Introduction

Team TBD
Peyton Hartzell, Sarah Nicholson, Mykal Burris, and Kelly Ding

What is Blockly?

Blockly is a library created by Google that adds a visual code editor to both web and mobile apps. The Blockly editor uses interlocking, graphical blocks, similar to Scratch, to represent code concepts like loops, logical expressions, variables, and more. It allows users to apply programming principles without having to worry about minute details such as syntax. We chose to work with Blockly because we wanted to get a better grasp on JavaScript, and one of our team members is very familiar with it. We are also interested in the education of children in computer science early in their school careers, and Blockly helps to do that.

Blockly is for developers while Blockly apps are for students. Blockly is an intuitive, visual way to build code. Blockly is 100% client side, requiring no support from the server and there are no 3rd party dependencies. It is a ready made UI for creating a visual language that emits syntactically correct user-generated code which can then be exported to many programming languages including:

- JavaScript
- Python
- PHP
- Lua
- Dart

A few of Blockly's biggest strengths are as followed:

- Exportable code: Users can extract their block-based programs to common programming languages for a smooth transition to text-based programming.
- Open source: Everything about Blockly is open open source.
- Extensible: One can tweak Blockly to fit one's needs by adding custom blocks for an API or removing unneeded blocks and functionality.
- Highly capable: One can implement complex programming tasks using Blockly.
- International: Blockly has been translated to 40+ languages.

Getting Started

The first thing we did was to navigate to the “Getting Started” page Google for Education provided for Blockly. We then downloaded the source code from Google’s Github and verified that blocks could be dragged around by running the following command in Blockly’s root directory:

firefox demos/fixed/index.html

Next, in order to run the existing unit tests Blockly provides, we had to download the closure-library from Google’s Github. The reason why Closure is needed is because we are going to run and build the uncompressed version of Blockly and the dependency of the Closure Library is needed. Once we downloaded the Closure files, we placed them next to Blockly’s root directory and renamed the directory closure-library. The following is the directory structure we ended up with and what is required to run the tests:

- google
    - blockly
        - blocks
        - core
        - demos
        - generators
        - media
        - msg
        - tests
    - closure-library

Blockly was then able to work in uncompressed mode. Run it by entering the following command:

firefox tests/playground.html

Running Existing Tests

There are two sets of existing unit tests: JavaScript tests and block generator tests.

JS Tests

The JS tests confirm the operation of internal JavaScript functions in Blockly's core. To run this test, we went into the root directory of Blockly and run the following command:

firefox tests/jsunit/index.html

When we ran it, all tests passed and gave us the following output:

> Unit Tests for Blockly [PASSED]
> /home/USERNAME/Documents/blockly-master/tests/jsunit/index.html
> 267 of 267 tests run in 2415.495ms.
> 267 passed, 0 failed.
> 9 ms/test. 238 files loaded.
> .
> 23:36:04.296 Start

Block Generator Tests

Along with the tests to confirm the operation of functions in Blockly's core, each block has its own unit tests. These tests verify that blocks generate code than functions as intended.

Again, to run these tests, we went into the root directory of Blockly and this time, ran the command:

> *firefox tests/generators/index.html*

Running this command brings up a testing window for the web app to validate that the blocks generate code correctly, corresponding with which programming language you choose. Once the window was up, we chose what part of the system to test from the drop-down menu, and clicked "Load". The blocks then appeared in the workspace and XML code corresponding with the blocks appeared in the textbox.

We then checked each programming language in the workspace along with the system we were testing to make sure that syntactically correct code was being outputted. We first ran the generated JavaScript code and in order to make sure it was correct, copied and pasted it in a JavaScript console which outputted “OK” meaning that it passed. Next, we clicked the button to generate Python code and then copied and pasted the code in a Python interpreter which outputted “OK,” meaning it passed. After that, we clicked on “PHP” which generated the PHP code in the textbox. We then copied and pasted the generated code into a PHP interpreter and it outputted “OK,” meaning this one passed as well. Then, we clicked the “Lua” button to generate the Lua code in the textbox. We copied and pasted the Lua code into a Lua interpreter and it outputted “OK,” showing that it passed. The last language supported by Blockly that we tested is Dart. To test it, we clicked on the “Dart” button which generated the code. We then

copied and pasted it in a Dart interpreter which outputted “OK,” meaning that the test passed.

Experience So Far

We at first attempted to use the web app version of Blockly, which utilizes JavaScript, but we started having troubles with that and switched to the Android mobile app version that utilizes Java under a false assumption that it would integrate better with Linux. When attempting to get it working with Android, there were a lot of dependencies and licenses that needed to be downloaded and accepted which were not. We accepted all of the licenses, but when we would run the tests, it would say we hadn’t accepted licenses we had in fact accepted. So after working on that for quite awhile, we decided to switch back to the web app version and give it another try. Soon after making the switch, we realized the error we had made in trying to get the JS unit tests to work. We were unable to run them because we did not download the closure-library in order to run them in uncompressed mode. Once that problem was sorted out, we encountered no further issues with the project.