	
June 15 - June 21 (*1 week*)	Design the *class* definition for only the *rhythm*, *pitch*, and *drum* blocks in the *Music Palette*, while documenting by hand the properties each block needs to store (*parameters*) and what code they'll produce.	
June 22 - June 28 (*1 week*)	Write scripts to *parse* the *block stacks* (mentioned earlier with figure) and create an *AST* of *block objects.* Convert this *AST* to *ESTree* specified format, for serialization libraries to work.	
June 29 - July 3 (*Phase Evaluation 1*)	Some (*rhythm*, *pitch*, *drum*) *block stacks* can now be converted to *ASTs* which can be serialized to get their output code. The rest of the blocks can thereafter follow in a similar fashion.	
July 4 - July 26 (*3 weeks*)	Hand document the properties and output code for the rest of the blocks. Complete the *class* definition for all of them.	
	Week 1	(Music Palette) *meter*, *intervals*, *tone*, *ornament*, *volume* blocks to be dealt with.
	Week 2	(*Programming Palette*) *flow, action, boxes, number, boolean* blocks to be dealt with.

	Week 3	(*Graphics Palette*) *graphics*, *pen*, *media*, *sensors*, *ensemble* blocks to be dealt with as much as possible.
July 27 - July 31 (*Phase Evaluation 2*)	Most *block stacks* can now be exported to *JavaScript* code.	
Aug 1 - Aug 9 (*1 week + 2*)	Strip down the *HTML* to render only the essential *UI* elements (canvas, blocks). Implement the *play* feature to execute the blocks on the canvas locally.	
Aug 10 - Aug 16 (*1 week*)	Write scripts to analyze the (generated) *JavaScript* file (for user made changes). Write the library for the implementation of effect producing block functions (e.g. say *playpitch*(...) produces sound of some frequency). Write the script to run the (generated) *JavaScript* file.	
Most essential objectives have been completed by now.		
Aug 17 - Aug 23 (*1 week*)	Complete finalizing steps, e.g. writing scripts to restructure output files, garbage collection. Test exporting multiple block stacks; fix bugs (if any). Complete minor things (if left). Also, this is a buffer period just in case things shift a little bit.	

Progress Report

I plan to set up a blog to document the progress of my work. As in *Marion Zepf*'s case, she documented her progress through commit messages in an online repository; I plan to follow her approach. If possible, I'll set up a Wiki page too to keep track of my timeline.

Communication

I have already introduced myself to the mailing list, however I haven't used the IRC yet. I intend to more actively participate in these two. Again, it's too early in the process to foresee trouble, but there's a high possibility that I might get stuck somewhere. In case, I stumble around something and the mentors aren't around, I'll indeed rely on the two said platforms first, but if there's no fix, I'll currently suspend the task that I have a doubt in, and move to another task in the meanwhile, rather than wasting time. I'll contact the mentors once they're back.

Post GSoC Plans

I've mentioned previously how the *Sugar Labs* community's motivation and mine coincide, and so I'd definitely keep hanging around and keep contributing even post *GSoC*. As of now, I've done everything only around *Music Blocks*. Irrespective of whether I get selected in *GSoC* this year, I plan to contribute to *Sugar* and *Sugarizer* too. In fact, for starters, I already have a couple of *Sugarizer activity* ideas in mind. I've mentioned earlier that I've built three board games: *Minesweeper*, *Sudoku*, and *Snakes and Ladders*, as hobby projects using *HTML*, *CSS*, and *vanilla JavaScript* only. I'd probably start by refactoring and porting these to *Sugarizer*.

CONCLUSION

Well, now that I've provided a somewhat detailed information about my project and how I'm planning to do it, I want to end this with a little bit of arbitrary information regarding myself. I am currently in my 6th Semester and I'm introduced to *Formal Language and Automata Theory* (*Theory of Computation*) and *Compiler Design*. This project so much seems like a hands-on job for the concepts I've learnt like *Language*, *Grammar*, *Regular Expression*, *Lexical Analysis*, *Parsing*, *Semantic Analysis*, *Intermediate Code Generation*, etc. I think I'll really enjoy doing this project if given the chance to, since I'd get the opportunity of actually using my classroom knowledge in a programming job.

In addition, I'm a tech enthusiast and I love designing 3D models of real-world objects and buildings in my free time. I am somewhat obsessed with design perfection and so you might sometimes find me pointing out too many design issues (kindly, bare with it ;p).

Thanks for reading. Have a great day!