Facebook Open Academy

MIT App Inventor

Joel Haowen TONG

A0108165J

Table of Contents

Facebook Open Academy Report

Here is a list of posts of my journey in App Inventor 2, under the Facebook Open Academy Program.

If you are at NUS, Facebook Open Academy (listed under CP3101A Global Open Source project) is a really valuable opportunity learn something out of the textbook, contributing to a large-scale open-source project within a community, improving your Git-fu, and being mentored in the whole process, among others. Go for it!

Weekly logs

- Week 1: http://joeltong.org/blog/?p=274
- Week 2: http://joeltong.org/blog/?p=272
- Week 3: http://joeltong.org/blog/?p=270
- Week 4: http://joeltong.org/blog/?p=267
- Week 5: http://joeltong.org/blog/?p=263
- Week 6: http://joeltong.org/blog/?p=260
- Week 7: http://joeltong.org/blog/?p=258
- Week 8: http://joeltong.org/blog/?p=256
- Week 9: http://joeltong.org/blog/?p=253
- Week 10: http://joeltong.org/blog/?p=250
- Week 11: http://joeltong.org/blog/?p=245
- Week 12: http://joeltong.org/blog/?p=298

Summary

- http://joeltong.org/blog/?p=277

Developer Docs

- **API:** https://docs.google.com/document/d/1s-HOWTea0YBLP4UpYteeS94IgB7H-SlXPmQJ9hxhrGs/edit
- **Working Spec:** https://docs.google.com/document/d/1jcZ3FNsjMZPQVVCyaKAmAktT3MKZLapE3XMJdrMfM9U/edit?usp=sharing

Code

- **Java:** https://github.com/myrtleTree33/appinventor-sources/commit/7b4efa76e31a710bb9aa4c12e2947c108fb846f0
- **Javascript** (hosted on AppSpot): https://github.com/myrtleTree33/TestBlockly/commit/104f63f7a2fae5f19f4065f707d03adaea8e6bdd

PDF Generation

Generated with Gitbook 2.0.1. To compile the PDF, install Gitbook and use:

```
gitbook pdf . book.pdf
```

Facebook OA Kickoff / AI2 – Week 1 (January 28 – February 4)

I arrived in the Bay Area a week earlier on Friday January 23, with the intention of completing and offsetting school coursework on the Stanford premises in addition to getting adjusted to the time-zone and a brief exploration of the area. On the weekend and a day at San Francisco, I made a visit with a friend to Dropbox, Evernote, Google and surrounding startups. It was very exciting to see the work culture in Silicon Valley.

At the Facebook Open Academy kickoff, we were introduced to our team. The current team for App Inventor is in its third iteration, and Jose has had successful runs of a course on App Inventor with students from MIT as a formal course. Our team comprised of five CMU students, one from UIUC and me. Some members of the team had worked extensively with Java and apps via the Android Studio, and appeared to be from a combination of mainly CS, and one student from ECE.

Jose was extremely patient with us and I enjoyed his mentorship. He took great pains to familiarize us with the development environment and was extremely approachable, never out of reach for questions although he was having jet-lag.

For the first day, Jose got us familiarized with the team. We went through presenting our findings and corresponding apps and we broke the issues down into 3 columns, which feature on our trello: https://trello.com/b/hplMVWBb/open-academy-15-projects . The workflow is to create a proposal on Google Docs, receive feedback from the community, before coding it extensively and integrating their feedback such that a pull request can be accepted. Our first day was also spent reading the docs, configuring our IDE (in my case, Eclipse) and getting our hands dirty with the code and tutorials from the course conducted at MIT.

Our second day was spent familiarizing ourselves with the code base. It is interesting to note that AI2 compiles down to scheme, which is then used to generate YAML files for the creation of the program in Java. To support the live functionality of the debugger and official app at the same time, two different workflows are used to generate the Scheme needed. Hence developing a new block would require two different workflows. During the latter part of the second day, Jose got me up and running with the proper workflow of using Git, including the meaning of various branches. I am especially thankful for this run-through and am up to speed with using Git, with reference to a book on Git read on the plane.

It was very great meeting the new members of our team and I am now more confident of contributing to a mature code base, contributing to AI2, and finding the resources for AI2 as and when needed.

Facebook OA / AI2 – Week 2 (February 4 – February 11)

I arrived in Singapore Wednesday 1am. The next few days were spent getting up to speed and catching up with lectures, tutorial classes, TAing and homework. Just to familiarize myself with Git, this week was spent testing out the Git workflow learnt at Open Academy, and utilizing it on mock projects.

I spent the week covering the various documents and workshops we went through at Open Academy, just to make sure I fully understood the development process of working on AI2. The week was spent keeping in mind how to best integrate the Phaser development into the environment. I was perplexed how to multiplex Blockly blocks, which were targeted for code generation into scheme, into Javascript when a Phaser block was detected. I was thinking of using labels, that could be picked up by Java App Inventor framework. However, that is something very sketchy at this point in time and needs much more development.

Facebook OA / AI2 – Week 3 (February 11 – February 18)

I spent the first part of the week catching up with leftover labs and school assignments. I have also migrated my development to IntelliJ instead of Eclipse and it is working much better. Instead of configuring stuff, now more is done by the IDE and this eases my development with working on the large code base.

For this week, I proposed on the forum of moving development to a Docker container. This would make development easier via a single command, instead of running multiple commands in multiple terminal windows as we have to do now. Right now it is not as pressing and am leaving it as a suggestion to the community to develop it.

Facebook OA / AI2 – Week 4 (February 18 – February 25)

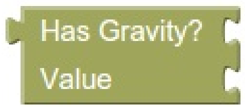
This week was spent thinking of the possible workflow. I have resolved the multiplexing problem by intending to move the Phasor Blockly IDE into a separate pop-up window. Thus the IDE based as a HTML5 webapp will solely output Javascript. As existing components are available in Blockly, such as the creation of lists and arrays, the task is eased somewhat with the provision of core components.

The whole integration is scratch-padded into a webapp document, with package management by Bower. I am pretty satisfied with the organization and this helps in getting up to speed with development. When ready, it may be possible for this IDE to be contained in a separate module (as per standard, modularized practice) and deployed as an iframe component in App Inventor2. This is still sketchy and the aim is to create a simplified game using a simplified Blocks workflow in AI2 first.

To understand the workflow of components, I have followed a tutorial online to create the game shown below. It is a game to collect stars. In addition, as Phasor IO relies heavily on function chains (and thinking in the line of a beginner programmer), I am adamant that these chains would have have to be simplified. For example, the API call to enable physics on an object is

```
rock.physics.enableBody = true;
rock.body.gravity = 300;
```

which probably does not make sense to a beginner programmer! So the Blockly equivalent should be instead should be instead look like



I will be suggesting these to Jose for consideration. We can hopefully move forward with our project next with the creation of a few varying blocks.

Right now, the current mock framework spews out PhaserIO Javascript which can be embedded into a template file. I will look into embedding these live, such that the game can be previewed live as we code.

The next phase of development is to get basic generator going, which can be embedded as a module the App Inventor's repository later.

Facebook OA / AI2 – Week 5 (February 25 – March 4)

For this week, I had a disussion with my mentor Jose about what we should implement for Phaser.

Originally, we discussed about implementing Phaser as-is, using blocks and building a separate add-on. This was submitted as Version 1 of the API. However, Jose cautioned against doing that and instead asking me questions about:

> What do you really like to see being carried forward?
>
> Think from the perspective of the user, a person who might just have had his first experience with programming.

He advised that instead of coding a solution, on roadmap should be instead listening to the user, drafting a proposal to the Open-Source community, create something fast and sketchy, and find out what they think about it. Then, improve and refactor later, add more new features later.

I found this to be really insightful and proceeded to follow his advice.

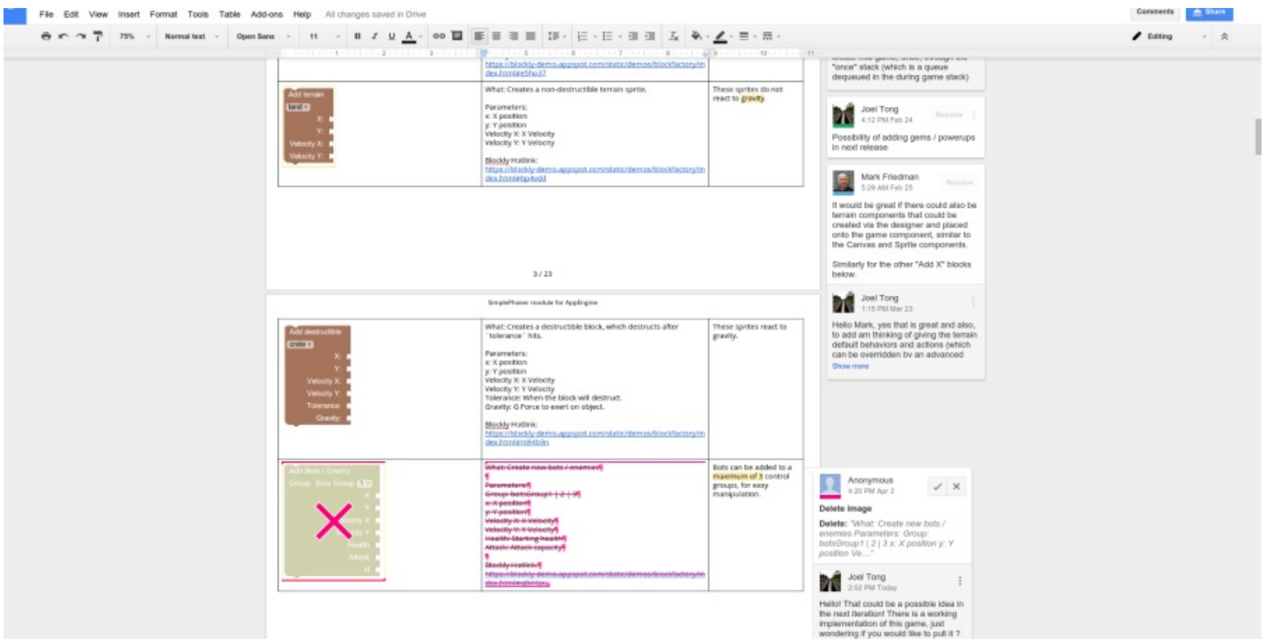
The week was spent playing around with Blockly factory, drafting a proposal and mocking components and thinking *what* should *really* developed, and what I would like to see in the game.

There are many types of games for example:

- Top-level games
- RPG
- RTS
- Arcade

I came to the conclusion that given our need for a certain degree of flexibility, we have to restrain our implementation to a certain type of game. And for this, we chose the **arcade** game type which is really fun to explore, to develop as a phone app, and really interesting for end-users to view.

So here's how the working spec proposal (Version 2 of the API) looks like on Google Docs! And we received some feedback here: https://docs.google.com/document/d/1jcZ3FNsjMZPQVVCyaKAmAktT3MKZLapE3XMJdrMfM9U/edit?usp=sharing



https://blockly-demo.appspot.com/static/demos/blockfactory/index.html#e5ho37

What: Creates a non-destructible terrain sprite.

Parameters:
x: X position
y: Y position
Velocity X: X Velocity
Velocity Y: Y Velocity

Blockly Hotlink:
https://blockly-demo.appspot.com/static/demos/blockfactory/index.html#bp4vdd

These sprites do not react to gravity.

3 / 23

SimplePhaser module for AppEngine

What: Creates a destructible block, which destructs after `tolerance` hits.

Parameters:
x: X position
y: Y position
Velocity X: X Velocity
Velocity Y: Y Velocity
Tolerance: When the block will destruct.
Gravity: G Force to exert on object.

Blockly Hotlink:
https://blockly-demo.appspot.com/static/demos/blockfactory/index.html#m4b9n

These sprites react to gravity.

~~What: Create new bots / enemies¶~~
~~¶~~
~~Parameters¶~~
~~Group: botsGroup1 | 2 | 3¶~~
~~x: X position¶~~
~~y: Y position¶~~
~~Velocity X: X Velocity¶~~
~~Velocity Y: Y Velocity¶~~
~~Health: Starting health¶~~
~~Attack: Attack capacity¶~~
~~¶~~
~~Blockly Hotlink:¶~~
~~https://blockly-demo.appspot.com/static/demos/blockfactory/index.html#ggfmipu~~

Bots can be added to a maximum of 3 control groups, for easy manipulation.

"once" stack (which is a queue dequeued in the during game stack)

Joel Tong
4:12 PM Feb 24
Possibility of adding gems / powerups in next release

Mark Friedman
5:29 AM Feb 25
It would be great if there could also be terrain components that could be created via the designer and placed onto the game component, similar to the Canvas and Sprite components.

Similarly for the other "Add X" blocks below.

Joel Tong
1:15 PM Mar 23
Hello Mark, yes that is great and also, to add am thinking of giving the terrain default behaviors and actions (which can be overridden bv an advanced
Show more

Anonymous
4:20 PM Apr 2
Delete image
Delete: *"What: Create new bots / enemies Parameters: Group: botsGroup1 | 2 | 3 x: X position y: Y position Ve..."*

Joel Tong
2:02 PM Today
Hello! That could be a possible idea in the next iteration! There is a working implementation of this game, just wondering if you would like to pull it ?

Next steps

Am really happy with the feedback and will proceed to implement the basics of the features.

Facebook OA / AI2 – Week 6 (March 4 – March 11)

This week, I am down with Dengue, a mosquito-born ailment in Singapore. Hence went slow on coding.

This week, I managed to churn a Gulp script together with Browserify. Previously, the build was a rough sketchup and hence utilized many `<script>` tags. Talk about having multiple scripts, handling dependencies, ensuring that things are called properly and the like. This is my first time using Gulp, and its going pretty well!

Now is is really simple, following NodeJS's pattern.

Dependencies: Managed with Bower

Build: Gulp

Minification: Browserify

So now we can simply do the usual node `require()` and dependencies will be automatically handled across modules.

```
require('../bla.js');
...
```

For sprites, I have settled on an inheritance pattern enforced by mixins. There will be a basic sprite which handles garbage collection methods in the background. Hence, the idea is that these sprites will feature very high-level methods to the user and will not bother about how to, for example, load a texture for the tree sprite or perhaps even configuring collision and physics behavior for a rock.

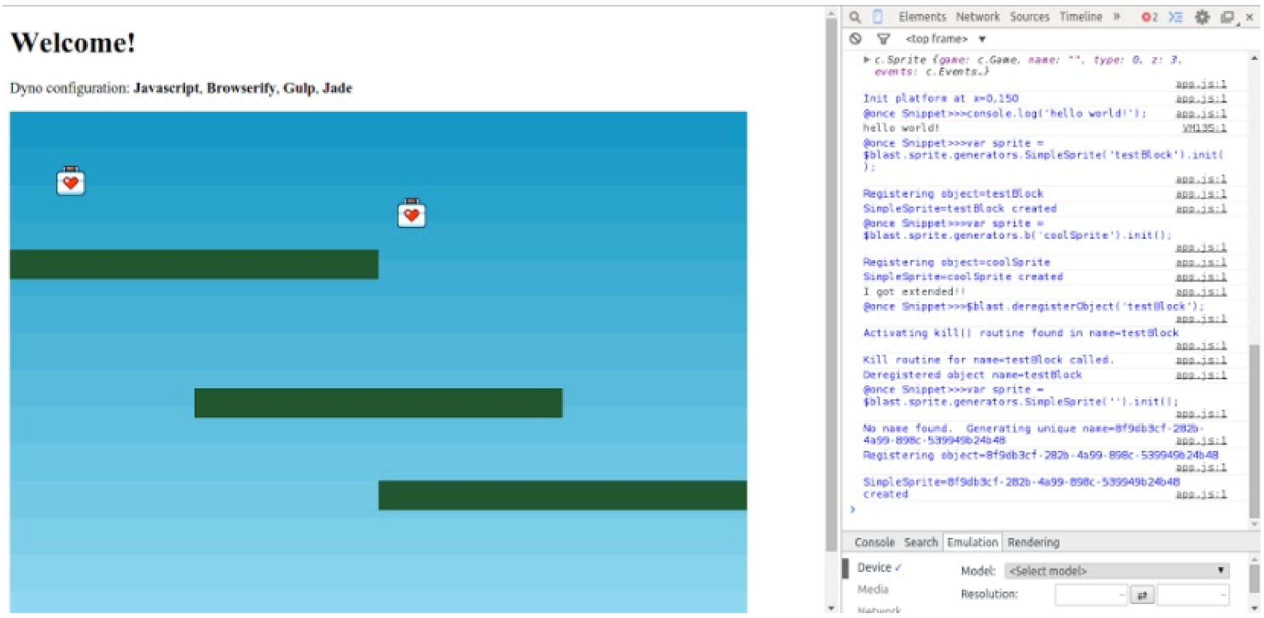
So after much experimentation, here is the final pattern for a basic sprite!

```
__generators.SimpleSprite = function(name) {
    var _scope = undefined;

    var _kill = function() {
    delete this;
    }

    var _init = function(scope) {
    _scope = scope;
    ...
    }

    /**
     * Override this implementation.
     */
    var init = function() {
    /** Insert code here **/
    /** End insert code here **/
    _init(this);
    }

    /**
     * Override this implementation.
     */
    var kill = function() {
    /** Insert code here **/
    /** End insert code here **/
    _kill();
    }

    return {
    _init: _init,
    _kill: _kill,
    init: init,
    kill: kill
    }
}
```

All future sprites implement the kill and init functions, instead of letting the user do so. No need for the user to program an explosion animation, no need to care for mass and etc. Take for example, the rock sprite:

```
__generators.rock = function (group, name, x, y, gravity) {
    //var nativeObject = __generators.SimpleSprite('rock1');
    var nativeObject = __generators.SimpleSprite(name);

    var init = function () {
    var rock = $blast._groups[group].create(x, y, 'rock1');
    rock.group = group;
    rock.body.gravity.y = gravity;
    rock.body.bounce.y = 0.7 + Math.random() * 0.2;
    rock.outOfBoundsKill = true;
    rock.body.collideWorldBounds = true;
    this.obj = rock; // add to object
    nativeObject._init(this);
    console.debug("Init rock at x=" + x + ',' + y);
    };

    var kill = function () {
    console.log("=KILL= I got called");
    __generators.explosion(this.obj.x, this.obj.y);  // plays an explosion
    this.obj.destroy();
    nativeObject._kill();
    };

    return _.extend({}, nativeObject, {
    init: init,
    kill: kill
    });
};
```

Here's a few sprites and the garbage collector dump, just to test things out on the backend:


Welcome!
Dyno configuration: **Javascript**, **Browserify**, **Gulp**, **Jade**
Elements Network Sources Timeline
<top frame>
Console Search Emulation Rendering
Device ✓
Media
Network
Model: <Select model>
Resolution:

```
▶ c.Sprite {game: c.Game, name: "", type: 0, z: 3,
  events: c.Events…}
                                                        app.js:1
Init platform at x=0,150                                app.js:1
@once Snippet>>>console.log('hello world!');            app.js:1
hello world!                                             VM135:1
@once Snippet>>>var sprite =
$blast.sprite.generators.SimpleSprite('testBlock').init(
);
                                                        app.js:1
Registering object=testBlock                            app.js:1
SimpleSprite=testBlock created                          app.js:1
@once Snippet>>>var sprite =
$blast.sprite.generators.b('coolSprite').init();
                                                        app.js:1
Registering object=coolSprite                           app.js:1
SimpleSprite=coolSprite created                         app.js:1
I got extended!!                                        app.js:1
@once Snippet>>>$blast.deregisterObject('testBlock');
                                                        app.js:1
Activating kill() routine found in name=testBlock
                                                        app.js:1
Kill routine for name=testBlock called.                 app.js:1
Deregistered object name=testBlock                      app.js:1
@once Snippet>>>var sprite =
$blast.sprite.generators.SimpleSprite('').init();
                                                        app.js:1
No name found.  Generating unique name=8f9db3cf-282b-
4a99-898c-539949b24b48                                  app.js:1
Registering object=8f9db3cf-282b-4a99-898c-539949b24b48
                                                        app.js:1
SimpleSprite=8f9db3cf-282b-4a99-898c-539949b24b48
created                                                 app.js:1
>
```

Looking good! For next steps, will attempt to link the Java and Javascript sides together.

Facebook OA / AI2 – Week 7 (March 11 – March 18)

This week, I was just up recovering from Dengue. Missed a midterm, but making up for it after the period. Lots of schoolwork catching up this week.

I have managed to connect the interface between Java and SimplePhaser, and it looks really neat now. It is a relief as the theory behind our program implementation works!

Wrote an app to test things out. And so this is how bi-directional callbacks actually work!

Java to JS

```
WebView.loadUrl("javascript:myJavascriptFunction()");
```

JS to Java

This is what you do on the Java side, exposing the interface:

```
/** Java **/
@JavascriptInterface
public void showToast(String toast) {
    Toast.makeText(mContext, toast, Toast.LENGTH_SHORT).show();
}
...
// from http://developer.android.com/guide/webapps/webview.html
WebView webView = (WebView) findViewById(R.id.webview);
webView.addJavascriptInterface(new WebAppInterface(this), "Android");
```

And on the JS side, call it like this:

```
function showAndroidToast(toast) {
    Android.showToast(toast);
}
```

For next steps, I will work on the collision handling.

Facebook OA / AI2 – Week 8 (March 18 – March 25) - Added a collision manager!

For this week, I have successfully implemented the collision manager. It builds on top of PhaserIO's collision manager, which already supports some basic collision handling across groups.

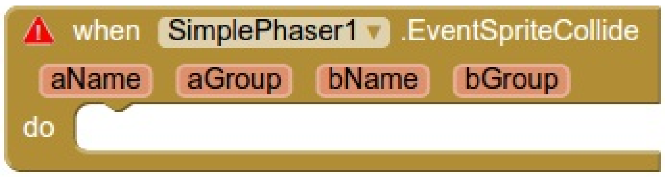
Triggering an event in PhaserIO can be done through the following:

```
_game.physics.arcade.collide(__._groups.destructibles, __._groups.bullet, function () {
    console.log("triggered!")
}, null, this);
```

Helpful, but too low-level in my opinion for our needs of code blocks.

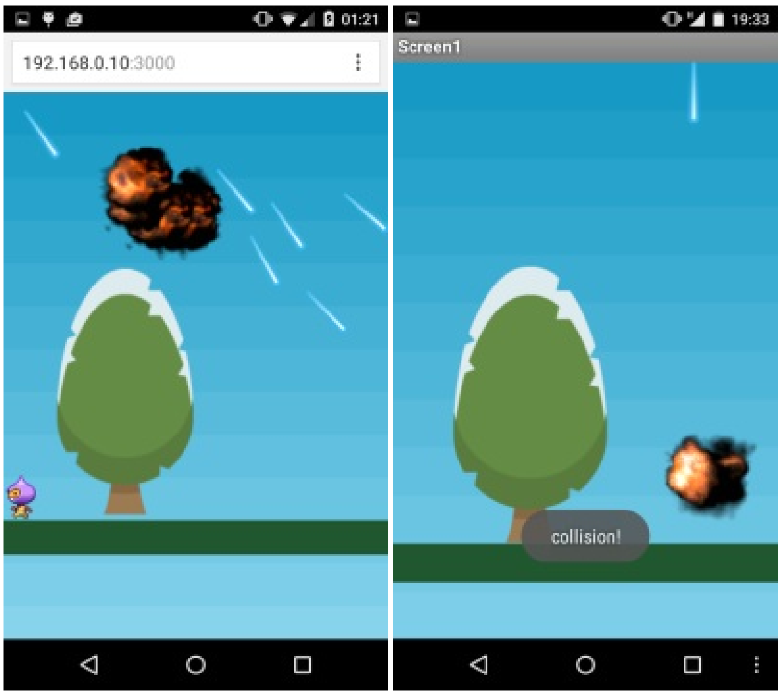
Our idea was to make it really easy for the user to write code, certain objects (bullets) have automatic collision behavior embedded in them. For example, bullets should blow up when coming into contact with a sprite group.

So here is the new block!



So, when Sprite A collides with Sprite B, we worked this event to fire, on the Java end, behind the scenes. It it works fine!!

Below shows 2 different scenes in action (one in the browser, the other a native app created through the code blocks.)



It comes with a nice destruction animation, upon being destroyed.

Implementation

Behind the scenes, a centralized collision manager handles the collisions between sprites. If you recall, we decided upon setting SimplePhaser to utilize fixed groups, instead of letting the user control the collision groups (doing otherwise would be really messy and too low-level setting up collisions, as users would have to tweak a little of the low-level code, at least 3 to 4 steps before doing anything. By attaching pre-defined collision hooks to these sprites in a given fashion, we avoid having to attach collision callbacks for every single sprite to every other sprite in the world.

As it is, this is pretty heavy in SimplePhaser and here is how it looks like:

```
        // collision detection
        /** terrain 1 **/
        __._game.physics.arcade.collide(__._groups.terrain1, __._groups.terrain1);
        __._game.physics.arcade.collide(__._groups.terrain1, __._groups.terrain2);
        __._game.physics.arcade.collide(__._groups.terrain1, __._groups.powerups);
        __._game.physics.arcade.collide(__._groups.terrain1, __._groups.destructibles);
        __._game.physics.arcade.collide(__._groups.terrain1, __._groups.player1);
        __._game.physics.arcade.collide(__._groups.terrain1, __._groups.player2);
        __._game.physics.arcade.collide(__._groups.terrain1, __._groups.enemy1);
        __._game.physics.arcade.collide(__._groups.terrain1, __._groups.enemy2);
        __._game.physics.arcade.collide(__._groups.terrain1, __._groups.enemy3);

        /** Terrain 2 **/
        __._game.physics.arcade.collide(__._groups.terrain2, __._groups.terrain2);
        __._game.physics.arcade.collide(__._groups.terrain2, __._groups.terrain1);
        __._game.physics.arcade.collide(__._groups.terrain2, __._groups.powerups);
        __._game.physics.arcade.collide(__._groups.terrain2, __._groups.destructibles);
        __._game.physics.arcade.collide(__._groups.terrain2, __._groups.player1);
        __._game.physics.arcade.collide(__._groups.terrain2, __._groups.player2);
        __._game.physics.arcade.collide(__._groups.terrain2, __._groups.enemy1);
        __._game.physics.arcade.collide(__._groups.terrain2, __._groups.enemy2);
        __._game.physics.arcade.collide(__._groups.terrain2, __._groups.enemy3);

        /** Bullet **/
            //__._game.physics.arcade.collide(__._groups.bullet, __._groups.destructibles);
            //__._game.physics.arcade.collide(__._groups.bullet, __._groups.terrain1);
            //__._game.physics.arcade.collide(__._groups.bullet, __._groups.terrain2);
        __._game.physics.arcade.collide(__._groups.destructibles, __._groups.bullet, _collisionManager, null, this);
        __._game.physics.arcade.collide(__._groups.terrain1, __._groups.bullet, _collisionManager, null, this);
        __._game.physics.arcade.collide(__._groups.terrain2, __._groups.bullet, _collisionManager, null, this);
```

All the unified collision manager has to do now is to send the signal over to Java, and raise an event:

```
    if (hasAndroid) {
        Android.onCollision(spriteA.name, spriteA.obj.group, spriteB.name, spriteB.obj.group);
    }
```

Other blocks

In addition, I have added deletion, setting the XY coordinates of a sprite, this week.

Next steps

Next steps will be adding getters and setters to sprite blocks.

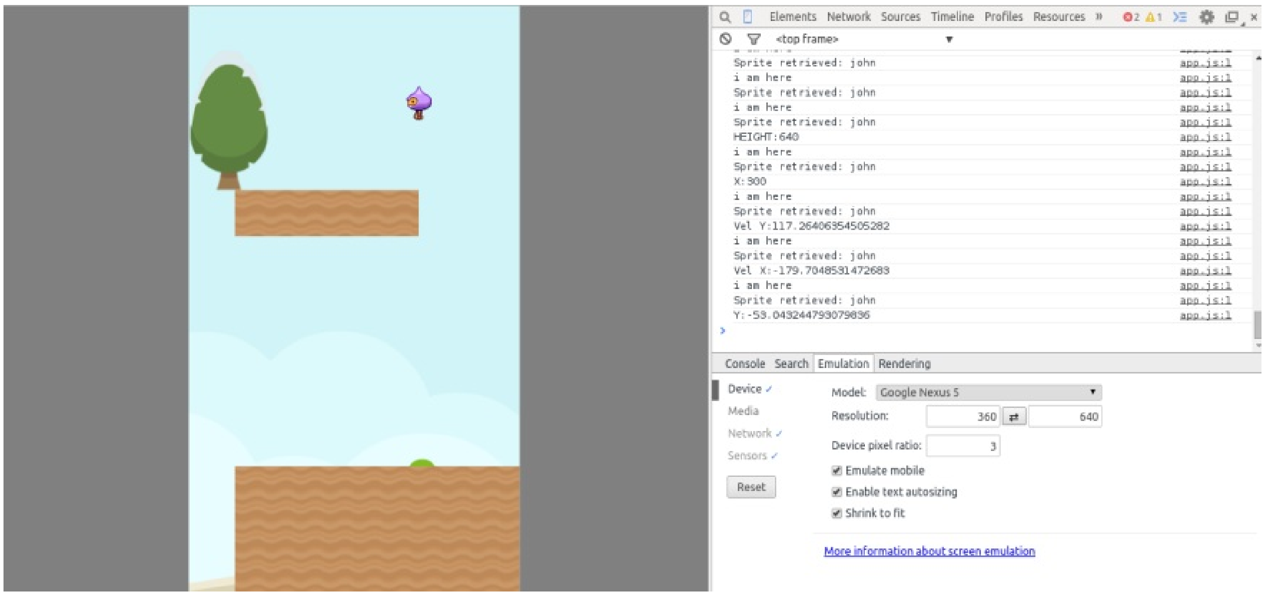
Facebook OA / AI2 – Week 9 (March 25 – April 1)

For this week, I added new components to the game. One of these components is a flexible platform.

Recalling our previous implementation of platform, there was no support for custom-sized platforms. In the latest iteration however, this is possible.

I also added a couple of animated sprites to control the user animations. However, instead of dealing with the low-end Blockly code, it is possible to change the state of the player sprite (running / moving / left / right) via a simple function call. Phaser already does some of the heavy-lifting (For example, cutting up the spritesheet into the various frames), however this is not intuitive and may be too slow for the novice user if it were implemented through blocks.

So here is the game in action, with a couple of new sprites:



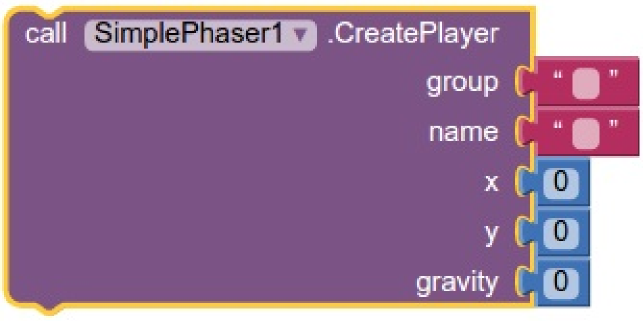
```
Sprite retrieved: john                    app.js:1
i am here                                 app.js:1
Sprite retrieved: john                    app.js:1
i am here                                 app.js:1
Sprite retrieved: john                    app.js:1
HEIGHT:640                                app.js:1
i am here                                 app.js:1
Sprite retrieved: john                    app.js:1
X:300                                     app.js:1
i am here                                 app.js:1
Sprite retrieved: john                    app.js:1
Vel Y:117.26406354505282                  app.js:1
i am here                                 app.js:1
Sprite retrieved: john                    app.js:1
Vel X:-179.7048531472683                  app.js:1
i am here                                 app.js:1
Sprite retrieved: john                    app.js:1
Y:-53.043244793079836                     app.js:1
```

Model: Google Nexus 5
Resolution: 360 ⇄ 640
Device pixel ratio: 3
☑ Emulate mobile
☑ Enable text autosizing
☑ Shrink to fit

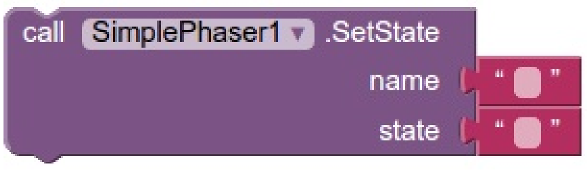
Oh and, you could change the texture of the custom platform!

And the blocks:

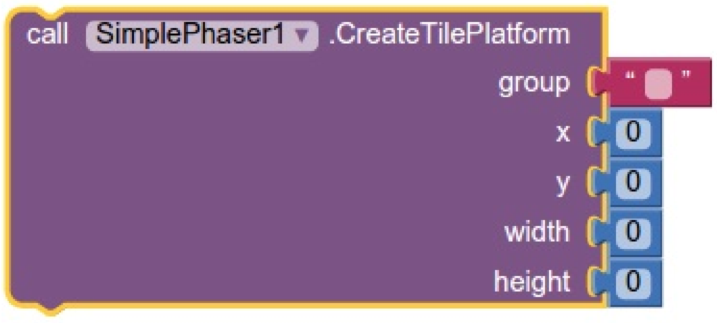
Create a user:



Ever wanted the user to turn left? No problem!



And the custom platform block:



For next steps, I am debugging the getter function. Right now the issue with the getter is, synchronizing a bi-directional callback between the Java and Javascript endpoints. Its using an event-driven callback, and while it seems alright, the game somehow hangs every time. Debugging it for next week! (Pressing issue).

Facebook OA / AI2 – Week 10 (April 1 – April 8)

This week, I am into my third week resolving the getter / setter bug, and finally managed to solve it!

The problem

To recap, the setter was working all fine, as the game state was maintained on the Javascript end and altering was simply performed through a one-time function call. However, the getter was slightly different. The event had to be triggered from the Java end, calling a Javascript function and somehow returning a value back to Java, in-order and synchronous. There was no method existing to directly retrieve a Javascript function call's routine directly, so a method had to be devised.

Previously, this was done through an event-driven method. Java would call the Javascript function with a unique UUID as a reference. Upon completing the routine, the Javascript function call another universal Java function to put it into a key-value data structure, using the UUID given. The original Java function would be polling this data structure in the meantime, and upon finding a UUID which matches, would deque the entry, and return the value to the user.

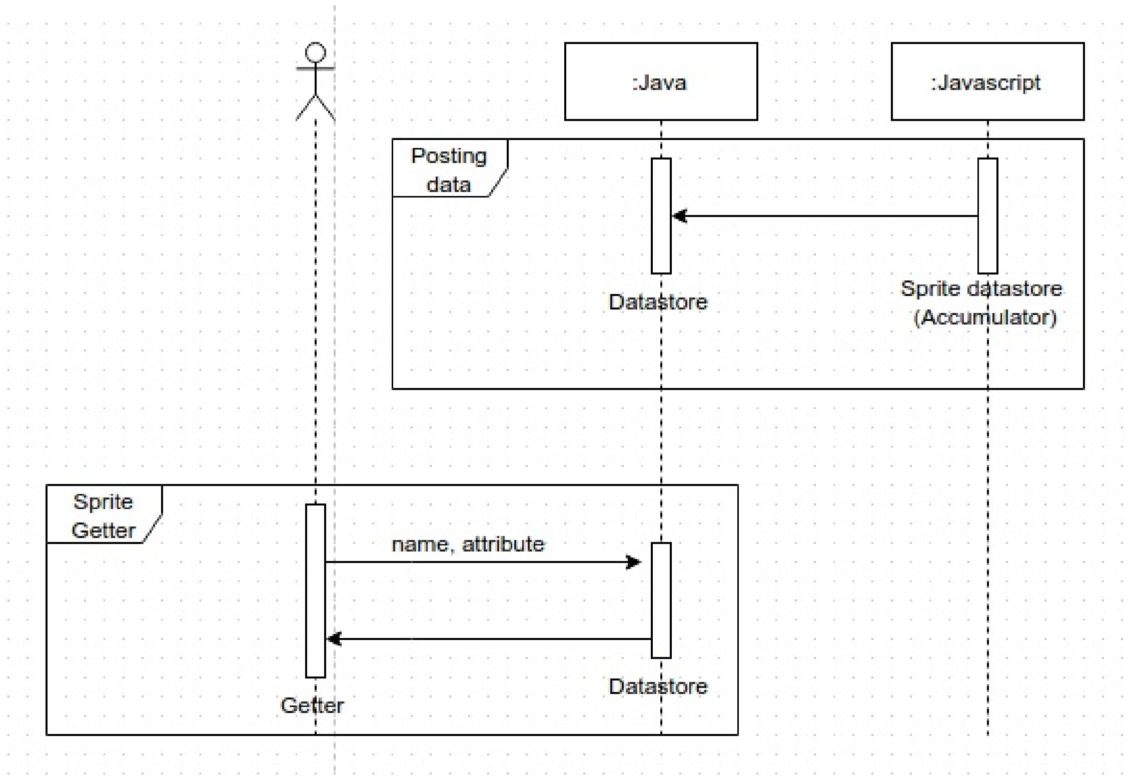
This method did not work! After numerous checks, there was no reason why it should not work and the problem was isolated to a possible way Java webview might have handled the thread calls. Apparently, there was a blocking thread somewhere after the original Java function was called, leading to the game blocking on this thread and hence hanging. Given not much form of debugging and a relatively long build time, this was frustrating and took a long time to trace. But nevertheless fun indeed!

The solution

In the end, I adopted another alternative (thank God!) that solved the issue. Instead of storing state on the Javascript function, sprite states were also posted frequently (~0.2 secs, any slower would report the wrong value) to the Java end. These reports were stored in Java, and upon requesting for a getter function call, the Java function would return its last reported value back to the program executed. Dirty, quick, and somewhat sounding a little inefficient, but it works!

TL;DR

This might be getting a little hard to see without visuals, so here is a callback diagram:



Just to test the piece works, I wrote a program to set the value of a sprite using its getter (which should effectively not hang the game and run as per normal).

Phew. It worked.

For next steps, I will be moving on to adding user interfacing and packaging the game for release to the public.

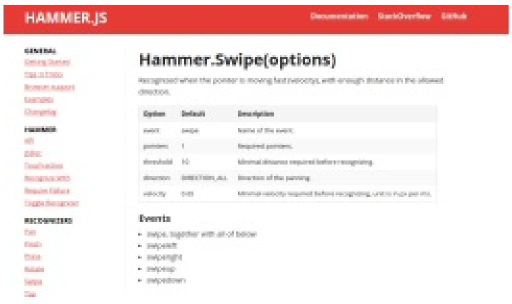
Facebook OA / AI2 - Week 11 (April 8 - April 15)

This week, I spent some time wrapping up the functionality of the SimplePhaser component. To recap, I am developing a component that allows one to easily create games via blocks through the App Inventor framework.

The previous few weeks were intense as the getter code functions refused to work, and was finally narrowed down to a thread issue.

For this week, I worked on adding some touch interactivity to SimplePhaser. Now, there is an integrated event manager(s) for the tracking of swipe (up/down/left/right), in addition to single taps and drags. This was accomplished via HammerJS, and for the record, this added some nice swipe on the Javascript end:

```
// for swiping features
var _bindHammer = function () {
    var ele = document.getElementsByTagName('body')[0];
    var hammertime = Hammer(ele);

    hammertime.on('swipeup', function (event) {
        _inputManager.onSwipe('up');
    });

    hammertime.on('swipedown', function (event) {
        _inputManager.onSwipe('down');
    });

    hammertime.on('swipeleft', function (event) {
        _inputManager.onSwipe('left');
    });

    hammertime.on('swiperight', function (event) {
        _inputManager.onSwipe('right');
    });

    hammertime.on('swipe', function (event) {
        _inputManager.onSwipe('swipe');
    });
    console.log('--- Attached Hammer.JS ---');
};
```



Hammer.Swipe(options)

Recognized when the pointer is moving fast (velocity), with enough distance in the allowed direction.

Option	Default	Description
event	swipe	Name of the event.
pointers	1	Required pointers.
threshold	10	Minimal distance required before recognizing.
direction	DIRECTION_ALL	Direction of the panning.
velocity	0.65	Minimal velocity required before recognizing, unit is in px per ms.

Events

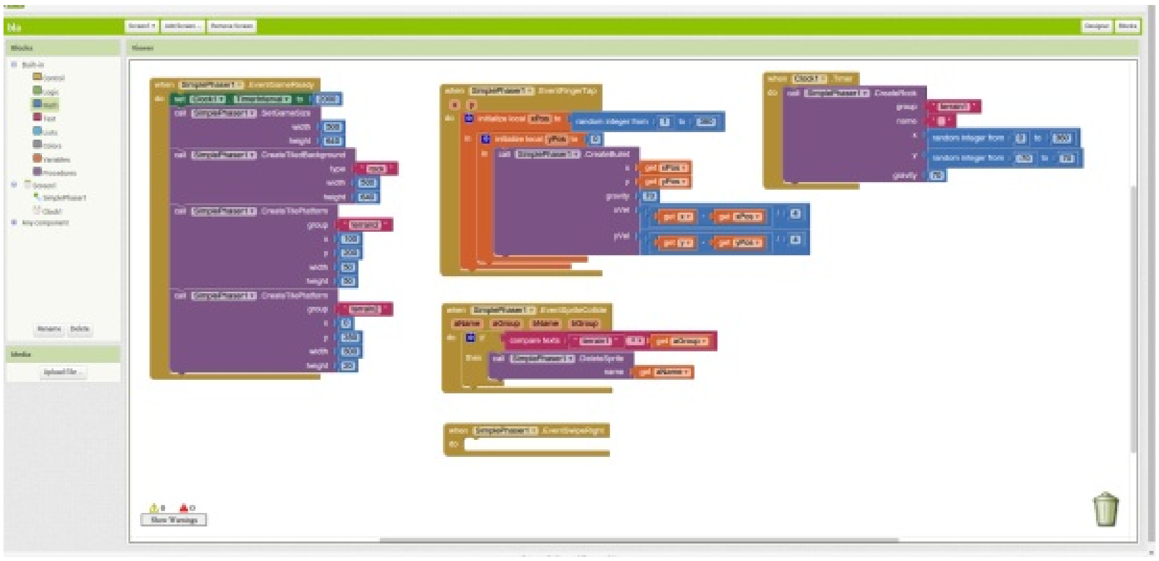
- swipe, together with all of below
- swipeleft
- swiperight
- swipeup
- swipedown

Also coded this week is the ability to increase world size, and also the ability to move the camera around the world, in addition to being able to track a sprite object with the camera.

Testing

A small game using the code blocks was written to test the functionality of the game. In this game, rocks fall down from the sky. It is the user's duty to clear the rocks by tapping on the screen, causing them to explode. Bullets in the direction of the user's tap fall from the sky, and are influenced by gravity. The game in action:

Here is the program. It also uses another external component, the Timer compoent. The inspiration is that users will mix and match components / sensors in future, to create something interesting! (Proximity?)



Code base for v0.1-alpha

Code bases have been tagged as v0.1-alpha and released as the following. Still planning to work on a couple of additions / bugs so there might be hotfixes along the way. In either case, this is stable.

- The Javascript at https://github.com/myrtleTree33/TestBlockly/releases/tag/v0.1-alpha and is hosted on my remote server.
- The Java at https://github.com/myrtleTree33/appinventor-sources/releases/tag/SimplePhaser-v0.1-alpha .

Next steps

For the next week, I will concentrate on improving the official documentation. Right now there's still some online proposal going on at https://docs.google.com/document/d/1jcZ3FNsjMZPQVVCyaKAmAktT3MKZLapE3XMJdrMfM9U/edit?usp=sharing , which is targeted as a proposal for features and discussion. The official spec will have information how to properly enforce tasks, for e.g. specifying the correct groups, as well as getting up and running with the game. Hopefully, that can be a testbed to see how quickly users adapt to the SimplePhaser framework, and future adjustments made accordingly.

Also to be added are bux fixtures and introducing this to the community.

What is implemented

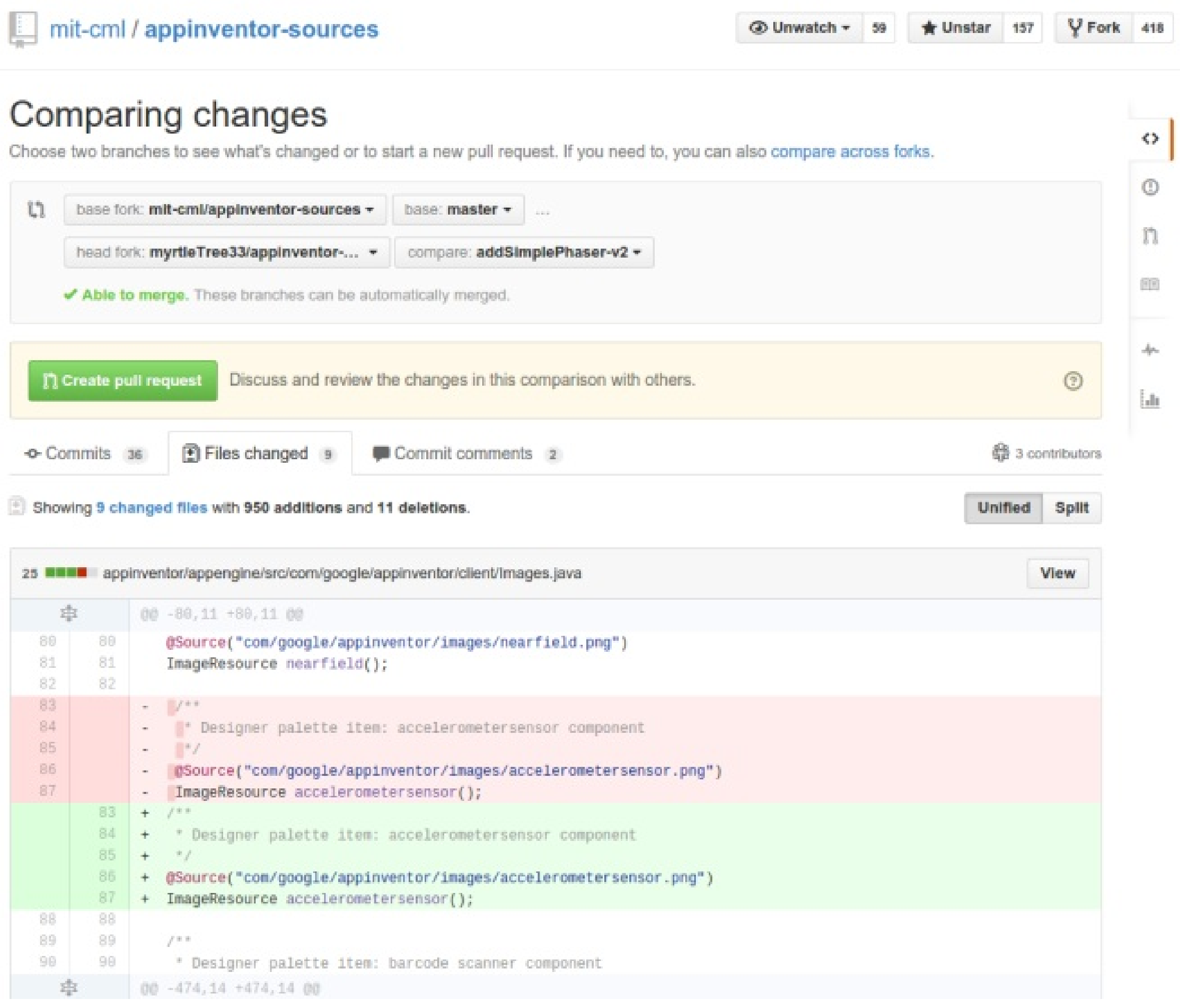
For the record, taken from the release logs, the following are implemented:

```
## Sprites supported:

- Tree
- Player
- Rock
- Sky
- TiledBackground
- TilePlatform
- Platform
- Bullet

## Groups supported:

- bullet
- terrain1
- terrain2
- destructibles


## Collision detection handling

### Bullets

- terrain1
- terrain2
- destructibles


## Camera

- Camera XY setters
- Camera tracking an object


## User Interaction

- Single Tap / drag
- Swipe left / right / up / down / general


## Player

- Ability to change player sprite state (left animation / right animation)


## Features

- Collision detection
- Automated destruction animation and deletion upon sprite destory
- Automated destruction of bullets upon hitting object
- Gravity
- Getters / Setters
- Making world larger than screen size
- Moving camera around world
- Tracking camera on object
- Swipe and touch events


## Production-ready URL

- Loaded from a remote server.  To be implemented on device during production.
```

Facebook OA / AI2 – Week 12 (April 15 – April 22)

The week was spent tidying up SimplePhaser with the master branch for submission. Some comments had to be renamed, accidental indentations had to be removed. The code was not synced with master for a while, and hence needed to be updated with `upstream`.

SimplePhaser is now ready for merging with upstream, as far as the code base is concerned:



Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks.

base fork: mit-cml/appinventor-sources base: master

head fork: myrtleTree33/appinventor-... compare: addSimplePhaser-v2

✓ Able to merge. These branches can be automatically merged.

Discuss and review the changes in this comparison with others.

Showing 9 changed files with 950 additions and 11 deletions.

25 appinventor/appengine/src/com/google/appinventor/client/Images.java

```
@@ -80,11 +80,11 @@
   @Source("com/google/appinventor/images/nearfield.png")
   ImageResource nearfield();

-   /**
-    * Designer palette item: accelerometersensor component
-    */
-   @Source("com/google/appinventor/images/accelerometersensor.png")
-   ImageResource accelerometersensor();
+  /**
+   * Designer palette item: accelerometersensor component
+   */
+  @Source("com/google/appinventor/images/accelerometersensor.png")
+  ImageResource accelerometersensor();

   /**
    * Designer palette item: barcode scanner component
@@ -474,14 +474,14 @@
```

Right now, its mainly tidying up the user API, and convincing users to get aboard the system - simplifying workflows which may have been overlooked, and making it not overtly complex such that a beginner has to read a manual. So its refining the workflow, thinking about what the user would have after viewing the prototype.

> What is a callback?
>
> What is SimplePhaser?
>
> what are the sprite groups available and how do I use them?
>
> How do I create many sprites on the canvas?

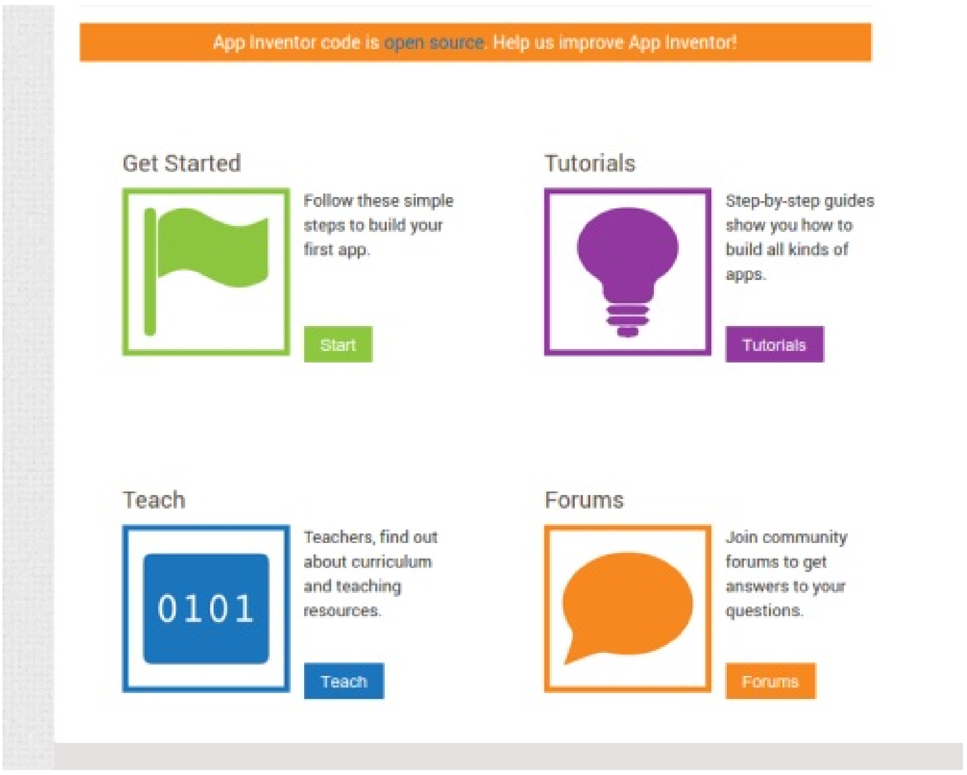
This is crucial, as the whole SimplePhaser package features many blocks (due to its nature of being a game creation engine component), and would need a slightly different approach towards user adoption (Users might be overwhelmed by the blocks).

> Follow the principle of least astonishment.

To facilitate user adoption, next steps would fall under two categories:

1. Empowering the user by providing online tutorials on using SimplePhaser to build games. This could be similar to the videos we have started on App Inventor 2 on the main page:
2. Providing a community of support. As it is, to ensure that questions on the component by users get answered responsively and without delay. This would require support from the community, which should be easier if this gets through a couple of vetting / refinement prior to release (Releasing suddenly might surprise the community)
3. Simplifying the workflow of events. This has been considered during the drafting of SimplePhaser in the developer's forum, and should continue to adapt to input based on what the community suggests. More feedback might imply more support from the community.
4. Renaming SimplePhaser to something which says much about its name. Few outside the Javascript / Game development community might have heard about PhaserIO. Hence, renaming the SimplePhaser component to something like ArcadeGame would help user to be less astonished by the introduction of the component.

Some thoughts to consider for next steps after initial prototyping and development. The next phase would be an interesting one for me, and I am keen to see SimplePhaser being adopted by the community after Facebook OA.


App Inventor code is open source. Help us improve App Inventor!

Get Started

Follow these simple steps to build your first app.

Start

Tutorials

Step-by-step guides show you how to build all kinds of apps.

Tutorials

Teach

Teachers, find out about curriculum and teaching resources.

Teach

Forums

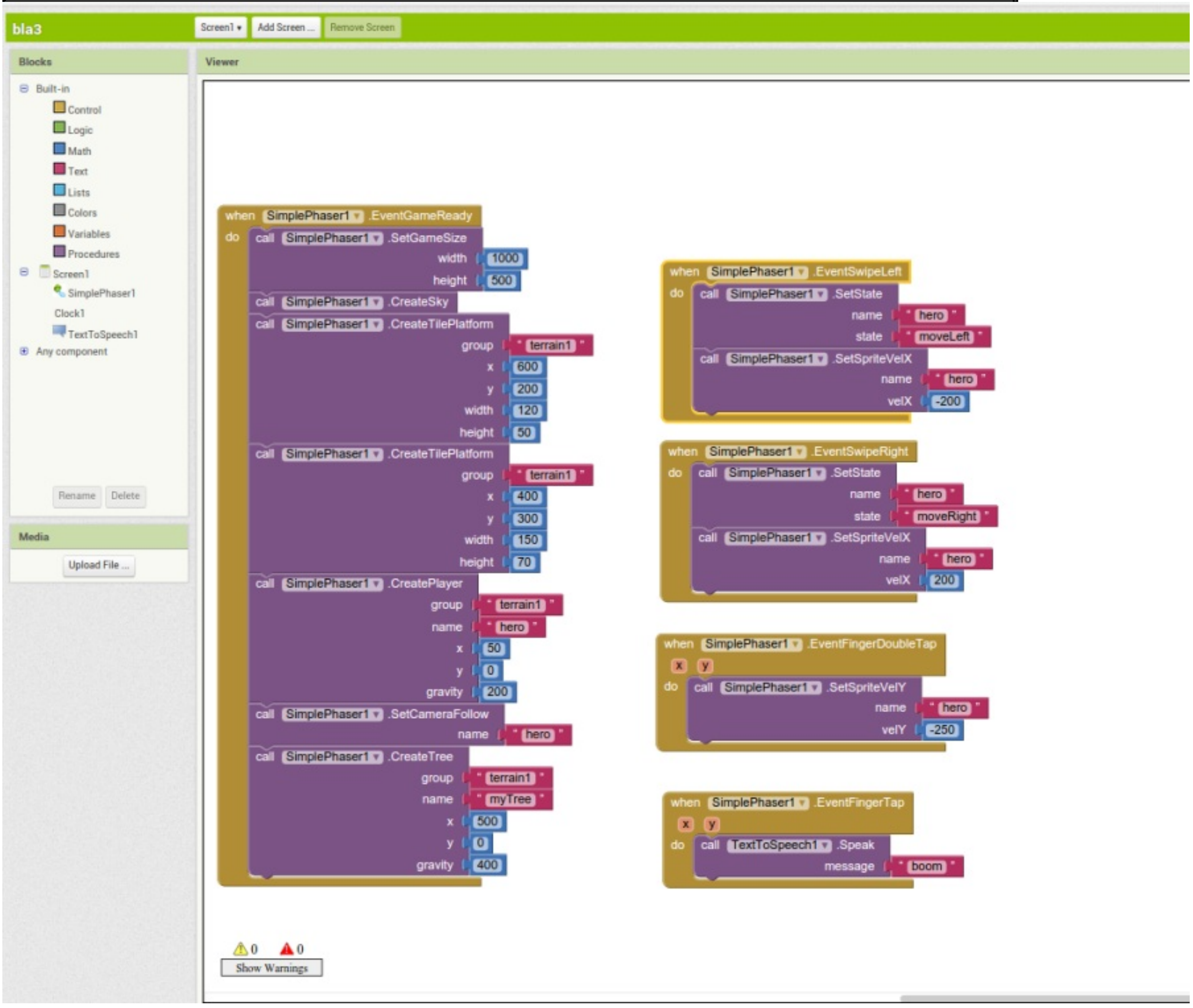
Join community forums to get answers to your questions.

Forums

Adding double-tap functionality

This week was also spent fixing a couple of undetected bugs, and refining the user interaction manager such that it is able to recognize double taps. This was not as trivial as the double-tap handler would have to wait for a while before it can be sent out, and in the event of not detecting a second tap, would fire a single tap event handler.

As the current user interaction system features inputs that are much richer than that supported by pure PhaserIO (for example, taps, pinches, rotations and swipes / drags), the implementation was a coordination between PhaserIO and Hammer.JS, a popular mobile user input framework. Mutex locks were first considered, however they frequently locked and are not recommended in Javascript. Instead, a timed delay of 5ms was given to avoid race condition between the PhaserIO native input and Hammer.JS's event manager, which worked pretty well.



```
when SimplePhaser1 .EventGameReady
do  call SimplePhaser1 .SetGameSize
         width  1000
         height 500
    call SimplePhaser1 .CreateSky
    call SimplePhaser1 .CreateTilePlatform
         group  "terrain1"
         x      600
         y      200
         width  120
         height 50
    call SimplePhaser1 .CreateTilePlatform
         group  "terrain1"
         x      400
         y      300
         width  150
         height 70
    call SimplePhaser1 .CreatePlayer
         group   "terrain1"
         name    "hero"
         x       50
         y       0
         gravity 200
    call SimplePhaser1 .SetCameraFollow
         name "hero"
    call SimplePhaser1 .CreateTree
         group   "terrain1"
         name    "myTree"
         x       500
         y       0
         gravity 400

when SimplePhaser1 .EventSwipeLeft
do  call SimplePhaser1 .SetState
         name  "hero"
         state "moveLeft"
    call SimplePhaser1 .SetSpriteVelX
         name "hero"
         velX -200

when SimplePhaser1 .EventSwipeRight
do  call SimplePhaser1 .SetState
         name  "hero"
         state "moveRight"
    call SimplePhaser1 .SetSpriteVelX
         name "hero"
         velX 200

when SimplePhaser1 .EventFingerDoubleTap
 x  y
do  call SimplePhaser1 .SetSpriteVelY
         name "hero"
         velY -250

when SimplePhaser1 .EventFingerTap
 x  y
do  call TextToSpeech1 .Speak
         message "boom"
```

So yes, Phaser can now swipe left and right! To test the functionality, here is a simple game, which moves a player sprite according to the following moves:

- Swipe left: Move left
- Swipe right: Move right
- Double tap: Jump
- Tap: trigger App Inventor's text-to-speech component to say "boom!"

Note how the camera locks onto the player during moves.

TL;DR and next steps

Having seen a component from conceptualization, to grappling with the AI2 framework, to coordinating data synchronization, to the end in a relatively unconventional way (using Java to control Javascript and controlling state!) has been a wonderful experience for me, to challenging things out of the box as well as receiving input from wonderful mentors and the community. The product is done, deliverables submitted, however next steps to its integration would be a different set of learning experiences (less of coding and more of listening and user adoption), which I am keen to continue learning after the completion of the App Inventor program.

Facebook Open Academy: A Summary

12 weeks have passed since my involvement in App Inventor 2, and while it has been a while of a time, it was a really enjoyable one and fulfilling experience - something that cannot just be learnt through books alone.

Being attached to a large code base, and having credits to do that is a rare opportunity, and I am thankful for it. If you're thinking of going for Facebook Open Academy, well, go for it! Here are pointers I have learnt from a somewhat Git-ignorant, Java newbie to someone more comfortable with handling these tools after this semester.

Listen, listen, listen.

Listening is an important skill in any field - and this includes coding. I found this topic resonant throughout the duration of Facebook Open Academy. Code is cheap, but it may not be necessarily so that the code written is useful to others. Learn to listen to them, learn to engage with the community, and implement code that people will use. This is something I am trying to learn still.

In all things, ask the community for their ideas. Find out what is needed from the developers, from the people, and the things they would like to see.

This really struck me during my discussions with my mentor Jose, a key contributor / maintainer of the project. Originally, I had in mind a rough sketch of what the component should look like and proceeded to implement parts of it. However, throughout this process, Jose encouraged me to relook at the process by engaging the community with a proposal. He encouraged me to put a proposal out there with SimplePhaser. While the idea was to implement most parts of Phaser.IO, we eventually restrained it to what the user really needed - building an Arcade game, using high-level blocks that a new beginner programmer would like to use. And we did that.

A proposal was put in place, and there was some feedback gathered. Having to discuss ideas and discuss why some features were needed was a constructive part of the discussion - something that was revisited time and again during the duration of SimplePhaser. Additionally for blocks, we moved away from the notions of low-level code and attempted to offer something new to the table - being motivated to create a Mario-style game with high-level behavior and blocks that a novice user would understand. And it worked - both for the implementation (it appears), as well as for the community and us.

We will still continue in that direction for the next phase of user adoption.

Implement first, think later. Think out of the box.

During the duration of the project, I was stuck for three weeks on a bug which appeared trivial - retrieving a sprite's property through getters. Recalling the blog post, it was not trivial as it involved calling a Javascript function, and awaiting the value of its callback on the Java end - all without direct support of retrieving the return value of the named Javascript function.

Three weeks were spent debugging this issue. The initial implementation was to implement an event-driven framework, whereby the said values would be identified by UUIDs in a key-value Java store. However, in every single instance where the getter was called, the program would hang! There was no idea where it would hang - it just did.

As there was not much form of debugging - there was not much of a debugger in place, and compiling the Java end would take around 7 minutes - leading to 4 changes an hour, debugging was tedious, sometimes tiring and at times painful. Throughout the whole process, I kept with the event-based protocol, having ruled out that constantly parameter polling would be too intensive and inefficient.

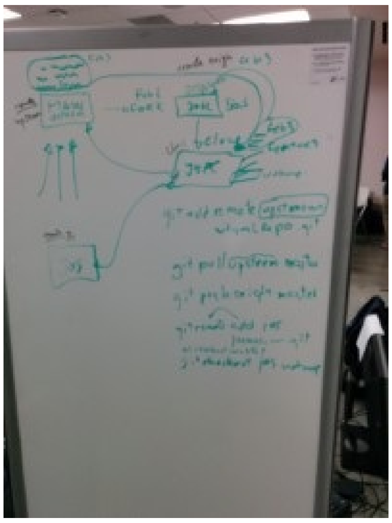
Finally, there was some progress after 2.5 weeks and it appeared that while there did not seem to be a bug with the code, it might be possible that there was an issue with how Javascript's webview functions were called and it appeared while they could be called simultaneously out-of-order, somewhere there was a thread pending internally with the first Javascript call (even thought it appeared calls were out of order).

Whichever the case, things did not seem pretty, and I eventually became open to the method of passing the keystore as a JSON continuously between the Java and Javascript ends, at an interval of 0.2 seconds.

It worked! (With not as much lag, but this remains to be seen with more sprites of course).

Be flexible.

I was hesitant at first contributing to a large code-base, in Java, having come from a C / NodeJS background with little experience in the Java world. IntelliJ was something new to me - what is IntelliJ? Oh and Java - is it inefficient? Why is it using Ant?! The usual stereotypes.



But over time and with prayer, it got better and the more we put our practice into it, the better it became. I was thankful for a good mentor who shared advice on working with App Inventor, getting into Java development in IntelliJ (made the switch from Eclipse and loving it), and even getting up and running with Git through a graphical explanation by my mentor (which I realized, I was using wrongly). Read more books, learn the shortcuts, practice with empty Git repositories, read Stack Overflow, and pray.

Be not afraid to ask, to practice, and learn. It will get better.

Maintaining a large code base isn't that hard.

Prior to AI2, I have never touched a large code base, only started mini-projects / coded stuff in my spare time / did work for school projects. Maybe because there was not a need, but perhaps because it appeared intimidating. Too many lines. Hard to build. How do I navigate / where do I start?! With practice, it got better.

Somehow, IntelliJ helped (especially with the CTRL + SHIFT + N quick navigation keys, and the target buttons), and being exposed to the code base proved helpful. What really helped though was keeping in touch with my mentor, and discussing with him problems which seem to crop up.

A word of advice: If the docs do not seem too informative, and you tried, approach the community online for help! It really helps when you do, and especially when someone can help you navigate through things.

TL;DR

Contributing to the MIT App Inventor 2 codebase under the Facebook Open Academy Program in Winter 2015 was a really enjoyable experience. What worked was having a great mentor, and coding something that we really believed in, and enjoyed doing. A new component was drafted, and we had the opportunity to learn things outside the box and would love to see this project to fruition - adoption to the community at large is another phase of learning.

Gained more confidence with contributing to Open Source? Yes I did. Will do it again? Yes, sure will!

And, thank God for a wonderful semester.

Acknowledgement

The past few weeks were enjoyable, fulfilling and enlightening. Besides levelling up as a web developer, and gaining experience in inter-platform (embedded) development, the following people have contributed greatly to mentoring the project. They are:

Jos Flores My mentor, who was instrumental in bringing the team together, giving advice and advising us to listen to the community's needs instead of jumping into the code straight. He was a great and patient teacher, never discounting any question, despite his busy schedule.

He got me up and running with Git, and through him I have learnt much about the App Inventor workflow, and contributing to the codebase - something that would have taken a considerable longer time had he not been present.

Prof. Ben Leong Our course instructor, who provided us with considerable advice that agurs well beyond the scope of this program.

I recall his ancedotes as a programmer, never failing to give up but more importantly, re-examining the process of learning and making corrections where possible, in addition to the importance of estimating our code speed given the timeframe. He has a heart and passion for teaching, which clearly inspired the whole team here at NUS.

The community The community, for giving ever needed timely advice and support during the development. Their willingness to help on a voluntary project attests to their passion for contributing to App Inventor, as well as helping beginners get involved in code through education (learning via code blocks).

Appendix A: Working Spec

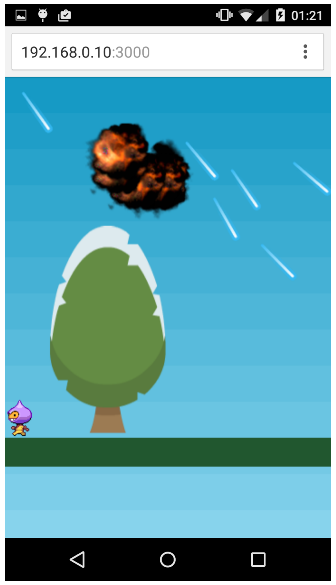
Proposal: A Game Editor with App Inventor 2 and Phaser.io

Joel Haowen Tong | Mentor: José Flores

UPDATE: Implemented version ready! Please see https://groups.google.com/d/msg/app-inventor-open-source-dev/3KHLNn2l-vQ/w_JzfzEvJWUJ
Current Stable Version: v0.1-alpha

Hello everyone,

We are keen to use App Inventor to create mobile games. Specifically, we are attempting to make this possible via a webapp interface powered by the Javascript-based engine, Phaser.io (www.phaser.io) . We propose implementing a balanced yet rich subset of Phaser.io that is powered by a mixture of Webview, Javascript and YAIL, to create something that looks like this:



Credit: Sprites from http://www.photonstorm.com/phaser/tutorial-making-your-first-phaser-game

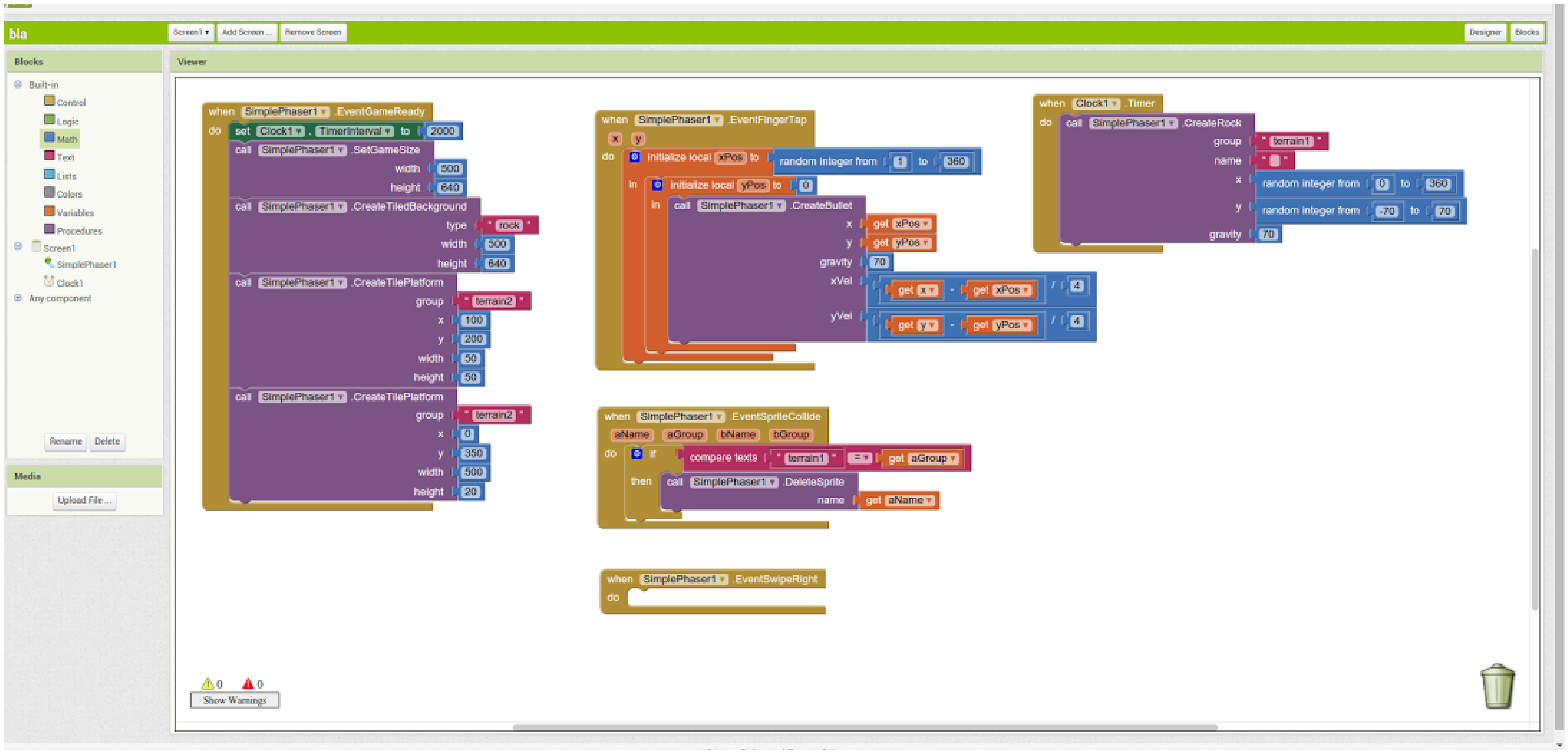
Try SimplePhaser!

SimplePhaser v0.1-alpha is ready for testing! The game only works in the APK (compiled version of app). Here's how you can test it:

1. Pull the workig AI2 SimplePhaser branch from https://github.com/myrtleTree33/appinventor-sources/commit/7b4efa76e31a710bb9aa4c12e2947c108fb846f0
2. Compile the code using `ant`, and the updated companion app using `ant comps`
3. View the API here: https://docs.google.com/document/d/1s-HOWTea0YBLP4UpYteeS94IgB7H-SlXPmQJ9hxhrGs/edit
4. Try a simple program!
5. Generate APK and upload it to phone.

Try the sample game below!

A sample game is shown below. In this game, rocks fall from the sky and you can clear rocks by tapping on the screen, which fires bullets.
A preview of it on Youtube: https://www.youtube.com/watch?v=4HriQzQxUFs



```
when SimplePhaser1 .EventGameReady
do  set Clock1 . TimerInterval to 2000
    call SimplePhaser1 .SetGameSize
                       width 500
                       height 640
    call SimplePhaser1 .CreateTiledBackground
                       type " rock "
                       width 500
                       height 640
    call SimplePhaser1 .CreateTilePlatform
                       group " terrain2 "
                       x 100
                       y 200
                       width 50
                       height 50
    call SimplePhaser1 .CreateTilePlatform
                       group " terrain2 "
                       x 0
                       y 350
                       width 500
                       height 20
```

```
when SimplePhaser1 .EventFingerTap
  x  y
do  initialize local xPos to random integer from 1 to 360
    in  initialize local yPos to 0
        in  call SimplePhaser1 .CreateBullet
                               x get xPos
                               y get yPos
                               gravity 70
                               xVel ( get x - get xPos ) / 4
                               yVel ( get y - get yPos ) / 4
```

```
when Clock1 .Timer
do  call SimplePhaser1 .CreateRock
                       group " terrain1 "
                       name " "
                       x random integer from 0 to 360
                       y random integer from -70 to 70
                       gravity 70
```

```
when SimplePhaser1 .EventSpriteCollide
  aName  aGroup  bName  bGroup
do  if  compare texts " terrain1 " = get aGroup
    then  call SimplePhaser1 .DeleteSprite
                             name get aName
```

```
when SimplePhaser1 .EventSwipeRight
do
```

Rough overview

Phaser is a very rich toolset and employs function chaining heavily. On the other hand, while building a blocks interface for Phaser direct might be a possible option, it does not fit well into the current component scheme of App Inventor. Furthermore, nesting of code blocks is complicated by the relatively simple nature of 2-way control between Android Java and the webview (calling "javascript" and adding annotations to expose Java functions). This introduced many problems into the implementation of a game.

We propose a reduced yet rich set of Phaser features that should work in tandem with AI2. To target a users new to programming, we suggest the implementation of the following code blocks as follow.

Programming structure can be built according to one's experiences with gaming, instead of feature-heavy / complex API calls and Math.

A game comprises of 2 compulsory stages:

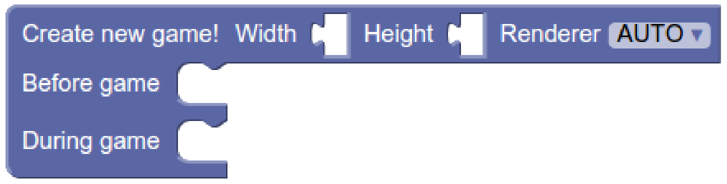
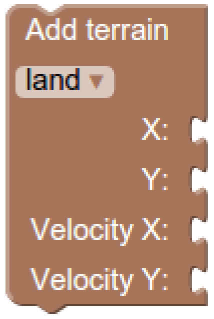
- Before the game, where the world and actors are set up,
- During the game, where the world, player and bots interact with each other.
- End of game, which is optional.

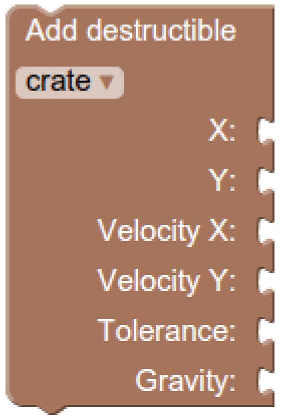
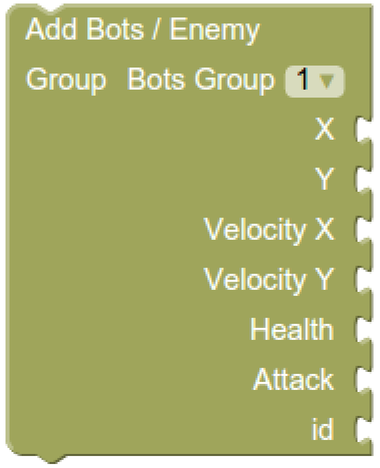
The user is free to create the world from the existing web editor, at any of the 2 compulsory stages. Various objects such as bots, destructible world objects can be created.

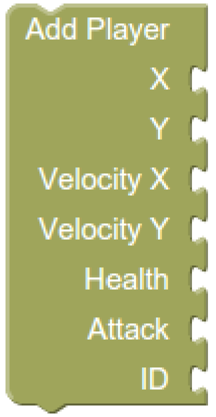
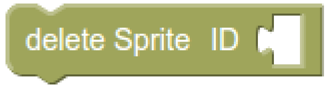
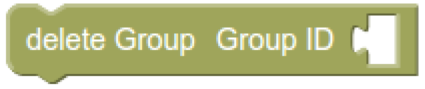
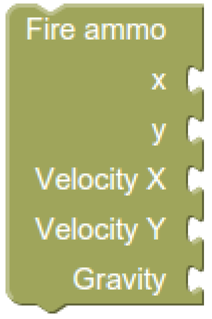
Basic behavior is programmed into these objects, freeing the user from implementing these basic functions. For example, ammunition rounds should destruct and play a destruction animation, with sound, upon collision with the floor. Destructible objects should play their own animations and have the physics and extra code, such as gravity, pre-programmed into the sprite.

A rough detail of the blocks as follow:

World population blocks

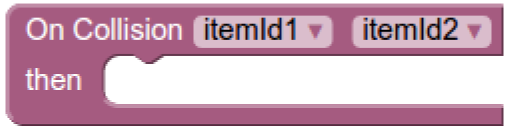
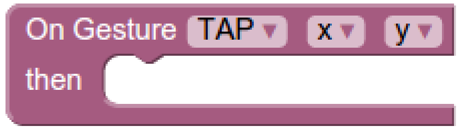
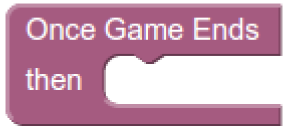
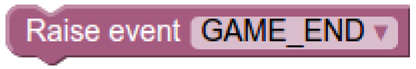
Block	Description	Comments
 Create new game! Width Height Renderer AUTO Before game During game	What: Creates a new SimplePhaser game, included by default. Parameters: width: Width of game height: Height of game renderer: AUTO | WEBGL | CANVAS Blockly hotlink: https://blockly-demo.appspot.com/static/demos/blockfactory/index.html#e5ho37	There exists an onLoad() routine in vanilla Phaser. Resources are preloaded by default.
 Add terrain land X: Y: Velocity X: Velocity Y:	What: Creates a non-destructible terrain sprite. Parameters: x: X position y: Y position Velocity X: X Velocity Velocity Y: Y Velocity Blockly Hotlink: https://blockly-demo.appspot.com/static/demos/blockfactory/index.html#bp4vdd	These sprites do not react to gravity.

 Add destructible crate X: Y: Velocity X: Velocity Y: Tolerance: Gravity:	What: Creates a destructible block, which destructs after \`tolerance\` hits. Parameters: x: X position y: Y position Velocity X: X Velocity Velocity Y: Y Velocity Tolerance: When the block will destruct. Gravity: G Force to exert on object. Blockly Hotlink: https://blockly-demo.appspot.com/static/demos/blockfactory/index.html#m94b9n	These sprites react to gravity.
 Add Bots / Enemy Group Bots Group 1 X Y Velocity X Velocity Y Health Attack id	What: Create new bots / enemies Parameters: Group: botsGroup1 | 2 | 3 x: X position y: Y position Velocity X: X Velocity Velocity Y: Y Velocity Health: Starting health Attack: Attack capacity Blockly Hotlink: https://blockly-demo.appspot.com/static/demos/blockfactory/index.html#q8mtpu	Bots can be added to a maximum of 3 control groups, for easy manipulation.

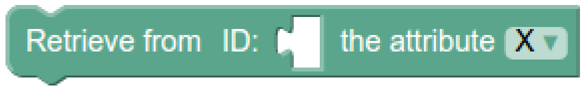
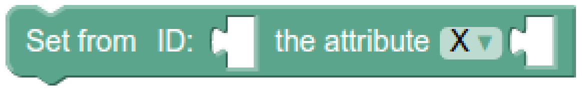
 Add Player X Y Velocity X Velocity Y Health Attack ID	What: Creates a new player. Parameters: x: X position y: Y position Velocity X: X Velocity Velocity Y: Y Velocity Health: Starting health Attack: Attack capacity Blockly Hotlink: https://blockly-demo.appspot.com/static/demos/blockfactory/index.html#f7jqc5	Should this be colored differently? For uniformity, player is added to a control group containing one sprite, player. The player should have multiple sprites. For example, it could be a hero or it could be a spaceship for an invaders game.
	What: Deletes a sprite by ID. Blockly Hotlink: https://blockly-demo.appspot.com/static/demos/blockfactory/index.html#ifi7f9	
	What: Deletes a group by ID. Blockly Hotlink: https://blockly-demo.appspot.com/static/demos/blockfactory/index.html#b3f4qs	
	What: Fires ammo, which deletes itself when out of screen on collision with object. Parameters: x: X position y: Y position Velocity X: X Velocity Velocity Y: Y Velocity Gravity: Specifies gravity exerted on ammo. If zero, no gravity exerted.	Features gravity effect on ammo. Implements behavior to delete itself on collision with any object or out of bounds.

	Blockly Hotlink: https://blockly-demo.appspot.com/static/demos/blockfactory/index.html#2ehyxz	

Events and Event handlers

	What: On Collision Parameters: itemId1: The ID of item1 that hit ID of item2. itemId2: The item that got hit. Blockly Hotlink: https://blockly-demo.appspot.com/static/demos/blockfactory/index.html#xcyieg	In the original Phaser API, collision events must be explicitly specified between 2 events. A unified event dispatcher will have to be constructed to filter for the specific event.
	What: On Gesture Parameters: X: X position of gesture Y: Y position of gesture Blockly Hotlink: https://blockly-demo.appspot.com/static/demos/blockfactory/index.html#yhmnw9	This replaces keyboard events. As Phaser.io does not have event support, we intend to use Hammer.js
	What: Event handler for GAME OVER. Blockly Hotlink: https://blockly-demo.appspot.com/static/demos/blockfactory/index.html#s6kpim	Once the game has ended
	What: Raises a game event. Blockly Hotlink: https://blockly-demo.appspot.com/static/demos/blockfactory/index.html#6kiwoz	Can be used to issue a GAME END.

Getters and Setters

	What: Retrieve a property from sprite Parameters: ID: The ID of sprite. x: X position y: Y position Velocity X: X Velocity Velocity Y: Y Velocity Health: Starting health Gravity: Gravity Blockly Hotlink: https://blockly-demo.appspot.com/static/demos/blockfactory/index.html#fa6m84	A corresponding group getter will be available. References to object will be additionally stored in Javascript side in a dictionary (key-value store), where the ID is the key, and the reference to object is the store.
	What: Set a property from sprite Parameters: ID: The ID of sprite. x: X position y: Y position Velocity X: X Velocity Velocity Y: Y Velocity Health: Starting health Gravity: Gravity Blockly Hotlink: https://blockly-demo.appspot.com/static/demos/blockfactory/index.html#t88dcp	A corresponding group setter will be available. References to object will be additionally stored in Javascript side in a dictionary (key-value store), where the ID is the key, and the reference to object is the store.

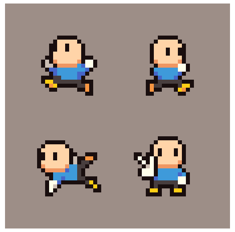
The status of all implemented functions depicted in table below:

API	Implemented?		Comments
	Javascript	Java	
CreateSky()	Done	Done	
CreatePlatform(group,x,y)	Done	Done	
CreateTilePlatform(goup,x,y,width,height)	Done	Done	
CreateRock(group,x,y,gravity)	Done	Done	
CreateTree(group,x,y,gravity)	Done	Done	
CreateBullet(x,y,gravity,xVel,yVel)	Done	Done	
CreatePlayer(group,name,x,y,gravity)	Done	Done	
DeleteSprite(name)	Done	Done	
SetPosition(name,x,y)	Done	Done	Refactor to use actions scope instead.
SetState(name, state)	Done	Done	
GenerateGame()	Done	Done	To deprecate; should be called instantly
GameWidth()	Done	Done	
GameHeight()	Done	Done	
SpriteX(name)	Done	Done	
SpriteX(name, x)	Done	Done	
SpriteY(name)	Done	Done	
SpriteY(name, y)	Done	Done	

SpriteVelX(name)	Done	Done	
SpriteVelX(name, velX)	Done	Done	
SpriteVelY(name)	Done	Done	
SpriteVelY(name, velX)	Done	Done	

Sprite list

The following sprite images will be preloaded for use. Placeholders from stock Phaser.io library will be used in development phase.

	Hero	http://opengameart.org/content/classic-hero
	Terrain	http://opengameart.org/content/outside-tileset
	Enemy 1	
	Enemy 2	
	Enemy 3	
	Barrel	
	Bush	
	Rock	
	Platform	
	Crate	
	Barrel	
	Tree	
	Shrub	
	Water	to be implemented

Next Steps

For next steps (Iteration 2 of development, after v1.0 is released), the following branches are suggested:

Phase	Description	Comments
1	Improve user input by identifying tap sequences. For example, tap - double tap double tap - tap could mean a sequence for a game fighter series for example.	
2	Implement behaviors. Various Artificial Intelligence Path finding / graph search can be implemnted for enemies to find the player, for example.	
3	Richer sprites. Behaviors is an ongoing experimentation within SimplePhaser. Getting sprites in with behaviors is a good start.	
4	MIDI support with LibPD. This should be a standalone component in AI2 and allows one to create MIDI music.	

The Implementation: How it works

Prior background information on Phaser.io

Phaser is a 2D Javascript framework used to write games on the web. It features heavily on using function chains, and is widely used by developers. One can easily create a game in Phaser within half an hour, given the right sprites. It supports either of WebGL or Canvas.

A Phaser game has the structure signature:

```
var game = new Phaser.Game(800, 600, Phaser.CANVAS, '', {
     preload: preload,
     create: create,
     update: update
});
```

Where preload, create, and update are functions. The preload function block does game bootstrapping, for example, preloading sprite image resources and sound files, and setting up other variables. Once all resources are loaded, the create block is triggered, which is fired once to set up the world. These can be used to set up event handlers / collision handlers as well. Thereafter, the update function block is run on every frame.

Hence inserting and injecting code within these code blocks, coupled with the eval (evil!) function, it is possible to dynamically create a game using some external Javascript code and hence, AI2.

Implementation

While it is indeed possible to pass and maintain state between a Javascript and Webview interface, through some form of markup, this is indeed heavy, complex and inefficient. A better implementation would be to maintain the game state in a Javascript-heavy environment, while reporting certain key events to App Inventor 2, App-side.

Hence the SimplePhaser component addresses these issues through the following:

1. Code injection into target functions which is intepreted via the Javascript Just-In-Time (JIT) interpreter,
2. A heavy emphasis on keeping states via a combination of Javascript and Yail code,
3. The usage of a sprite key-value store (the “accumulator”) to store the state of sprites,
4. Fixed control groups of “terrain”, “destructibles”, “player” and “enemy” that features general in-built behaviors. For example, an enemy should be programmed to find a player sprite and attack it without much heavy-lifting of math and possibly Artificial Intelligence path-routing code.
5. An event-driven framework for tasks.

The Javascript backend: The Blast Framework structure

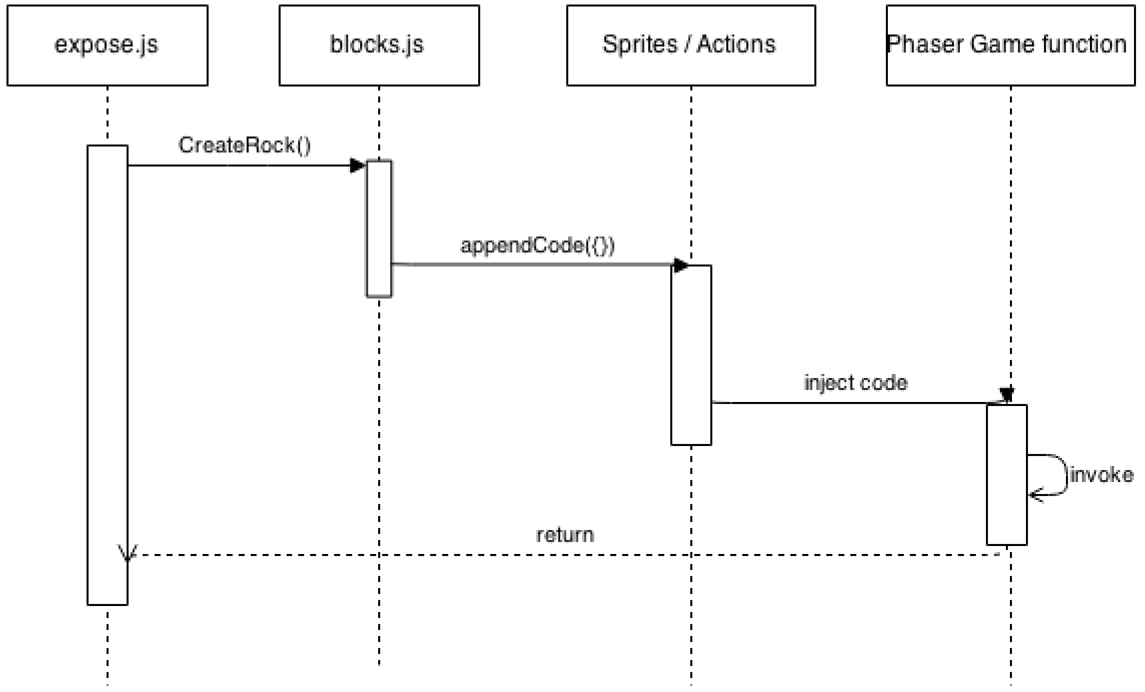
The current Blast prototype adopts the Browserify module-based (require) toolchain. Each file handles a different scope:

Module	Description	Notes
blastFramework_bootstrap.js	Bootstraps any necessary functions, assets or libraries needed for Phaser. In-game assets (images, sound, etc) not loaded here.	
blastFramework.js	Contains the core skeletal framework that runs by itself.	
blastFramework_actions.js	Appends the actions and miscellaneous getters and setters that can control sprites.	
blastFramework_behaviors.js	Specifies the behaviors and callbacks of the relevant functions.	TODO
blastFramework_sprites_terrain.js	Contains necessary information to create sprites	
blastFramework_blocks.js	Wraps all lower-level code injection blocks to create sprites, introduce getter and setters, etc. into a higher level API. The module's focus is for the purposes of code injection.	
blastFramework_expose.js	Exposes the endpoints to be linked to the App Inventor Java interface.	
scratch.js	A scratchpad which pulls everything together.	

Files attach themselves to the Blast Framework via mixins with the Blast framework singleton object in Javascript.

The callback cycle: How code is injected:

The following diagram illustrates how the callback cycle for calling commands is implemented. In this case, creating a rock sprite:



Expose.js

This defines the endpoints for the Java interface, and calls the blocks function below. As shown is a following code snippet:

```
api.prototype.CreateRock = function (x, y, gravity) {
  $blast.appendCode('onCreate', 'createRock', {
     x: x,
     y: y,
     gravity: gravity
  });
};
```

As shown, the framework handles injection into the code snippet automatically. Internally, `appendCode` generates an anonymous function, which can be unwrapped into the corresponding target function. It can be also specified to be inserted into e.g. runtime after a certain event. (Events are not handled directly by an event handler, but are instead inserted into a priority queue and executed during the game execution). Hence, the entire game is populated on the fly by runtime injections.

Blocks.js

Blocks specifies the code specifically to be injected. Hence it is a lower-level implementation which injected wrapped Phaser code sprites in. An example for the Rocks sprite.

```
__blocks.createRock = function (opts) {
  var opts = opts || {};
  var x = opts.x || 0;
  var y = opts.y || 0;
  var gravity = opts.gravity || 30;
  var code = 'var sprite = $blast.sprite.generators.rock(' + x + ','
     + y + ',' + gravity + ').init();\n';
  return code;
};
```

sprites_terrain.js

The actual rock sprite and its creation and deletion handlers are specified in this file. All sprites derive and implement SimpleSprite. This is performed behind the scenes as a mixin, where the scope of the object is passed in. The SimpleSprite makes the object accessible by registering it with the accumulator, and deregistering it when it is killed.

A sprite implementation looks like the following:

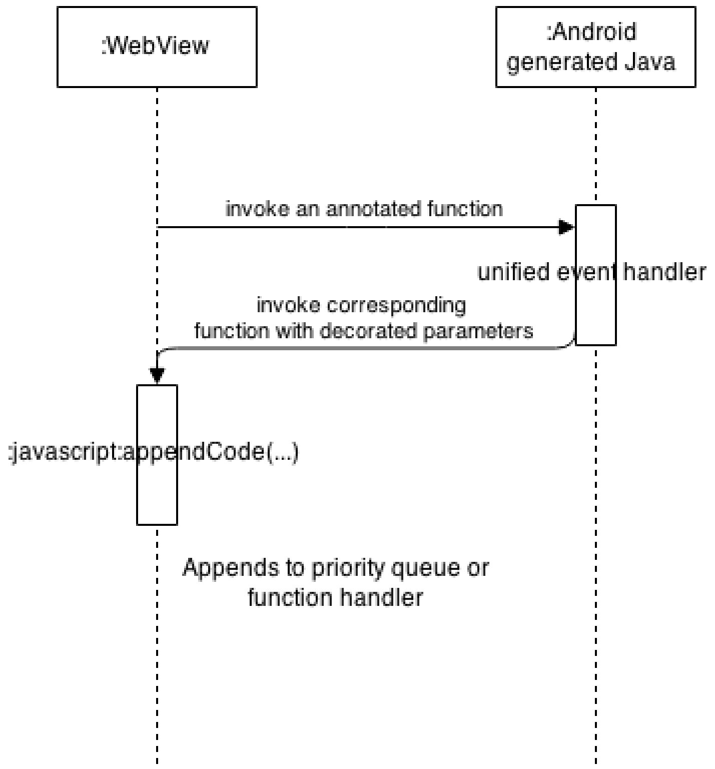
```
/**
* This is an example of an extended object
* @param name
* @returns {*}
*/
__generators.sampleSprite = function (name) {
  var nativeObject = __generators.SimpleSprite(name);

  var init = function () {
     nativeObject._init(this);
     /** Insert code here **/
    /** End insert code here **/
     console.log("I got extended!!");
  };

  var kill = function () {
     /** Insert code here **/
    /** End insert code here **/
     nativeObject._kill();
  };

  var hello = function () {
  };

  return _.extend({}, nativeObject, {
     init: init,
     kill: kill,
     hello: hello
  });
};
```

An example for the Rock Sprite, which features an explode upon kill animation:

```
__generators.rock = function (x,y, gravity) {
  var nativeObject = __generators.SimpleSprite('rock1');
  //var nativeObject = __generators.SimpleSprite();

  var init = function () {
    var rock = $blast._groups.destructibles.create(x, y, 'firstaid');
    rock.group = "destructibles";
    rock.body.gravity.y = gravity;
    rock.body.bounce.y = 0.7 + Math.random() * 0.2;
    rock.outOfBoundsKill = true;
    rock.body.collideWorldBounds = true;
    this.obj = rock; // add to object
    nativeObject._init(this);
    console.debug("Init rock at x=" + x + ',' + y);
  };

  var kill = function () {
    console.log("=KILL= I got called");
    var explosion = $blast._game.add.sprite(this.obj.x, this.obj.y,'explosion');
    explosion.anchor.setTo(0.5,0.5);
    var anim = explosion.animations.add('explode', null, 60, false);
    anim.killOnComplete = true;
    anim.play('explode');
    anim.onComplete.add(function() {
      console.log('Explosion played.');
      explosion.kill();
    });
    this.obj.destroy();
    nativeObject._kill();
  };

  return _.extend({}, nativeObject, {
    init: init,
    kill: kill
  });
};
```

Binding to the Java interface

Events are first issued by the webview, and then raised at the Android Java end. This triggers generated code by AI2 which is processed. Eventually, the corresponding Phaser Blockly block is called by call to a Javascript function, which injects the required code dynamically into the target function. A call overview is shown:


:WebView
:Android generated Java
invoke an annotated function
unified event handler
invoke corresponding function with decorated parameters
:javascript:appendCode(...)
Appends to priority queue or function handler

Sprite Groups

Group	Description	Comments
background	for background sprites (blue sky, etc)	
terrain1	For inanimate terrain (i.e. no gravity applied)	
terrain2	For animate terrain	
destructibles	For destructible objects (e.g. crates, bottles, etc)	
player1	Friendly sprites	
player2	Friendly sprites	
enemy1	Enemy1	
enemy2	Enemy2	
enemy3	enemy3	
fire	For objects which cause damage. Bullets, etc.	

Groups can be controlled. Multiple groups are analogous to fixed multiple movie channels in movie editing software; they make it easy for beginner programmers to create a structure between groups easily.

Sprite Groups

Description	URL
Blast Framework (Javascript) repository. Checkout develop branch.	https://github.com/myrtleTree33/TestBlockly/tree/develop
Java implementation of SimplePhaser component (in development)	https://github.com/myrtleTree33/appinventor-sources/tree/addSimplePhaser-v2

Changelog

Feb 24, 2015	Initial spec v0.1.0 released
Feb 24, 2015	v0.1.1 - Added data structures and refactored misleading labels
Mar 24, 2015	v0.2 - Updated with current implementation and group structure; next steps
Mar 27, 2015	Updated mobile graphics with actual game in development
Apr 22, 2015	v0.1-alpha released, updated with testing instructions

Appendix B: SimplePhaser API v0.1-alpha

SimplePhaser API

`Version: v0.1-alpha`

Key

Type	Color
Event	🟨
Function	🟪
Getter / Setter	🟩

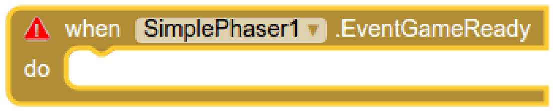
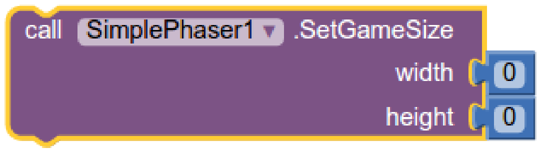
Valid Groups:

Group	Description	Comments
terrain1	For terrain objects	
terrain2	For terrain objects	
bullet	For bullets	Do not use
destructibles	For objects that destroy.	

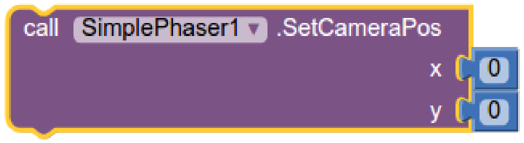
NB:

If sprite names are empty during instantiation, they are assigned randomly.

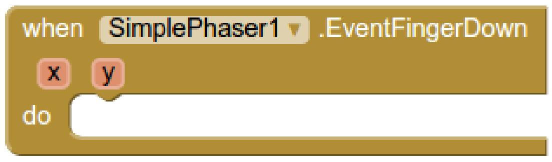
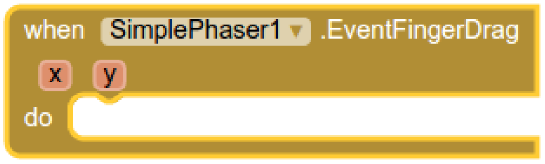
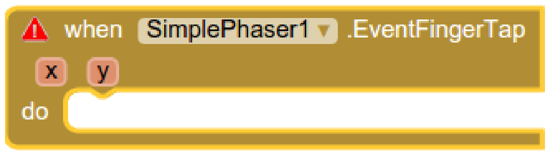
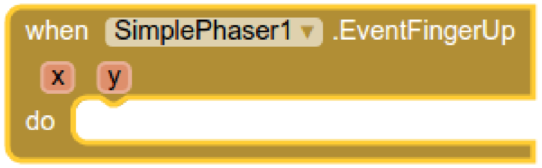
Game

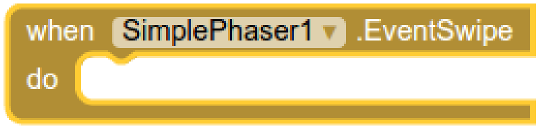
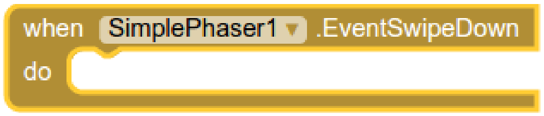
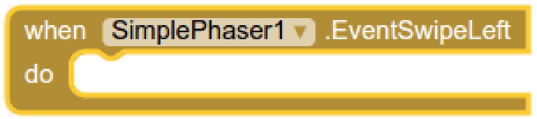
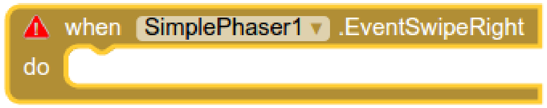
Block	Specification	
	Triggered when game is ready.	All code should start from this block.
	Sets the size of the game world.	Can be larger than screen size, in which case change the view of the camera.

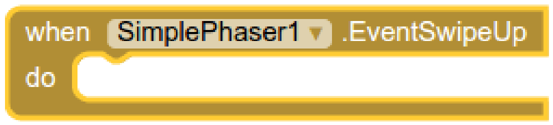
Camera

Block	Specification	Comments
	Sets the Camera position Params: x: x-coordinate y: y-coordinate	

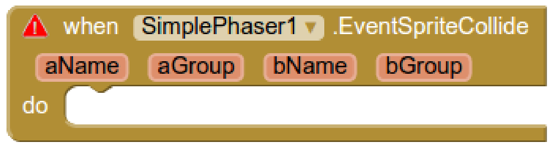
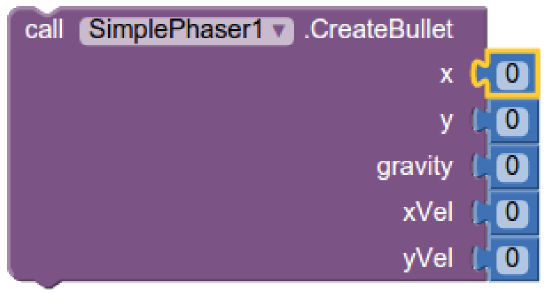
User Interaction

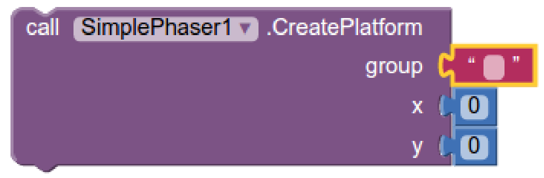
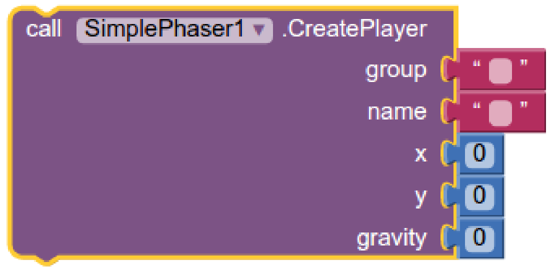
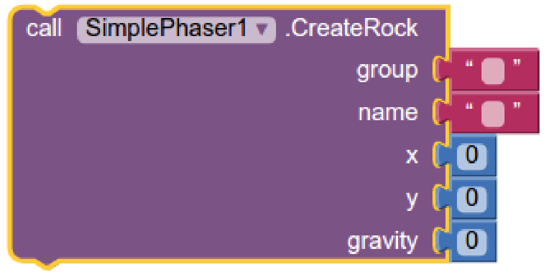
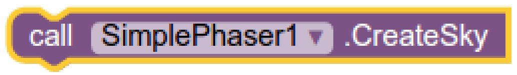
Block	Specification	
 when SimplePhaser1 .EventFingerDown x y do	EventFingerDown Triggered when finger down. Returns: x: x-coordinate of tap, with respect to world. y: y-coordinate of tap, with respect to world.	
 when SimplePhaser1 .EventFingerDrag x y do	EventFingerDrag Triggered when finger drag. Returns: x: x-coordinate of tap, with respect to world. y: y-coordinate of tap, with respect to world.	
 when SimplePhaser1 .EventFingerTap x y do	EventFingerTap Triggered when finger tap. Returns: x: x-coordinate of tap, with respect to world. y: y-coordinate of tap, with respect to world.	
 when SimplePhaser1 .EventFingerUp x y do	EventFingerUp Triggered when fingerup. Returns:	

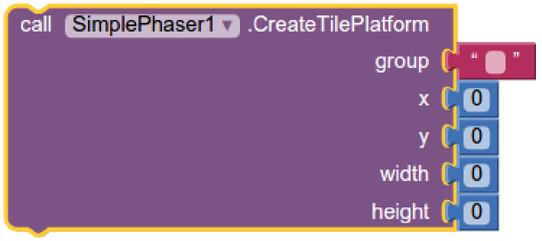
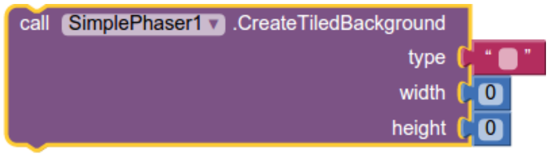
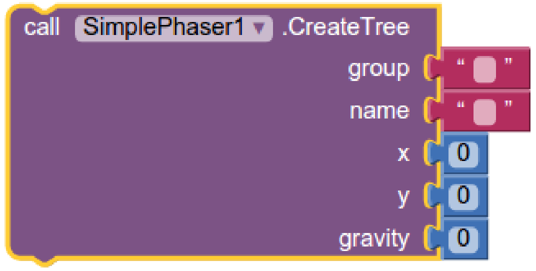
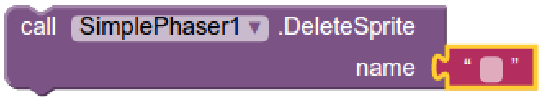
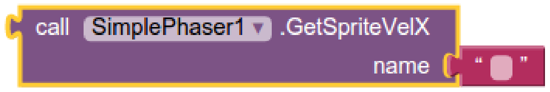
	x: x-coordinate of tap, with respect to world. y: y-coordinate of tap, with respect to world.	
	EventSwipe Triggered when swipe. Returns: x: x-coordinate of swipe, with respect to world. y: y-coordinate of swipe, with respect to world.	
	EventSwipeDown Triggered when swipe down. Returns: x: x-coordinate of swipe, with respect to world. y: y-coordinate of swipe, with respect to world.	
	EventSwipeLeft Triggered when swipe left. Returns: x: x-coordinate of swipe, with respect to world. y: y-coordinate of swipe, with respect to world.	
	EventSwipeRight Triggered when swipe right. Returns: x: x-coordinate of swipe, with respect to world. y: y-coordinate of swipe, with respect to world.	

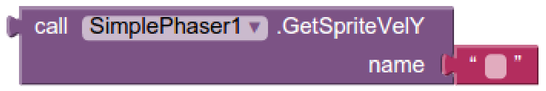
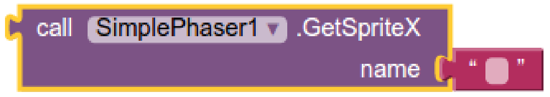
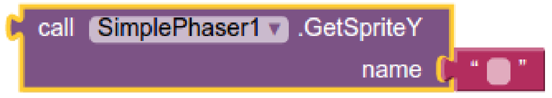
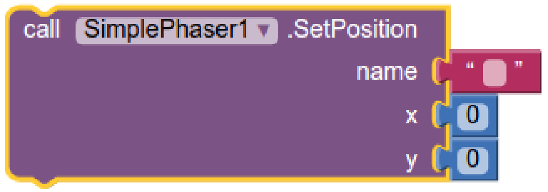
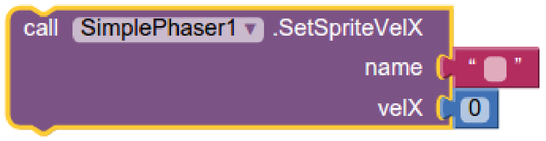
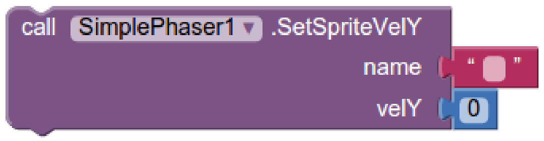
	EventSwipeUp Triggered when swipe up. Returns: x: x-coordinate of swipe, with respect to world. y: y-coordinate of swipe, with respect to world.	

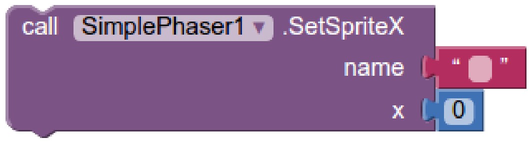
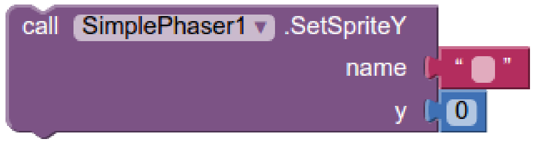
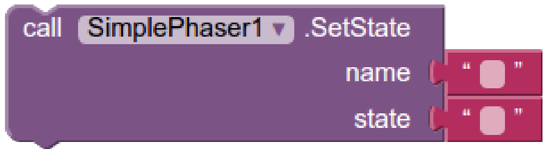
Sprites

Block	Specification	Comments
 when SimplePhaser1 .EventSpriteCollide aName aGroup bName bGroup do	EventSpriteCollide Triggered when 2 sprites collide. Returns: aName: Name of collided object. aGroup: Group of collided object. bName: Name of collided object. bGroup: Group of collided object.	Sprite A is guaranteed not to be a bullet. If Sprite B is a bullet, then the sprite B will be invalid by time of call.
 call SimplePhaser1 .CreateBullet x 0 y 0 gravity 0 xVel 0 yVel 0	Creates a bullet. Params: x: x-coordinate y: y-coordinate gravity: gravity xVel: Velocity X yVel: Velocity Y	Destroys upon impact.

 call SimplePhaser1 .CreatePlatform group x y 0 0	Creates a platform. Params: Group: Groupname x: x-coordinate y: y-coordinate	
 call SimplePhaser1 .CreatePlayer group name x y gravity 0 0 0	Creates a player sprite. Params: group: Group name: name x: x-coordinate y: y-coordinate gravity: gravity	
 call SimplePhaser1 .CreateRock group name x y gravity 0 0 0	Creates a player sprite. Params: group: Group name: name x: x-coordinate y: y-coordinate gravity: gravity	
	Creates a sky background.	

 call SimplePhaser1 .CreateTilePlatform group x y width height	Creates a repeating tile platform. Params: group: Group x: x-coordinate y: y-coordinate width: width height: height	
 call SimplePhaser1 .CreateTiledBackground type width height	Creates a tiled background. Params: group: Group width: width height: height	
 call SimplePhaser1 .CreateTree group name x y gravity	Creates a tree. Params: group: Group name: name x: x-coordinate y: y-coordinate gravity: gravity	
	Deletes a sprite. Params: name: name of sprite to delete	
	Gets the sprite velocity X. Params:	

	name: Name of sprite to retrieve	
	Gets the sprite velocity Y. Params: name: Name of sprite to retrieve	
	Gets the sprite Y. Params: name: Name of sprite to retrieve	
	Gets the sprite X. Params: name: Name of sprite to retrieve	
	Sets the position. Params: name: Name of sprite X: X coordinate Y: Y coordinate	
	Sets the sprite velocity. Params: name: Name of sprite velX: Velocity X	
	Sets the sprite velocity. Params: name: Name of sprite	

	velY: Velocity Y	
	Sets the sprite position. Params: name: Name of sprite X: Coordinate X	
	Sets the sprite position. Params: name: Name of sprite Y: Coordinate Y	
	Sets the sprite state. Params: Name: name of sprite State: State of sprite	only applies to player sprite

<INSERT NAME HERE>

Block	Specification	Comments