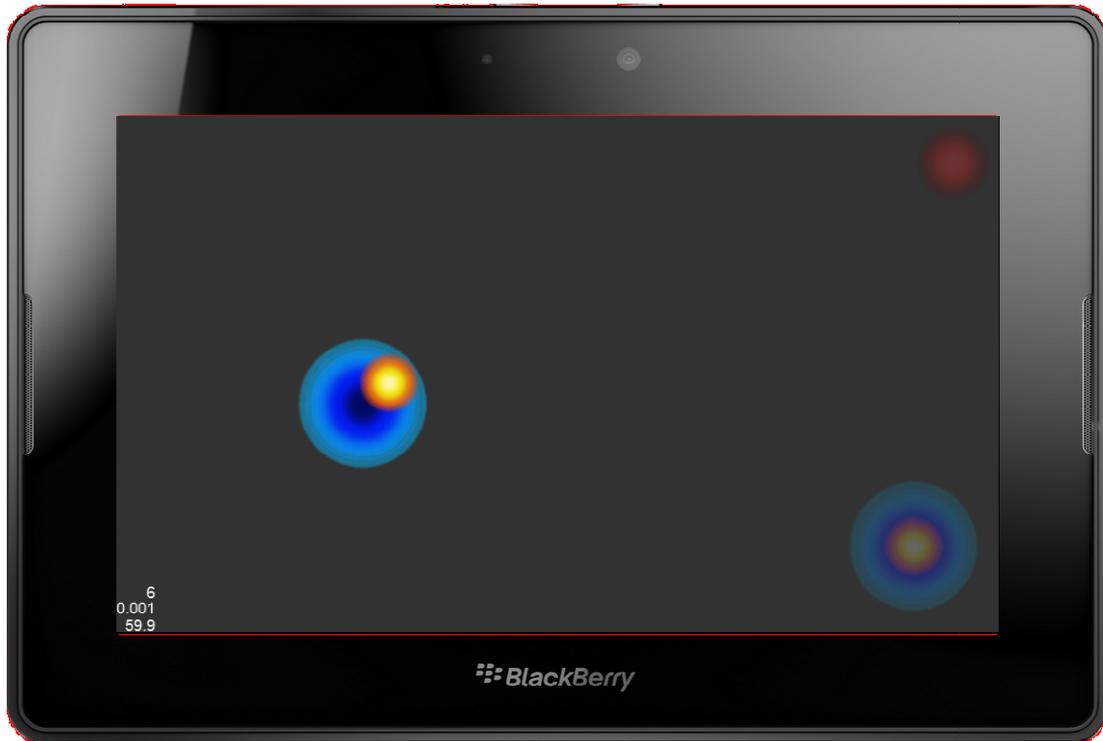


Cocos2d-HTML5: Virtual Controls Overlay



Sample usage showing two Joysticks and one Button.

Introduction

There are a lot of games that web developers design with the desktop-space in mind; creative, enjoyable games that would be suited for the mobile-space as well. One of the largest hurdles that regularly arise is the difference in control mechanics on a desktop (mouse and keyboard) compared to that in the mobile-space. In this case, we will be focussing on all-touch mobile devices like the BlackBerry PlayBook.

Provided in this document is the ControlsOverlay.js framework that is intended to simplify the process for Cocos2d-HTML5 developers in bringing their desktop-focussed application to the BlackBerry PlayBook. This framework is distributed by Research In Motion under the Apache License version 2.0.

Occasionally, you may see references to Cocos2d-HTML5 specific components such as Points or Rects, these will be designated with an appended **cc** identifier. For example, **cc.Rect** refers to the rectangle Class in the Cocos2d-HTML5 framework. More information on these Classes can be found in the [Cocos2d-HTML5 documentation](#).

What Does the Framework Provide

With ControlsOverlay.js, developers can implement any number of Joystick and Button controls into their application. The Joysticks provide information for a full 360 degree range of motion, but also enable more classic 4 or 8-axis systems.

Joysticks can be either fixed (i.e. always appear in one place) or non-fixed (will re-position on touch.) Touch areas can be assigned to each control as well, as a means to set regions for non-fixed Joysticks, but can also be used to increase the region a fixed control will be triggered in.

Default visual effects have been added to both Joysticks and Buttons and can be adjusted by configuring the minimum and maximum opacities of each individual control.

Joystick Example

The following creates a fixed Joystick that is positioned on the bottom-right corner of the canvas. The control is always visible, but will only fade to full opacity when touched.

```
this.controls.addJoystick({
  imageBase: './Resources/controls_overlay/dpad.png',
  imagePad: './Resources/controls_overlay/pad.png',
  fixed: true,
  pos: new cc.Point(ccCanvas.width - 100.0, 100.0),
  opacLow: 80.0
});
```

A side-by-side comparison of this Joystick not being interacted with, and being interacted with, can be seen below.



Required Arguments

- imageBase: A **string** path to the image file (PNG, IPG, BMP) that you want to use for the base of the Joystick.
- imagePad: A **string** path to the image file (PNG, JPG, BMP) that you want to use for the Joystick pad. The pad will follow the user's finger as they move it around the Joystick.
- fixed: A **boolean** variable indicating whether the control should remain in place, or reposition itself to the touch point, on a **touchstart**.

Optional Arguments

- pos: A **cc.Point** that denotes the position of the control. Primarily for use with fixed controls, or non-fixed controls that start visible (i.e. non-zero opacLow.) This value defaults to **new cc.Point(0, 0)**. The position is relative to the center of the Joystick.
- trigger: A **cc.Rect** that identifies the screen region where a **touchstart** event will be triggered for this control. By default, this value is set to the dimensions of the actual control via **cc.Sprite.getBoundingBox**.
- opacLow: A **number** between 0 and 255 to denote the minimum (or fade out) opacity of this control. By default, this is set to 255.
- opacHigh: A **number** between 0 and 255 to denote the maximum (or fade in) opacity of this control. By default, this is set to 255.

Button Example

The following creates a Button that is positioned in the top-right corner of the screen. The control is always visible but semi-transparent when not being interacted with.

```
this.controls.addButton({
  image: './Resources/controls_overlay/buttonred.png',
  pos: new cc.Point(ccCanvas.width - 54.0, ccCanvas.height - 54.0),
  opacLow: 80.0
});
```

A side-by-side comparison of this Button not being interacted with, and being interacted with, can be seen below.



Required Arguments

- image: A **string** path to the image (PNG, JPG, BMP) file that you want to use for the Button.
- pos: A **cc.Point** that denotes the position of the control. This value defaults to **new cc.Point(0, 0)**.

Optional Arguments

- trigger: A **cc.Rect** that identifies the screen region where a **touchstart** event will be triggered for this control. By default, this value is set to the dimensions of the actual control via **cc.Sprite.getBoundingBox**.
- opacLow: A **number** between 0 and 255 to denote the minimum (or fade out) opacity of this control. By default, this is set to 255.

- `opacity`: A **number** between 0 and 255 to denote the maximum (or fade in) opacity of this control. By default, this is set to 255.

ControlsOverlay.js

The full source for this framework and working sample can be found attached.

As mentioned, the intent is to provide Cocos2d-HTML5 developers with as simple of a solution as possible to integrate with their already existing applications.

As such, let's look at a sample Cocos2d-HTML5 Layer that would leverage this framework. Here, we will create two Joystick controls and one Button control. The first will be triggered anywhere on the left-hand side of the screen and will move the Button around the screen. The second Joystick will change the color of the Button depending on the direction the Joystick is moved (north = green, east = blue, south = yellow, and west = red.) Finally, when the Button is clicked, it will change the **cc.LayerColor** background to the colour of the Button.

First, we'll set a few global definitions and then extend **cc.LayerColor** as our primary layer.

```
/*global window, document, console, cc, ccCanvas, ControlsOverlay */
var CocosApp = cc.LayerColor.extend({
```

Next, we'll create a placeholder variable for the ControlsOverlay object we will instantiate.

```
controls: null, /* Will be used to instantiate a ControlsOverlay Node. */
```

Following this, we'll define our **init** function for our layer; we will subsequently instantiate and integrate our ControlsOverlay object here.

```
init: function () {
    'use strict';
    var layer, movement, button, colour;
```

Next, we'll initialize our layer with a gray background and keep a reference via **layer**.

```
/* Our Layer. */
this._super();
this.initWithColor(new cc.Color4B(50, 50, 50, 255));
layer = this;
```

Now we get to the ControlsOverlay object. First thing we need to do is instantiate our **controls** object.

```
/* Create a new ControlsOverlay object. */
this.controls = new ControlsOverlay({
    canvas: ccCanvas
});
```

You can see here that we are passing **ccCanvas**. The **<canvas>** element that we pass in will be the element that gets the touch events assigned to it. In most cases, this will be your primary **<canvas>** element that you define for Cocos2d-HTML5 in your **index.html** file.

Now that we have a **controls** object, let's add the **movement** Joystick.

```
/* Add a Joystick. We will use this to move the button around the screen. */
movement = this.controls.addJoystick({
  imageBase: './Resources/controls_overlay/dpad.png',
  imagePad: './Resources/controls_overlay/pad.png',
  fixed: false,
  trigger: new cc.Rect(0, 0, ccCanvas.width / 2.0, ccCanvas.height),
  opaLow: 0.0
});
```

You can see that all we're really doing is making a call to **ControlsOverlay.addJoystick** and supplying various arguments. This call returns an instance to the Joystick which we will use to implement the movement functionality shortly.

In the sample code above, we are creating a Joystick that will position itself at the user's touch point (**fixed: false**), and will fade in/out between 0 (none) and 255 (full) opacity. Since the Joystick can freely float, we've also increased the **trigger** rectangle to be anywhere on the left-side of the screen.

Next, we'll add the **colour** Joystick.

```
/* Add a Joystick. We will use this to change the colour of the button. */
colour = this.controls.addJoystick({
  imageBase: './Resources/controls_overlay/dpad.png',
  imagePad: './Resources/controls_overlay/pad.png',
  fixed: true,
  pos: new cc.Point(ccCanvas.width - 100.0, 100.0),
  opaLow: 80.0
});
```

Unlike the **movement** Joystick, the **colour** Joystick will not move (**fixed: true**) and will always be visible with a minimum opacity of **80.0**. Since we have not set a **trigger** rectangle, it will default to the dimensions of the control itself.

Finally, we add a Button with the same approach.

```
/* Add a Button. We add it last so that it is rendered on top of other controls. */
button = this.controls.addButton({
  image: './Resources/controls_overlay/buttonred.png',
  pos: new cc.Point(ccCanvas.width - 54.0, ccCanvas.height - 54.0),
  opaLow: 80.0
});
```

Similar to above, we are now calling **ControlsOverlay.addButton** and supplying our arguments. We're capturing the returned object in **button** which will be leveraged for additional functionality. First, we implement our **button** movement.

```

    /* Update the Button's position based on the movement Joystick. We'll schedule an update every
    frame. */
    button.update = function (dt) {
        var position;

        /* Calculate the time since last frame to normalize movement speed (at 60 fps we expect 0.017
        seconds per frame), and multiply by an arbitrary scaling factor. */
        dt = dt / 0.017 * 0.15;

        /* Get the current position of the Button. */
        position = this.getPosition();

        /* Update our position based on the velocity of the movement Joystick. We'll keep a 54.0 pixel
        border that the button can not cross. */
        position.x = Math.max(54.0, Math.min(position.x + movement.velocity.x * dt, ccCanvas.width -
        54.0));
        position.y = Math.max(54.0, Math.min(position.y + movement.velocity.y * dt, ccCanvas.height -
        54.0));

        /* Update our Button's position and trigger area. */
        button.setPosition(position);
        button.trigger = button.getBoundingBox();
    };

    /* Start updating our Button's position every frame. */
    button.scheduleUpdate();

```

Once we schedule the **update** function, it will be called every frame. Each frame we will calculate the time that has passed and leverage **movement.velocity** to get how much the Joystick is being pushed in the **x** and **y** directions. Based on these values, we update the position of the Button.

Next, we implement the functionality for when the Button is tapped.

```

    /* When the Button is tapped, we'll change the background colour. */
    button.color = new cc.Color4B(120, 0, 0, 255);
    button.onTouchEnd = function (touch, point) {
        if (cc.Rect.CCRectContainsPoint(this.trigger, point)) {
            layer.setColor(this.color);
        }
    };

```

We default a custom **color** property of the **button** to red to match the default Sprite image. When the touch ends, we ensure that the touch point is inside the actual control (i.e. the user hasn't moved their finger outside of the control after initiating a touch.) If we have a valid tap, we update the colour of our **layer**.

Next we'll implement the **colour** Joystick that actually updates the **button's** image and **color** property.

```

        /* Our colour Joystick will update the image used for the Button. To minimize work, we'll keep
        track of the previous Joystick direction. */
        colour.prevDirection = -1;
        colour.onTouchMove = function () {
            /* If this Joystick's direction has changed since our previous direction... */
            if (this.prevDirection !== this.direction) {
                /* ...update to the new direction. */
                this.prevDirection = this.direction;

                /* We are ignoring diagonals, and only updating with a 4-axis system. We could easily add
                the omitted directions though. */
                if (this.direction === 0) {
                    /* When this Joystick is positioned West, we will have a red Button. */
                    button.setTexture(cc.TextureCache.getInstance().textureForKey('./Resources/controls_overlay/buttonred.png'
                    ));
                    button.color = new cc.Color4B(120, 0, 0, 255);
                } else if (this.direction === 2) {
                    /* When this Joystick is positioned North, we will have a green Button. */
                    button.setTexture(cc.TextureCache.getInstance().textureForKey('./Resources/controls_overlay/buttongreen.pn
                    g'));
                    button.color = new cc.Color4B(0, 120, 0, 255);
                } else if (this.direction === 4) {
                    /* When this Joystick is positioned East, we will have a blue Button. */
                    button.setTexture(cc.TextureCache.getInstance().textureForKey('./Resources/controls_overlay/buttonblue.png
                    '));
                    button.color = new cc.Color4B(0, 0, 120, 255);
                } else if (this.direction === 6) {
                    /* When this Joystick is positioned South, we will have a yellow Button. */
                    button.setTexture(cc.TextureCache.getInstance().textureForKey('./Resources/controls_overlay/buttoneyellow.p
                    ng'));
                    button.color = new cc.Color4B(120, 120, 0, 255);
                }
            }
        };

```

First, we add a custom **prevDirection** property to our **colour** Joystick indicating that we have no previous direction recorded. We then implement a function to be triggered anytime the touch point associated with this Joystick moves.

To minimize any performance impacts, we first check to ensure that the direction of this Joystick is different from the direction in the previous update. We don't perform any actions if the direction hasn't changed.

From there, all we're doing is checking the **direction** of the Joystick (0 = west, 2 = north, 4 = east, and 6 = west.) Based on this **direction** we update the image of the **button** and also its **color**; this **color** is what we referenced in our **button.onTouchEnd** implementation.

To finish off our LayerColor implementation we add our ControlsOverlay object, containing all of our controls, to our layer.

```

        /* Add our ControlsOverlay Node to our LayerColor. */
        this.addChild(this.controls);

        return true;
    }
};

```