|  |
| --- |
| HTML5 Elements Cookbook |
| H5E Experiment: Canvas Sprites |
| DRAFT  Published 15 July, 2011 |
|  |

|  |
| --- |
|  |
| H5E Scout Team  Windows Web Partners  Microsoft Corporation  Microsoft Confidential |

# Summary

HTML5 and related technologies provide key tools that supplement, or even circumvent, component-based solutions in bringing interactive experiences to both mobile and desktop devices. When used skillfully, the Canvas element offers some significant improvements in the immersive/interactive space.

**

## Practical uses for partners

HTML5’s Canvas element provides opportunities for partners to explore the possibilities of alternatives to component-based multimedia experiences. Games like this demo are just one example. Interactive and animated advertising are also logical uses of Canvas.

## Scope

This document describes an experiment conducted by the H5E scout team using Clarity Consulting. Our objective is to test the limits of HTML5 solving real-world partner questions. This document assumes an existing knowledge of JavaScript and jQuery. This document does not supersede any requirements or instructions provided by the IE team.

## Keywords

HTML5, Canvas, JavaScript, Sprites, IE9

## Contact us

To contact us for questions or support, please email Chewy Chong ([ChewyC](mailto:ChewyC?subject=HTML5%20Cookbooks)). Feedback is welcome.

Contents

Summary 2

Practical uses for partners 2

Scope 2

Keywords 2

Contact us 2

Canvas Sprites Demo Overview 4

A quick overview of sprite sheets 4

Touch enabling through HTML5 4

Making common elements aspirational 4

Experience Walkthrough 5

How do I build this using HTML5? 7

Primary files in this solution 7

Combining sprite sheets with the Canvas element 8

Conclusions and Recommendations 17

Notes about Sprite.js 17

Resources 18

Relevant Web sites and specifications 18

Microsoft Resources 18

Appendix A: About H5E 19

What is an HTML5 Elements Cookbook? 19

Contact us 19

Copyright and trademark information 19

Appendix B: Sprite.js 20

Document Revision History 27

# Canvas Sprites Demo Overview

With the advances of HTML5 and JavaScript, there is a lot of conversation surrounding the practicality of switching from a component-based model for rich site elements to an HTML5-based approach. There are few more heated debates about this than in game development and interactive circles.

In this experiment, we are taking a look at the viability of using the HTML Canvas tag in combination with a “traditional” digital animation technique: sprite sheets, to determine if Canvas is a practical alternative to a component-based solution.

## A quick overview of sprite sheets

Sprite sheets are one of the fundamental tools for building games and animations, and have been central to game development since the 70’s. Rather than loading separate image files for each state change of a visual element, it is generally far more efficient to load a single file that contains multiple states of the image, and then programmatically change the portion that is displayed when the state changes.

A simple example is the classic rollover effect on menu buttons. Three common approaches to button rollovers are swapping multiple image files with JavaScript (outdated but still pervasive), using CSS to change the style dynamically, or using CSS sprites to change the portion of a single preloaded image that is displayed.

In this case, we are testing whether it is practical to perform sprite manipulation through JavaScript and jQuery, and then painting the results onto the HTML5 Canvas element.

## Touch enabling through HTML5

Part of the challenge that was outlined for this experiment was to explore some of the basics of detecting and supporting touch-enabled devices. This includes orientation change and basic touch events.

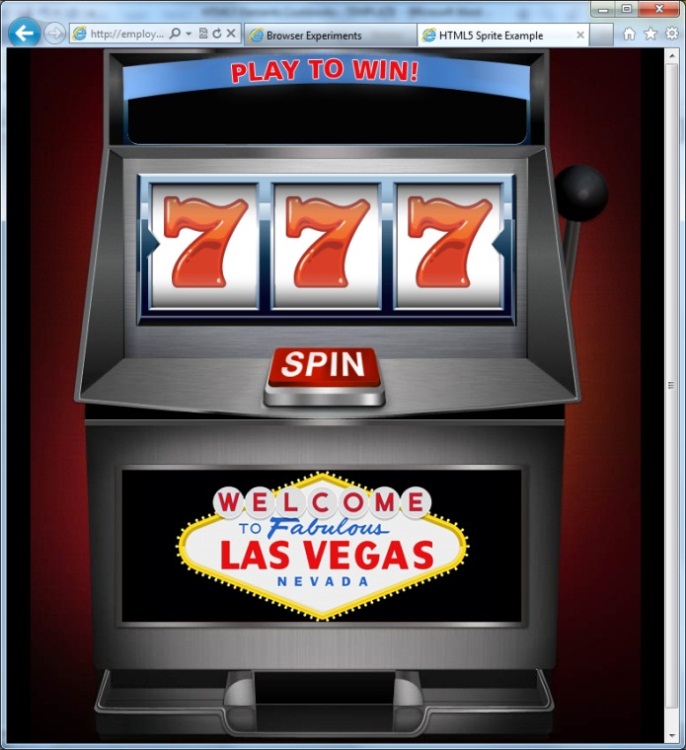
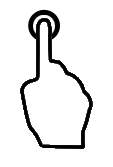
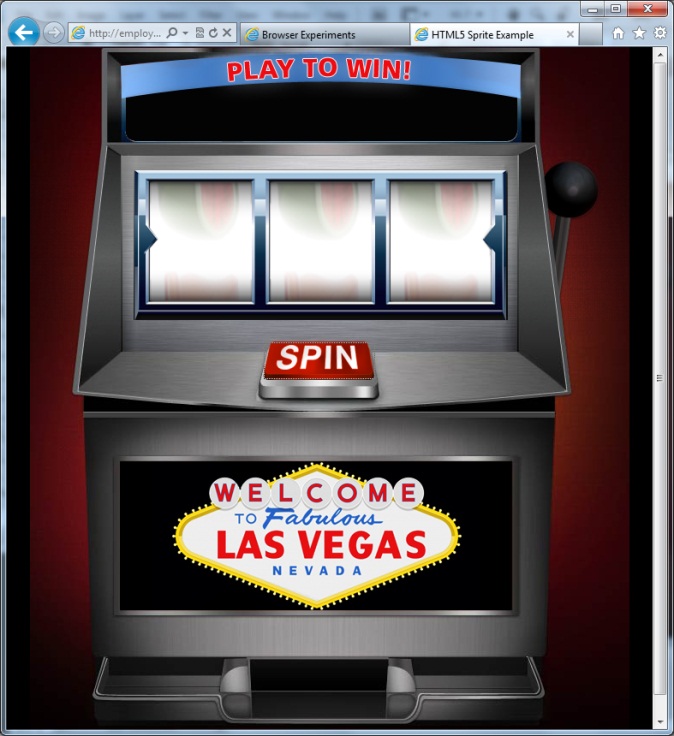
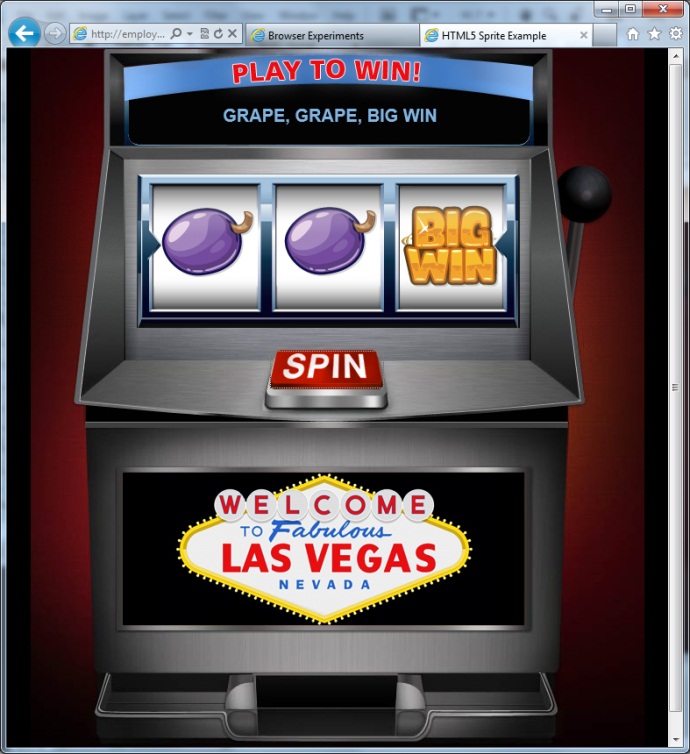
The touch events in this experiment are limited to touch that works in the same manner as a basic click. For a more detailed example of supporting touch events, see *HTML5 Elements Cookbook: DOM Scroller*.

## Making common elements aspirational

This demo is a prime example of the necessity to insert small details to achieve larger immersive impact. Aspirational Experiences are often equal parts design and development, and this sample is no exception. Look for the details that the developer has inserted to make the experience more aspirational without incurring performance tradeoffs. Elements like the blur illusion, which could be accomplished in a number of ways, are particularly interesting in terms of increasing the suspension of disbelief without using an expensive technique performance-wise.

## Experience Walkthrough

The Canvas Sprites demo uses a slot machine, a classic game element, to experiment with some of the advantages and limitations animating sprites and then displaying them on the Canvas element. The demo application includes the following behavior:

1. The user navigates to the web page, and the initial content is loaded.   
   
2. ****The user clicks (or touches) **Spin**, and the slot machine reels appear to spin. Note that this also applies to a touch interface.  
   
3. When the spin completes, the symbols on the reels and a message above the reels describing the result is displayed.   
   

# How do I build this using HTML5?

Until recently, building this type of interactive animation typically fell into the realm of a component like Flash or Silverlight. For this experiment, we are using a combination of the HTML5 Canvas element, jQuery, and traditional sprite sheet animation to challenge that notion, and determine if HTML5 is a viable alternative for lightweight web-based games and interactive media.

***NOTE****: Many of the HTML5 Experiments are still under development. Our initial target is to build prototypes that work on current HTML5-supported browsers and tablet devices, including the iPad. The experiments do not aim for full cross-browser support at this stage, but we will likely build in graceful degradation in future updates.*

To ensure that users have a similar cross-browser experience, the following table describes the compatibility of the solutions in this document:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| HTML5 Logo**HTML5 Feature** | IE6.0 | IE7.0 | IE8.0 | IE9.0 | IEPP | Chrome11 | | Chrome12 | Safari4.x | Safari5.x | Firefox3.6 | Firefox4.x | Firefox5.x | Opera11 |
| Canvas |  |  |  |  |  |  | |  |  |  |  |  |  |  |
|  | | | | | | | Full Support | | | | | | |  |
|  | | | | | | | Supported with Shim | | | | | | |  |
|  | | | | | | | No current support | | | | | | |  |

***NOTE****: In general, shims are not incorporated into the HTML5 Experiments at this stage. If a shim or polyfill is required for cross-browser support, see* [*http://browserexperiments.com*](http://browserexperiments.com) *for details on shim implementation.*

## Primary files in this solution

|  |  |
| --- | --- |
| **File name** | **Description** |
| index.html | Web page for the sample |
| results.xml | Data file that contains the list of possible results |
| common.js | Primary game logic |
| sprite.js | Sprite class |

### Source Location

<https://github.com/molant/BrowserExperiments/tree/master/cookbook/1_Sprites>

### Sample Location

<http://employees.claritycon.com/eklimczak/html5Cookbook/1_Sprites/index.html>

## Combining sprite sheets with the Canvas element

Part of the draw of an application such as this is the small, nearly intangible details that the developer builds into the experience. Prior to HTML5, these details via JavaScript could become too expensive from a performance standpoint, forcing tradeoffs that diminished the aspirational features. By using Canvas, we are able to maintain the majority of these aspirational nuances that enrich the experience. Adding details like the small bounce effect that occurs when the slot machine reels stop spinning is a small example of a procedural detail that might have been omitted in the past for the sake of performance on the web.

This example uses a combination of sprite sheet manipulation, jQuery, and the HTML5 Canvas element. The demo is optimized to perform well in current browsers, including Safari on iPad.

### Initializing the page elements

The first task during initialization is to determine if the HTML5 Canvas element is supported. This approach is similar to the Modernizr approach to Canvas detection.

//determine if this browser supports the HTML5 canvas

this.detectSupportsCanvas = function() {

return !!document.createElement('canvas').getContext;

}

If Canvas is supported, the corresponding elements and attributes are detected. Other pieces are initialized, including creating an array to contain sprite data. The XML data that describes the result of pressing the spin button is also acquired through jQuery and ajax. Note the cached jQuery object to improve performance.

//canvas method

if (\_this.supportsCanvas) {

//get a handle to page elements

\_this.canvasElement = $('#canvas');

//measure content size

\_this.canvasWidth = \_this.canvasElement.attr('width');

\_this.canvasHeight = \_this.canvasElement.attr('height');

//get a handle to the canvas elements

\_this.canvasElementContext = \_this.canvasElement.get(0).getContext('2d');

}

//create array to hold sprites

if (\_this.spriteQueue) \_this.spriteQueue.removeAll();

\_this.spriteQueue = new SpriteQueue();

\_this.resultsHTML = $("#results");

\_this.loadXML();

\_this.createSprites();

\_this.paintStart();

}

this.loadXML = function() {

$.ajax({

type: "GET",

url: "xml/results.xml",

dataType: "xml",

success: function(data){

resultXML = $(data); //cached jquery object to minimize jquery lookups later

}

}); //close ajax

### Creating the sprites

To create the three reels, two sets of sprite images are used. The first set includes all of the images that are displayed when the reels are not spinning. The second set is a set of blurred sprites that are displayed when **spin()** is invoked.

One of the simple optimizations employed is to use the same two sprite sheets for every reel, saving load time. **createSprites()** creates a reel image and blur image for each reel, assigns the sprite to the **canvasElement** member, sets the positioning, and sets the size. This process is repeated for each reel.

this.createSprites = function() {

//Create slot image

\_this.SlotImg = new Image();

\_this.SlotImg.src = 'images/slots.jpg';

//Create Blur Image

\_this.SlotBlurImg = new Image();

\_this.SlotBlurImg.src = 'images/blur.jpg';

//Create sprite for each slot

\_this.SlotSprite = new Sprite({

id: 'slot1',

container: \_this.canvasElement,

image: \_this.SlotImg,

alpha: 1,

x: 10,

y: 0,

rotate: 0,

zIndex: 1,

width: 120,

height: 2100

});

\_this.SlotSprite2 = new Sprite({

id: 'slot2',

container: \_this.canvasElement,

image: \_this.SlotImg,

alpha: 1,

x: 150,

y: 0,

rotate: 0,

zIndex: 1,

width: 120,

height: 2100

});

\_this.SlotSprite3 = new Sprite({

id: 'slot3',

container: \_this.canvasElement,

image: \_this.SlotImg,

alpha: 1,

x: 290,

y: 0,

rotate: 0,

zIndex: 1,

width: 120,

height: 2100

});

\_this.spriteQueue.add(\_this.SlotSprite);

\_this.spriteQueue.add(\_this.SlotSprite2);

\_this.spriteQueue.add(\_this.SlotSprite3);

//Create Blur for Each Slot

\_this.BlurSprite = new Sprite({

id: 'blur1',

container: \_this.canvasElement,

imageStrip: \_this.SlotBlurImg,

imageStripColCount: 2,

imageStripRowCount: 6,

alpha: 1,

x: 0,

y: 5,

rotate: 0,

zIndex: 1,

width: 140,

height: 140

});

\_this.BlurSprite2 = new Sprite({

id: 'blur2',

container: \_this.canvasElement,

imageStrip: \_this.SlotBlurImg,

imageStripColCount: 2,

imageStripRowCount: 6,

alpha: 1,

x: 70,

y: 5,

rotate: 0,

zIndex: 1,

width: 140,

height: 140

});

\_this.BlurSprite3 = new Sprite({

id: 'blur3',

container: \_this.canvasElement,

imageStrip: \_this.SlotBlurImg,

imageStripColCount: 2,

imageStripRowCount: 6,

alpha: 1,

x: 140,

y: 5,

rotate: 0,

zIndex: 1,

width: 140,

height: 140

});

}

### Spinning the reels

The **Spin** button initiates the spinning animation by first verifying that the reels are not spinning, and then calling **showBlur()**.

this.spin = function(){

if(!\_this.isSpinning)

{

\_this.showBlur(); //Show Blur

**showBlur()**sets several properties to track state, and then uses **setTimeout()** for each reel to create make the reels stagger the stop time This creates the illusion of each reel stopping one after the other from left to right. Within the call to **setTimeout()**,**snap()**is also called, which uses jQuery to simulate a bounce effect when the reel comes to a stop**.**

this.showBlur = function(){

\_this.isSpinning = true;

\_this.spin1 = true;

\_this.spin2 = true;

\_this.spin3 = true;

setTimeout(function(){ \_this.spin1 = false; \_this.snap(\_this.SlotSprite); }, 2000);

setTimeout(function(){ \_this.spin2 = false; \_this.snap(\_this.SlotSprite2); }, 2700);

setTimeout(function(){ \_this.spin3 = false; \_this.snap(\_this.SlotSprite3); \_this.isSpinning = false;}, 3200);

}

### Creating a bounce effect when the reel stops spinning

When each reel finishes the time interval that is specified in **setTimeout()**,**snap()**is invoked, passing in the slot parameter that indicates which reel is being stopped. **snap()** uses jQuery to ease the stopping motion and bounce back slightly to create a bounce effect.

this.snap = function(slot){

var yPos = slot.y;

$(slot).animate({ y: yPos-30 }, 0 )

.animate( { y: yPos } , 250, 'easeOutBack' );

### Painting to the Canvas element

The following prepares the Canvas element by clearing the contents, sets the time interval for painting to the Canvas element in milliseconds, and paints the current instance of the **spriteQueue** object. Each **spriteQueue** is created in sprite.js.

***Note****: The sprite class (or more accurately, prototype) that is contained in sprite.js, is a complex set of functions that perform the tasks necessary to paint to the Canvas element. For practical purposes, most developers would reuse a library like this to access the Canvas properties, so this document does not describe sprite.js in detail. If you are interested in deriving your own prototype from this class, see the source in the appendix.*

this.paintStart = function() {

\_this.paintInterval = setInterval(

function(){

try {

\_this.paint();

}

//stop the painting refresh if an error occurs

catch(e) {

\_this.paintStop();

throw(e);

}

}

, \_this.PAINT\_INTERVAL

);

}

this.paintStop = function() {

window.clearTimeout(\_this.paintInterval);

}

this.clear = function() {

if (\_this.supportsCanvas) {

\_this.canvasElementContext.clearRect(0, 0, \_this.canvasWidth, \_this.canvasWidth);

}

}

this.paint = function() {

//clear the canvas

\_this.clear();

\_this.spriteQueue.paint();

if(\_this.spin1) \_this.BlurSprite.paint();

if(\_this.spin2) \_this.BlurSprite2.paint();

if(\_this.spin3) \_this.BlurSprite3.paint();

}

}

### Choosing the result

When the **Spin** button is clicked, the resulting images that are displayed are determined by a call to **randomFromTo()**, which generates an ID value that is used to select which text and which sprites are displayed. **randomFromTo()** uses JavaScript’s provided **Math** object to determine a rounded, random number between 1 and the maximum id value listed in the XML file (in this case, 50), and returns the value.

The element that contains any existing text is cleared, and **randomFromTo()** is invoked.

//clear old result text

$('#results').empty();

//get random number 1-total number of results

var numResults = resultXML.find('result').length;

var r=randomFromTo(1,numResults);

[…]

function randomFromTo(from, to){

return Math.floor(Math.random() \* (to - from + 1) + from);

}

Once the random ID value is returned, the XML is searched for the node that has an ID that matches the number that was generated. Each XML node contains the text that is displayed for each result and the sprite coordinates (in this case, the y coordinate) for each of the three reels that are displayed in the Canvas element. The results string is then constructed so that it is ready to be displayed.

//Create empty result object

tempResult = new Object();

//loop through xml, find each result

resultXML.find('result').each(function(){

//store reference to avoid unncessary jquery lookups

var temp = $(this);

//if id matches random number

if(temp.attr('id') == r)

{

tempResult.disc1 = temp.find('disc01').text();

tempResult.disc2 = temp.find('disc02').text();

tempResult.disc3 = temp.find('disc03').text();

tempResult.yPos1 = temp.find('disc01').attr('y');

tempResult.yPos2 = temp.find('disc02').attr('y');

tempResult.yPos3 = temp.find('disc03').attr('y');

if (tempResult.disc2 != "") tempResult.disc1 += ",&nbsp;";

if (tempResult.disc3 != "") tempResult.disc2 += ",&nbsp;";

return true;

}

An excerpt of the XML file:

<?xml version="1.0" encoding="ISO-8859-1"?>

<results>

<result id="1">

<rowitems>

<disc01 y="0">Seven</disc01>

<disc02 y="1350">Seven</disc02>

<disc03 y="450">Big Win</disc03>

</rowitems>

</result>

<result id="2">

<rowitems>

<disc01 y="750">Lemon</disc01>

<disc02 y="1200">Bar</disc02>

<disc03 y="1200">Bar</disc03>

### </rowitems>

[…]

<result id="50">

<rowitems>

<disc01 y="300">Grape</disc01>

<disc02 y="1050">Orange</disc02>

<disc03 y="450">Big Win</disc03>

</rowitems>

</result>

</results>

### Displaying the resulting sprites and text

After the XML data has been parsed, jQuery is used to animate the images and text display. Each element in the XML contains the text that is displayed in the results pane, as well as the y coordinate that indicates the location on the sprite sheet where the corresponding sprite is located.

An interesting nuance here is the results text is delayed using the delay function, and the opacity is animated. In functional terms, there is no reason why both the text and images could not be displayed simultaneously since the data has already been acquired; however, this is a good example of subtlety that increases the quality of the experience. It feels more “natural” that the text would be displayed after the reels stop spinning.

$(\_this.SlotSprite).animate({ y: -tempResult.yPos1 }, 0 );

$(\_this.SlotSprite2).animate({ y: -tempResult.yPos2 }, 0 );

$(\_this.SlotSprite3).animate({ y: -tempResult.yPos3 }, 0 );

//display result

$('<div class="result" id="id\_'+tempResult.id+'"></div>').html('<span id="rowitems"></span>').appendTo('#results');

$('<h4>').html(tempResult.disc1).appendTo('#rowitems');

$('<h4>').html(tempResult.disc2).appendTo('#rowitems');

$('<h4>').html(tempResult.disc3).appendTo('#rowitems');

\_this.resultsHTML.animate({opacity: 0}, 0);

\_this.resultsHTML.delay(3300).animate({opacity: .95}, 300);

### Determining viewport orientation

To determine whether the orientation of the viewing device is portrait or landscape, the sample lets jQuery initially detect the orientation position, and then a bit of JavaScript updates the orientation settings as appropriate.

To determine if the orientation changes, the **window.onorientationchange** event is used to call the **updateOrientation()** function.

function updateOrientation() {

/\*indicates whether the screen is turned to the left, or turned to the right. \*/

var orientation = window.orientation;

switch (orientation) {

case 90: window.scrollTo(20, 35);

case -90: window.scrollTo(20, 35);

}

}

window.onorientationchange = updateOrientation;

### Determining touch support

The primary HTML page uses JavaScript through jQuery to determine whether a touch-supported device is used. After the document is loaded, the **hasTouchSupport()** function is called.

<script type="text/javascript">

var controller;

$(document).ready(function () {

controller = new Controller();

controller.load();

var useTouch = hasTouchSupport();

Based on the result that is assigned to the **useTouch** variable, a simple decision point is implemented.

if (!useTouch) {

$('#spin-button').mousedown(function () {

controller.spin()

$(this).css({ backgroundPosition: "bottom", border: "none" });

$('#clickme').remove()

}).mouseover(function () {

$(this).css({ backgroundPosition: "center", border: "none" });

}).mouseup(function () {

$(this).css({ backgroundPosition: "top", border: "none" });

});

}

else {

$('#spin-button').bind('touchstart', function () {

controller.spin()

$(this).css({ backgroundPosition: "bottom", border: "none" });

$('#clickme').remove()

}).bind('touchend', function () {

$(this).css({ backgroundPosition: "top", border: "none" });

controller.spin()

}).mousedown = null;

}

# Conclusions and Recommendations

A few notes on this solution:

* **The Canvas drawing API is very low-level and difficult to manage**. Managing X,Y, Scale, and rotate properties of many elements drawn on a Canvas element is cumbersome. Using a sprite class like the one that is provided in this sample gives developers a more familiar dev experience like that found in Flash or Silverlight.
* **Using a sprite class resulted in significant performance improvements over strictly DOM/Canvas manipulation.** Sifting through images in a sprite sheet is a time tested technique for squeezing performance out of hardware, and the same holds true in this case. This also proved true with the game logic (determining which elements are displayed, determining the text to display, etc.).   
  An early approach was to try to do all of the manipulation and logic within the Canvas element (changing the images and detecting them to determine which was being displayed, and so forth), but this proved to be harder to manage and did not perform well.
* **Using transforms on a sprite sheet as a background image in a div was not as effective.** One earlier approach was to lay out the sprite sheet inside a div as a background image, and then shift the visible portion of the div dynamically. This approach will work, but is not as smooth as the result in this demo.
* **Sound can be problematic for performance in certain browsers:** If you look through the source, you will notice that there are sound elements that have been commented out. In the initial versions of this solution, sound was enabled. However, performance became unpredictable particularly in Safari on mobile, so for this version sound support is not enabled. Workarounds for this may be explored in future versions.

One of the critical pieces of making this experience feel realistic was the illusion of blurring the reels when they spin. There are several ways to create a similar effect, but using a sprite sheet with blurred images was the most effective, and created an immersive result.

## Notes about Sprite.js

Sprite.js represents a body of work that helps developers to access common functionality to draw to the Canvas element. In environments and frameworks like XNA and Flash where images are used for gaming or interactive, many common sprite, frame, and general object management elements and objects are provided. In the case of Canvas, it is a raw surface that provides very little in terms of built-in support for managing such elements. Libraries such as this help developers that are used to writing visual elements to the screen using frameworks that are built for this purpose.

Most developers will likely use EaselJS (<http://easeljs.com/>) or a similar to take advantage of the abstracted tools provided that help with Canvas functions.

# Resources

## Relevant Web sites and specifications

|  |  |
| --- | --- |
| jQuery reference | <http://docs.jquery.com/Main_Page> |
| EaselJS | <http://easeljs.com/> |

## Microsoft Resources

|  |  |
| --- | --- |
| H5E Primary Contact | Chewy Chong ([ChewyC](mailto:%20chewyc?subject=HTML5%20Experiments%20Cookbooks)) |
| H5E Development Contact | Anton Molleda Quintana ([v-anmoll](mailto:v-anmoll?subject=HTML5%20Experiments%20Cookbooks)) |

# Appendix A: About H5E

## What is an HTML5 Elements Cookbook?

Each HTML5 Elements Cookbook reflects a case study of an aspirational experience that is provided by a native or component-based application. The HTML5 Experiments that are conducted by the H5E team use HTML5 and related technologies to replicate these experiences. Our primary objective is to learn from these experiments to determine if an HTML5 alternative to component-based or native implementations is both possible, and practical. Each Cookbook provides a description of the element and technical details of the HTML5 replication of that feature. We also include recommendations on whether it makes sense to pursue this approach.

## Contact us

If you need assistance with technical solutions or have a best practice to share, please contact us by sending email to Chewy Chong ([ChewyC](mailto:chewyc?subject=H5E%20Cookbooks%20and%20Documentation)).

## Copyright and trademark information

This document is Microsoft Confidential.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

 2011 Microsoft Corporation. All rights reserved.

# Appendix B: Sprite.js

/\*

\* Sprite.js

\* Represents a sprite to be rendered on a <canvas>

\* Version: 1.0

\*/

function SpriteQueue() {

this.sprites = new Array();

this.spritesHash = new Object();

this.updated;

this.paint = function() {

//paint all sprites in the array

$.each(this.sprites, function(i, sprite) {

sprite.paint();

sprite.updated = true;

});

//remove inactive sprites from the array

for (var i=0; i < this.sprites.length; i++) {

var sprite = this.sprites[i];

if (sprite.active == false) {

this.remove(i);

i--;

}

}

this.updated = false; //repaint is finished

}

this.add = function(sprite) {

//add sprite to sprites array

this.sprites.push(sprite);

this.sprites.sort(function(a, b) { return a.zIndex - b.zIndex; });

//add sprite to sprites hash for fast lookup based on id

this.spritesHash[sprite.id] = sprite

this.updated = true; //force a repaint

}

this.remove = function(i) {

var sprite = this.sprites[i];

if (sprite.deallocateFn) sprite.deallocateFn();

this.spritesHash[sprite.id] = null;

this.sprites.splice(i, 1); //remove the element from the array

}

this.removeAll = function() {

for (var i=0; i < this.sprites.length; i++) {

this.remove(i);

};

}

//fast lookup of sprite based on Id.

this.get = function(id) {

return this.spritesHash[id];

}

this.getAll = function() {

return this.sprites;

}

//lookup of sprites based on a specific property (eg, name)

this.find = function(property, value, like) {

var matchedSprites = new Array();

$.each(this.sprites, function(i, sprite) {

var match = false;

if (like) {

if (sprite[properly].indexOf(value) >= 0) {

match = true;

}

}

else {

if (sprite[property] == value) {

match = true;

}

}

if (match) matchedSprites.push(sprite);

});

return matchedSprites;

}

this.toString = function() {

var str = 'length: ' + this.sprites.length + '\n';

$.each(this.sprites, function(i, sprite) {

str += sprite.toString() + '\n';

});

return str;

}

}

function Sprite(params) {

//parse required parameters

this.id = params.id == null ? 'unnamed' : params.id; //id of sprite - MUST be unique

this.container = params.container == null ? null : params.container; //parent container element to append to

//parse optional parameters

this.name = params.name == null ? null : params.name; //name of sprite - does not need to be unique (could identify a group)

this.x = params.x == null ? 0 : params.x; //x coordinate

this.y = params.y == null ? 0 : params.y; //y coordinate

this.zIndex = params.zIndex == null ? 0 : params.zIndex; //z-index to determine layer

this.text = params.text == null ? null : params.text; //text to render

this.align = params.align == null ? null : params.align; //text alignment

this.font = params.font == null ? null: params.font; //font for text

this.color = params.color == null ? null : params.color; //color for text

this.dropShadow = params.dropShadow == null ? null : params.dropShadow; //if true, a drop shadow will be created

this.alpha = params.alpha == null ? 1 : params.alpha; //opacity value

this.imageStrip = params.imageStrip == null ? null : params.imageStrip; //Sprite sheet

this.imageStripColCount = params.imageStripColCount == null ? null : params.imageStripColCount; //Num of Cols in sprite sheet

this.imageStripRowCount = params.imageStripRowCount == null ? null : params.imageStripRowCount; //Nul of Rows in sprite sheet

this.image = params.image == null ? null : params.image; //Regular single image

this.scaleX = params.scaleX == null ? 1 : params.scaleX; //scale amount (height)

this.scaleY = params.scaleY == null ? 1 : params.scaleY; //scale amount (width)

this.rotate = params.rotate == null ? 0 : params.rotate; //rotate amount (in degrees)

this.width = params.width == null ? 0 : params.width; //width (for collision detection purposes only)

this.height = params.height == null ? 0 : params.height; //height (for collision detection purposes only)

this.deltaX = params.deltaX == null ? 0 : params.deltaX; //x increases at a constant velocity

this.deltaY = params.deltaY == null ? 0 : params.deltaY; //y increases at a constant velocity

this.deltaDelay = params.deltaDelay == null ? 0 : params.deltaDelay; //delay before x/y position is changed

this.composite = params.composite == null ? null : params.composite; //composite effect

this.evalFn = params.evalFn == null ? null : params.evalFn; //function to evaluate on every frame

this.deallocateFn = params.deallocateFn == null ? null : params.deallocateFn; //function to evaluate once sprite becomes in

this.onclick = params.onclick == null ? null : params.onclick; //click event handler

//variables

this.active = true;

this.currentCellX = 0;

this.currentCellY = 0;

this.context = this.container.get(0).getContext('2d'); //get a handle to the canvas context

this.customAnimate = true;

this.updated = false;

this.set = function(property, value) {

this[property] = value;

controller.spriteQueue.updated = true;

this.updated = true;

}

this.paint = function() {

if (this.active) {

//increment position

this.x += this.deltaX;

this.y += this.deltaY;

//make changes to canvas settings

this.context.save();

if (this.color != null) this.context.fillStyle = this.color;

this.context.globalAlpha = this.alpha;

var translateX = this.x;

var translateY = this.y;

if (this.align != null) this.context.textAlign = this.align;

this.context.translate(translateX, translateY);

if (this.dropShadow) {

this.context.shadowOffsetX = 2;

this.context.shadowOffsetY = 2;

this.context.shadowBlur = 5;

this.context.shadowColor = 'rgba(0, 0, 0, ' + (this.alpha \* 0.5) + ')';

}

var transformAboutCenter = (this.width != 0 && this.height != 0) ? true: false;

if (transformAboutCenter) this.context.translate(this.width \* 0.5, this.height \* 0.5);

if (this.scaleX != 1 || this.scaleY != 1) this.context.scale(this.scaleX, this.scaleY);

if (this.rotate != 0) this.context.rotate(this.rotate \* Math.PI / 180);

if (transformAboutCenter) this.context.translate( - this.width \* 0.5, -this.height \* 0.5);

if (this.composite != null) this.context.globalCompositeOperation = this.composite;

if (this.image != null) this.paintImage(); //render the image on the canvas

else if (this.imageStrip != null) this.paintImageStrip(); //render Image Strip

else this.paintText(); //render text on the canvas

this.context.restore(); //restore canvas settings

}

}

this.paintImage = function () {

try {

var image = this.image;

if (image != null) this.context.drawImage(image, 0, 0, this.width, this.height);

}

catch (err) {

}

}

this.updateImageStrip = function(){

this.currentCellX++;

if(this.currentCellX >= this.imageStripColCount)

{

this.currentCellX = 0;

this.currentCellY++;

if(this.currentCellY >= this.imageStripRowCount) this.currentCellY = 0;

}

}

this.paintImageStrip = function () {

var strip = this.imageStrip;

if(strip != null)

{

this.context.drawImage(strip, this.currentCellX \* this.width,

this.currentCellY \* this.height,

this.width,

this.height,

this.x,

this.y,

this.width,

this.height);

}

this.updateImageStrip();

}

this.paintText = function() {

if (this.context.fillText) {

if (this.context.font != null) this.context.font = this.font;

this.context.textBaseline = 'top';

this.context.fillText(this.text, 0, 0);

}

}

this.click = function(e) {

if (this.onclick != null) {

var position = this.getRelativePosition(e);

if (this.isHit(position.x, position.y)) {

this.onclick();

}

}

}

this.isHit = function(x, y) {

if (x >= this.x && x <= this.x + this.width &&

y >= this.y && y <= this.y + this.height)

return true;

else return false;

}

this.getRelativePosition = function(e) {

var t = this.container.get(0);

var x = e.clientX + (window.pageXOffset||0);

var y = e.clientY + (window.pageYOffset||0);

do {

x -= t.offsetLeft+parseInt(t.style.borderLeftWidth||0),

y -= t.offsetTop+parseInt(t.style.borderTopWidth||0);

}

while (t=t.offsetParent);

return {x:x,y:y};

}

this.toString = function() {

var str = this.id + ' ' +

'(' + parseInt(this.x) + ',' + parseInt(this.y) + ') ' +

'(' + parseInt(this.deltaX) + ',' + parseInt(this.deltaY) + ') ' +

(this.active ? 'active' : 'inactive')

return str;

}

}

Sprite.computeWidth = function(o, font, container) {

var context = container.get(0).getContext('2d'); //get a handle to the canvas context

var width = 0;

if(typeof o == 'string') {

if (context.measureText != null) {

context.save();

context.font = font;

var textDim = context.measureText(o);

width = textDim.width;

context.restore();

}

}

else {

width = o.width;

}

return width;

}

Sprite.computeHeight= function(o, font, container) {

var context = container.get(0).getContext('2d'); //get a handle to the canvas context

var height = 0;

if(typeof o == 'string') {

if (context.measureText) {

context.save();

context.font = font;

var textDim = context.measureText('gM');

height = textDim.width; //browsers don't currently support height measurement

context.restore();

}

}

else {

height = o.height;

}

return height;

}

// Allow animating object properties directly.

var $\_fx\_step\_default = $.fx.step.\_default;

$.fx.step.\_default = function (fx) {

if (!fx.elem.customAnimate) return $\_fx\_step\_default(fx);

fx.elem[fx.prop] = fx.now;

fx.elem.updated = true;

controller.spriteQueue.updated = true;

};

# Document Revision History

|  |  |  |
| --- | --- | --- |
| **Reviser** | **Date** | **Revisions** |
| **v-jgeige** | 15 July 2011 | Initial draft |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |