

PROPOSAL: A GAME EDITOR WITH APP INVENTOR 2 AND PHASER.IO

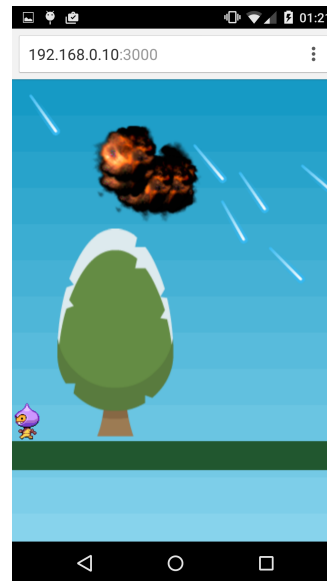
JOEL HAOWEN TONG | MENTOR: JOSÉ FLORES

UPDATE: Implemented version ready! Please see https://groups.google.com/d/msg/app-inventor-open-source-dev/3KHLNn2l-vQ/w_lzfzEvjWUJ

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-HOWTea0YBLP4UpYteeS94lgB7H-SIXPmQJ9hxhrGs/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>

bla Screen1 Add Screen ... Remove Screen Designer Blocks

Blocks

- Built-in
 - Control
 - Logic
 - Math
 - Text
 - Lists
 - Colors
 - Variables
 - Procedures
- Screen1
- SimplePhaser1
- Clock1
- Any component

Rename Delete

Media

Upload File ...

Viewer

```

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
do
  in x y
  initialize local xPos to random integer from 1 to 360
  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
do
  if compare texts terrain1 = get aGroup
  then call SimplePhaser1.DeleteSprite
    name get aName

when SimplePhaser1.EventSwipeRight
do

```

0 0
Show Warnings



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:

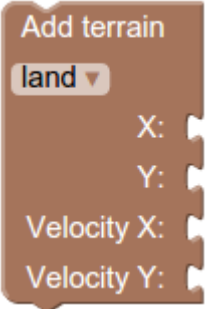
- Before the game, where the world and actors are set up,
- During the game, where the world, player and bots interact with each other.
- End of game, which is optional.

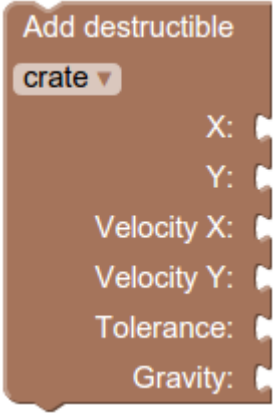
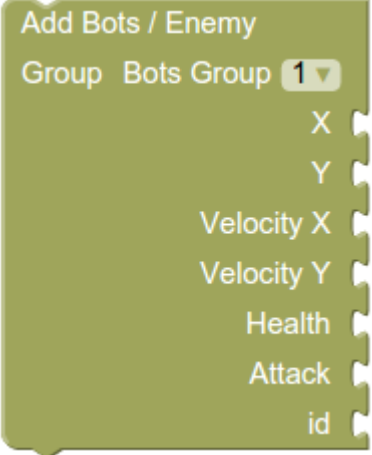
The user is free to create the world from the existing web editor, at any of the 2 compulsory stages. Various objects such as bots, destructible world objects can be created.

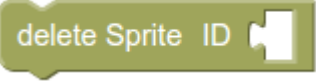
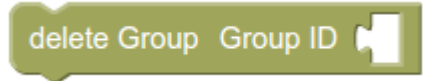
Basic behavior is programmed into these objects, freeing the user from implementing these basic functions. For example, ammunition rounds should destruct and play a destruction animation, with sound, upon collision with the floor. Destructible objects should play their own animations and have the physics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

A rough detail of the blocks as follow:

WORLD POPULATION BLOCKS

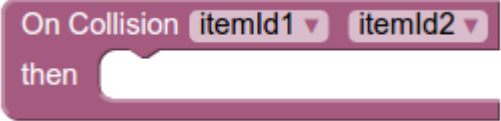
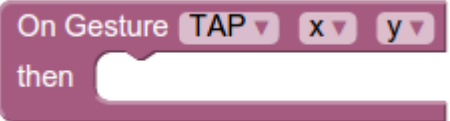
Block	Description	Comments
	What: Creates a new SimplePhaser game, included by default. Parameters: width: Width of game height: Height of game renderer: AUTO WebGL CANVAS Blockly hotlink: https://blockly-demo.appspot.com/static/demos/blockfactory/index.html#e5ho37	There exists an onLoad() routine in vanilla Phaser. Resources are preloaded by default.
	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.

	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.
	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.

	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#yhmwn9	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 interpreted 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

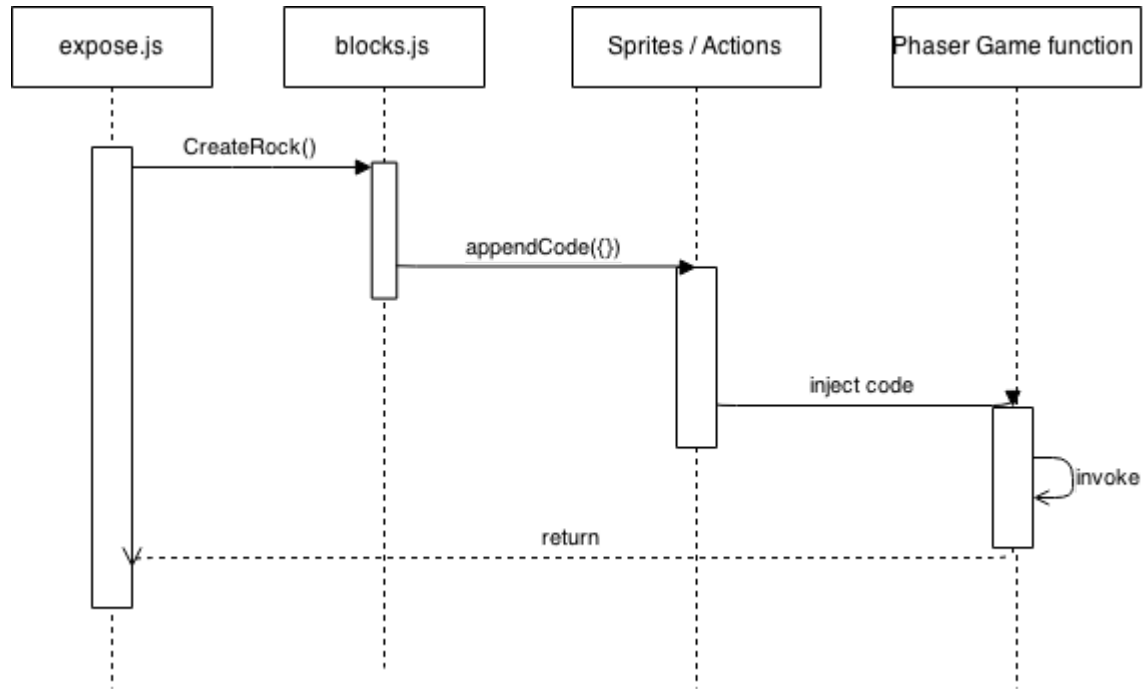
The current Blast prototype adopts the Browserify module-based (require) toolchain. Each file handles a different scope:

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);
  };

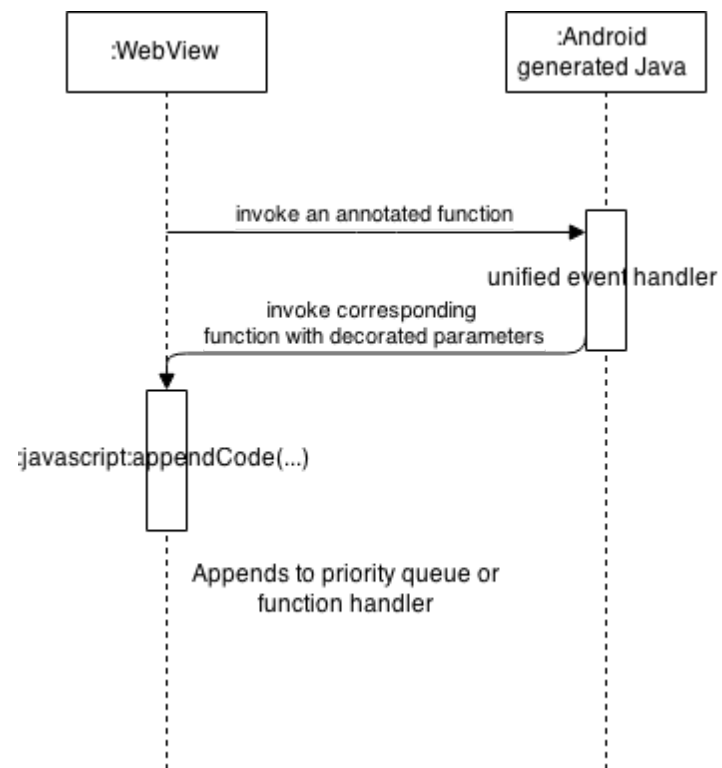
  var kill = function () {
    console.log("=KILL= I got called");
    var explosion = $blast._game.add.sprite(this.obj.x, this.obj.y, 'explosion');
    explosion.anchor.setTo(0.5,0.5);
    var anim = explosion.animations.add('explode', null, 60, false);
    anim.killOnComplete = true;
    anim.play('explode');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



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