Unknown Kadath

Understanding save() and restore() for the Canvas Context

Posted on July 10th, 2010 by James Litten

At first, I had a hard time grasping the purpose and use of the `save()` and `restore()` methods of the canvas' 2d context. It's really very simple and here are some examples that can help you understand it better.

Let's look at an official definition of `save()` and `restore()`

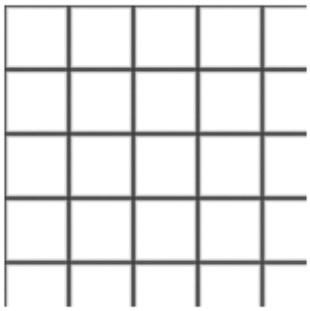
> *Each context maintains a stack of drawing states. Drawing states consist of:*
>
> ** The current transformation matrix.*
>
> ** The current clipping region.*
>
> ** The current values of the following attributes: strokeStyle, fillStyle, globalAlpha, lineWidth, lineCap, lineJoin, miterLimit, shadowOffsetX, shadowOffsetY, shadowBlur, shadowColor, globalCompositeOperation, font, textAlign, textBaseline.*
>
> *The current path and the current bitmap are not part of the drawing state. The current path is persistent, and can only be reset using the beginPath() method. The current bitmap is a property of the canvas, not the context.*
>
> *context.save() Pushes the current state onto the stack.*
>
> *context.restore() Pops the top state on the stack, restoring the context to that state.*
>
> *http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html#the-canvas-state*

Since a canvas can only have one 2d context, Save and restore can be used as a solution to a wide variety of situations. One of the most common uses is for transformations.

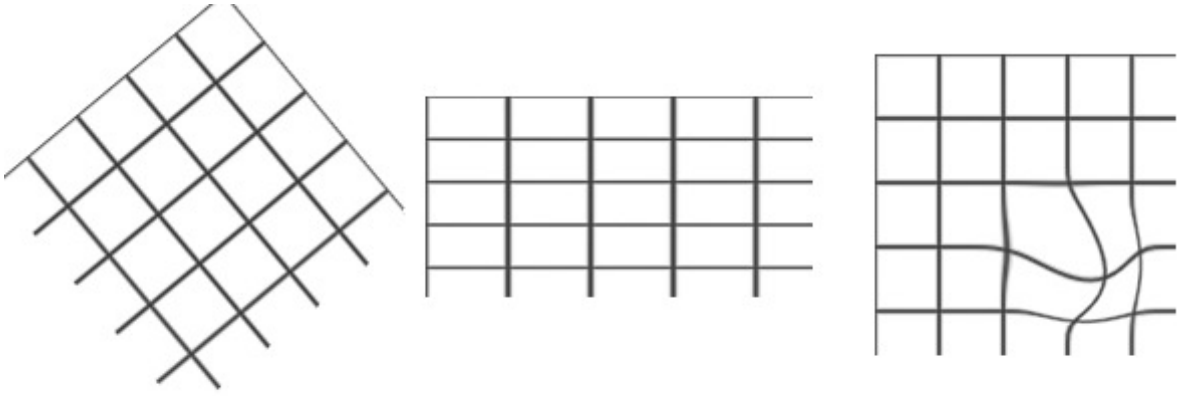
An example of how save() and restore() help with transformations.

When you perform a transformation, the entire context's coordinate system is transformed. After transforming, you often want the coordinate system to be back to normal for your next step. Reversing the transformation by using another transformation is a dicey affair and can easily introduce small errors that add up quickly. It's easier to simply save the normal starting coordinate system as a saved drawing state and then after we do our transformation and wish to have a normal coordinate system as opposed to our newly transformed one, we simply restore the state we saved before transforming. Got it? LOL. Here's some pictures to make it clearer.

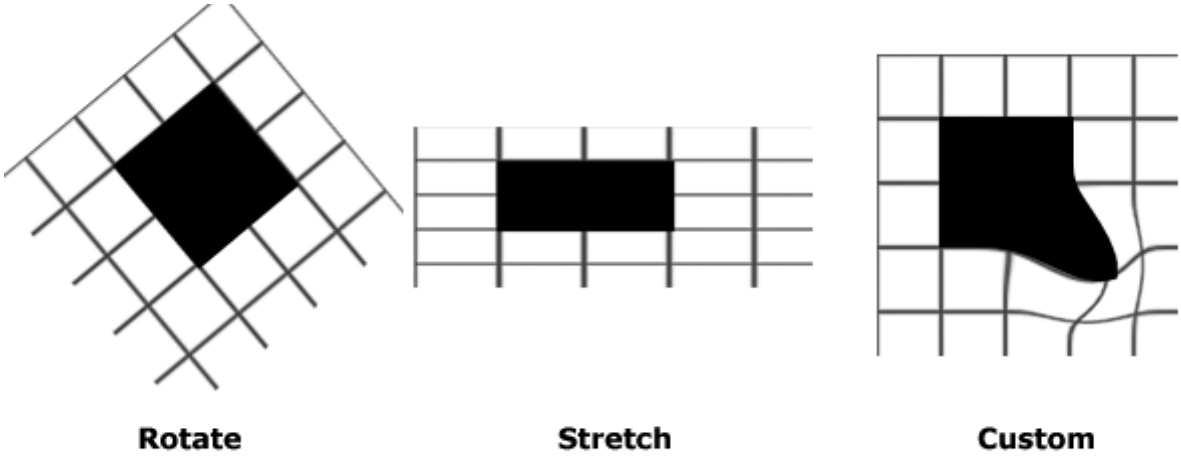
We start with our normal coordinate system and call canvascontext.save() to push a copy of our current drawing state onto our drawing state stack.



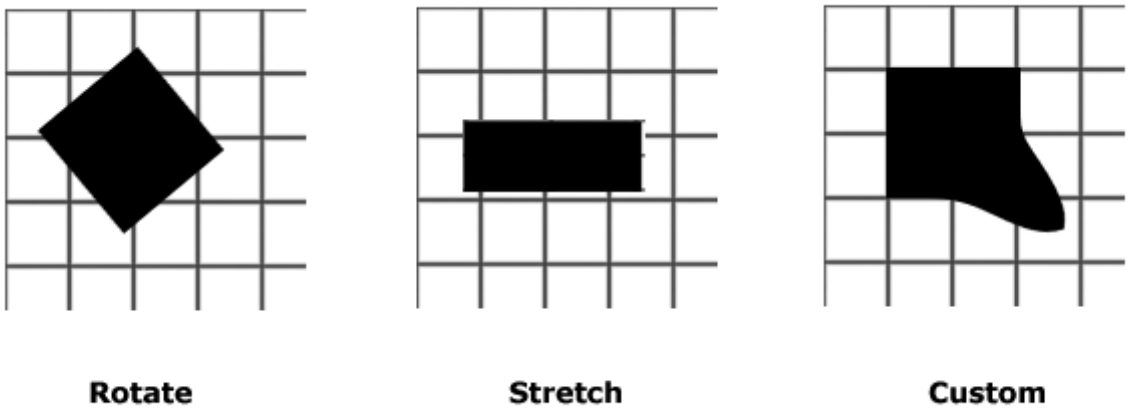
Then we transform our context.



Draw our transformed shape on it. canvascontext.fillRect(1,1,2,2)



Now we want to draw more shapes but we don't want the current transformations to apply so we call restore() and pop the last saved drawing state off of the drawing state stack.



Notice that the shapes did not change, just the drawing state as it applies to future drawing.

Super simple example

Here is a super simple example to illustrate how the drawing state stack works with save() and restore().

First, the source code listing so you can play around with it. This example is coded for readability and not for optimized operation. All you need is a text editor like notepad and an HTML5 friendly browser (I'm using Firefox 3.6).

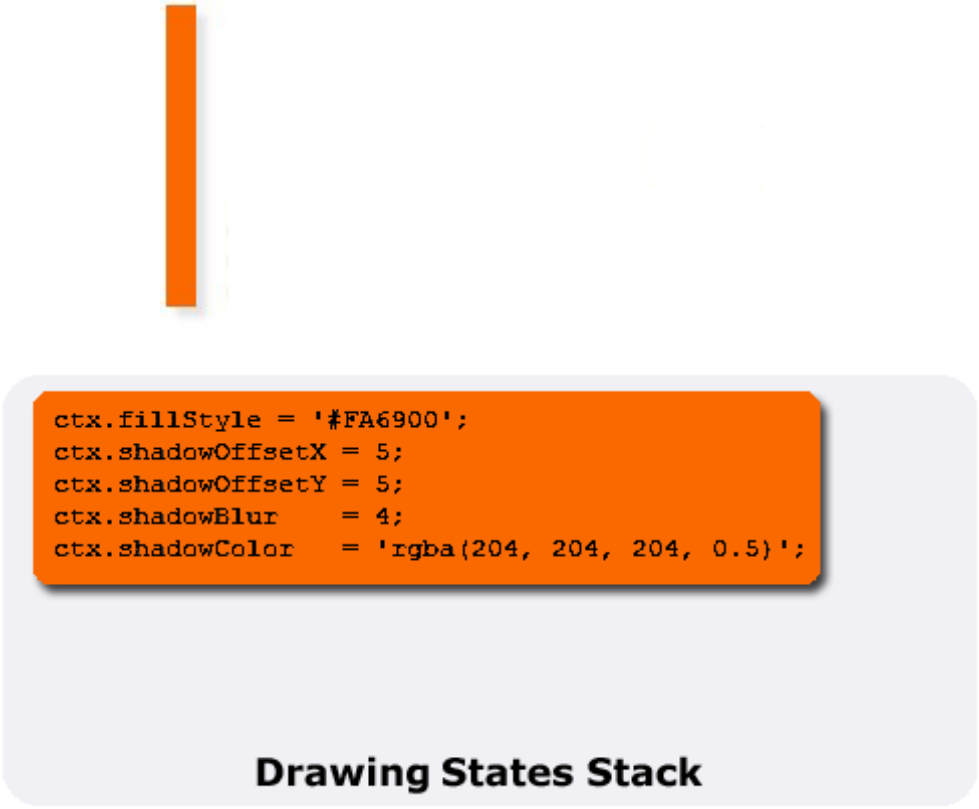
```
<!doctype html>
<html>
<head>
<meta charset="UTF-8" />
<title>Canvas Test</title>
</head>
<body>
<header> </header>
<nav> </nav>
<section>

<div>
<canvas id="canvas" width="320" height="200">
This text is displayed if your browser does not support HTML5 Canvas.
</canvas>
</div>

<script type="text/javascript">
var canvas;
var ctx;

function init() {
canvas = document.getElementById("canvas");
ctx = canvas.getContext("2d");
draw();
}


function draw() {

ctx.fillStyle = '#FA6900';
ctx.shadowOffsetX = 5;
ctx.shadowOffsetY = 5;
ctx.shadowBlur    = 4;
```

```
ctx.shadowColor   = 'rgba(204, 204, 204, 0.5)';
ctx.fillRect(0,0,15,150);
ctx.save();

ctx.fillStyle = '#E0E4CD';
ctx.shadowOffsetX = 10;
ctx.shadowOffsetY = 10;
ctx.shadowBlur    = 4;
ctx.shadowColor   = 'rgba(204, 204, 204, 0.5)';
ctx.fillRect(30,0,30,150);
ctx.save();

ctx.fillStyle = '#A7DBD7';
ctx.shadowOffsetX = 15;
ctx.shadowOffsetY = 15;
ctx.shadowBlur    = 4;
ctx.shadowColor   = 'rgba(204, 204, 204, 0.5)';
ctx.fillRect(90,0,45,150);
ctx.save();

ctx.restore();
ctx.beginPath();
ctx.arc(185, 75, 22, 0, Math.PI*2, true);
ctx.closePath();
ctx.fill();

ctx.restore();
ctx.beginPath();
ctx.arc(260, 75, 15, 0, Math.PI*2, true);
ctx.closePath();
ctx.fill();

ctx.restore();
ctx.beginPath();
ctx.arc(305, 75, 8, 0, Math.PI*2, true);
ctx.closePath();
ctx.fill();
}

init();
</script>
</section>
<aside> </aside>
<footer> </footer>
</body>
</html>
```

You can copy this code and paste it into a new file called something like saverestore.html and when you open it with an HTML5 friendly browser like Firefox 3.6 it will display the canvas.

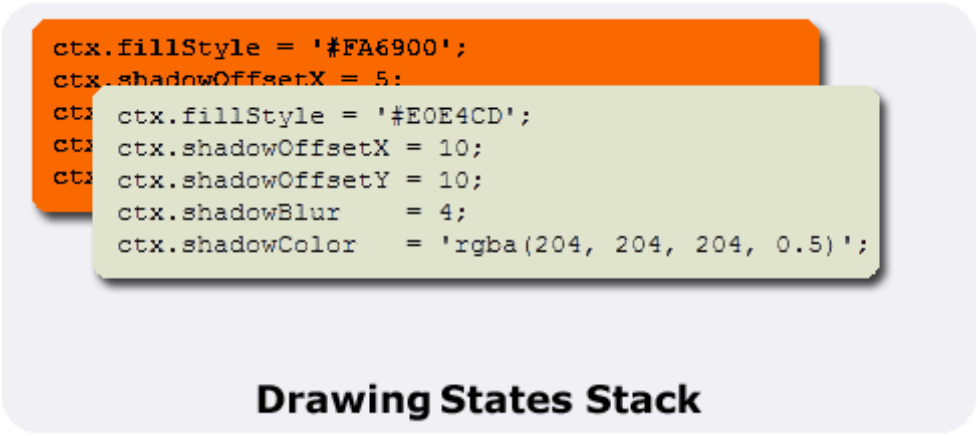
First we draw a rectangle setting the fillStyle and shadow properties for our shape. Then we call `ctx.save()` which adds our settings to the stack.

```
    ctx.fillStyle = '#FA6900';
    ctx.shadowOffsetX = 5;
    ctx.shadowOffsetY = 5;
    ctx.shadowBlur    = 4;
    ctx.shadowColor   = 'rgba(204, 204, 204, 0.5)';
    ctx.fillRect(0,0,15,150);
    ctx.save();
```



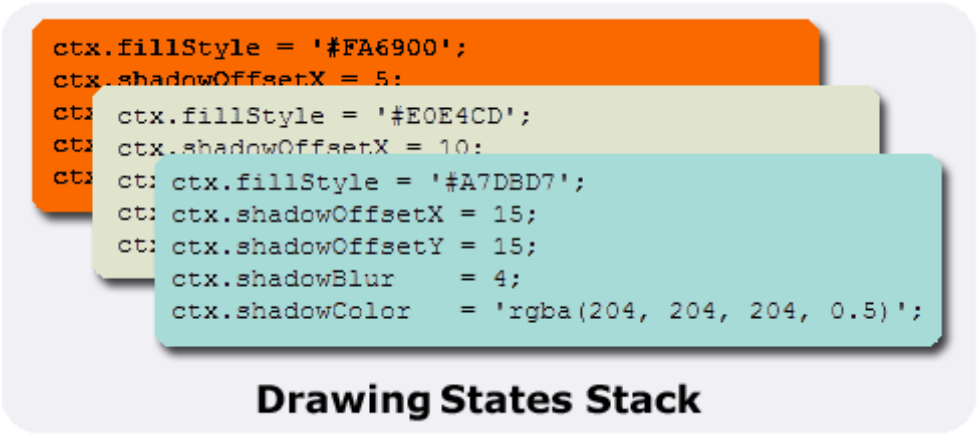
Then we draw another rectangle setting the fillStyle and shadow properties for our shape and call `ctx.save()` which adds our settings to the stack.

```
ctx.fillStyle = '#E0E4CD';
ctx.shadowOffsetX = 10;
ctx.shadowOffsetY = 10;
ctx.shadowBlur    = 4;
ctx.shadowColor   = 'rgba(204, 204, 204, 0.5)';
ctx.fillRect(30,0,30,150);
ctx.save();
```



And we draw another rectangle setting the fillStyle and shadow properties for our shape. Then we call `ctx.save()` which adds our settings to the stack.

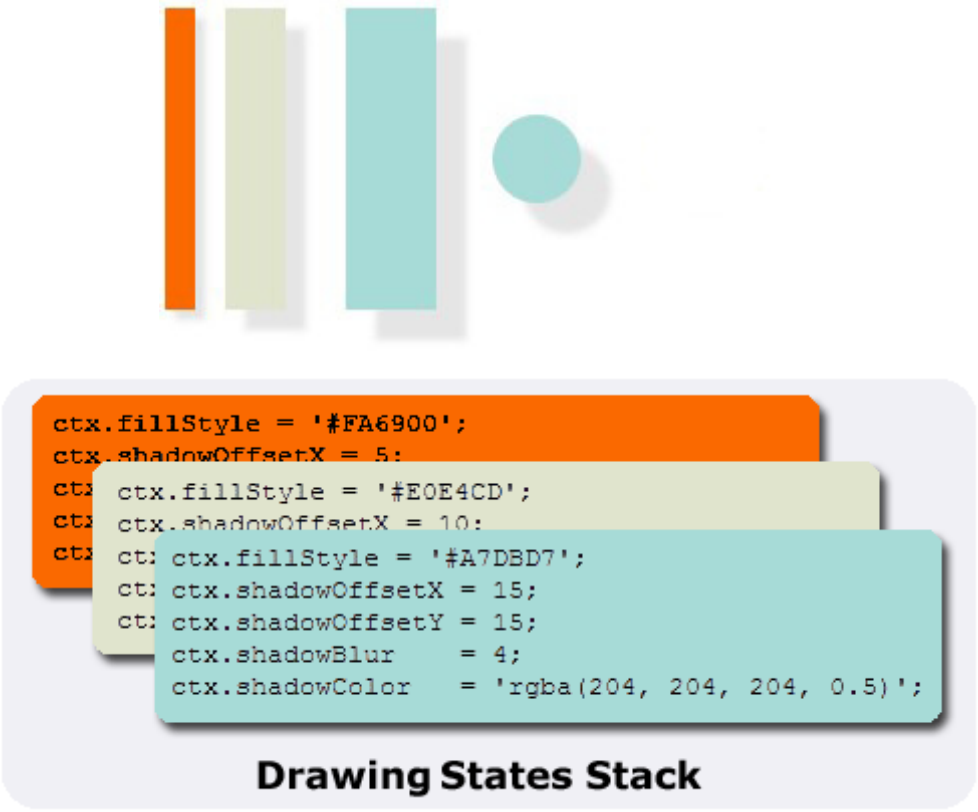
```
ctx.fillStyle = '#A7DBD7';
ctx.shadowOffsetX = 15;
ctx.shadowOffsetY = 15;
ctx.shadowBlur    = 4;
ctx.shadowColor   = 'rgba(204, 204, 204, 0.5)';
ctx.fillRect(90,0,45,150);
ctx.save();
```



Now we call `ctx.restore()` and pop the settings off of the top of our stack and we draw a circle using those settings.

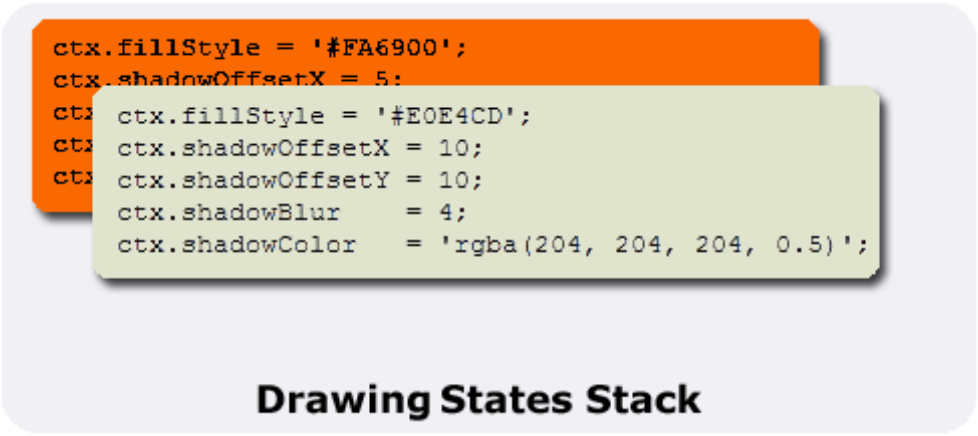
```
ctx.restore();
```

```
ctx.beginPath();
ctx.arc(185, 75, 22, 0, Math.PI*2, true);
ctx.closePath();
ctx.fill();
```



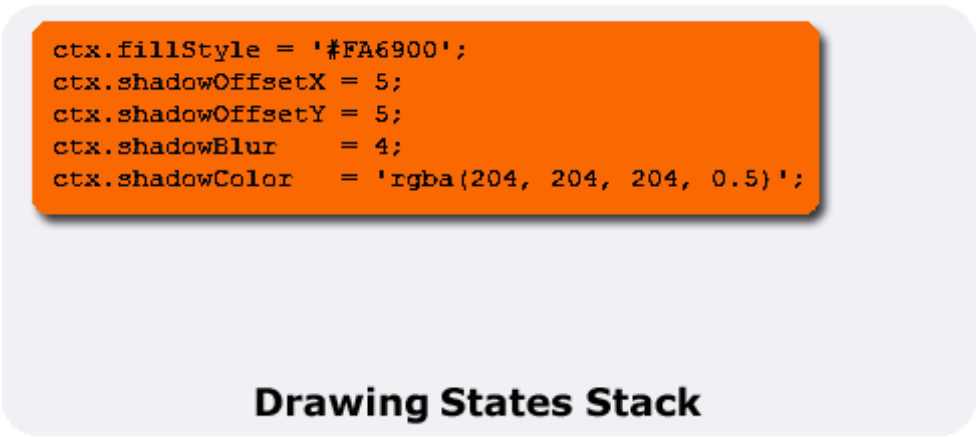
Another call to `ctx.restore()` pops the next settings off of the top of our stack and we draw a circle using those settings.

```
ctx.restore();
ctx.beginPath();
ctx.arc(260, 75, 15, 0, Math.PI*2, true);
ctx.closePath();
ctx.fill();
```



And a final call to `ctx.restore()` pops the next settings off of the top of our stack and we draw a circle using those settings.

```
ctx.restore();
ctx.beginPath();
ctx.arc(305, 75, 8, 0, Math.PI*2, true);
ctx.closePath();
ctx.fill();
```



If you have any questions or corrections, please leave a comment.

Tags: 2d context, Canvas Element, HTML5, javascript, restore, save

This entry was posted on Saturday, July 10th, 2010 at 6:32 pm and is filed under Canvas Element, HTML5. You can follow any responses to this entry through the RSS 2.0 feed. You can skip to the end and leave a response. Pinging is currently not allowed.

24 Responses to “Understanding save() and restore() for the Canvas Context”

Understanding save() and restore() for the Canvas Context « HTML5 Talk • **July 10th, 2010 at 7:09 pm**

[…] Full post here … Categories Uncategorized […]

Tweets that mention Unknown Kadath (a blog about HTML5) » Understanding save() and restore() for the Canvas Context — Topsy.com • **July 10th, 2010 at 7:28 pm**

[…] This post was mentioned on Twitter by nexus11. nexus11 said: RT @HTML5TalkNew post Understanding save() and restore() for the Canvas Context http://bit.ly/9Fe12E: New post Understanding save() an… […]

» Using Multiple HTML5 Canvases as Layers • **July 26th, 2010 at 9:29 pm**

[…] which is used in layer 1 also. For more information on transforms with save() and restore() go here Understanding save() and restore() for the Canvas Context var layer1; var layer2; var layer3; var ctx1; var ctx2; var ctx3; var x = 400; var y = 300; var […]

Jon • **August 18th, 2010 at 8:29 pm**

Thanks for your explanations! I was surprised how difficult it was to find out what the save() and restore() do but your explanations are very good.

Eric • **July 14th, 2011 at 5:57 am**

Whew, thanks. Technical descriptions do a fine job of describing WHAT something does, but you managed to explain WHY I would want to do it, which is very important 🙂

Derek • **July 28th, 2011 at 9:51 pm**

This was an awesome explanation!

Karter • **August 25th, 2013 at 4:52 pm**

Thank you for the explanation. I got it now. Stupid books do not give enough examples like this one.

Pogaca • **October 12th, 2013 at 10:29 am**

This is the best explanation I could ever find. I was so much confused by explanations I saw so far but this one make me understand what save and restore is for. 😀

shyam • **February 5th, 2014 at 6:30 am**

Clear and crispy example . Helped a lot .

Boris Varbanov • **March 11th, 2014 at 7:54 am**

Hey James,
great and quick explanation. Exactly what I was looking for.

Thanks!

Sandip • **May 20th, 2014 at 7:17 am**

is it possible to save entire canvas into database and retrieve canvas from database

Pierre Chamberlain • **September 4th, 2014 at 1:23 pm**

Thanks for the explanation.

I noticed though… just before you begin drawing the circles, you're doing a ctx.save() and ctx.restore() back-to-back. I realize that was probably for demo purposes, but that's an unnecessary step since you would get the same result if you were to draw the cyan Circle immediately after the cyan Rect, correct?

Miguel • **February 23rd, 2015 at 3:59 am**

Thanks for this helpful post. Makes more sense now!

Iliyan • **March 20th, 2015 at 3:10 pm**

Awesome article, thanks a ton ! Nobel prize for you !!!

Ferex • **May 16th, 2015 at 2:11 pm**

Really good explanation!

Srinivas RG • **June 6th, 2015 at 2:24 pm**

Hi Mr. Litten,
Thankfully I found this explanation! Many thanks for elucidating this point. Very good job Sir.

Regards,
Srinivas

thuc101 • **June 22nd, 2015 at 11:28 am**

very helpull, thank you so much

Mayank • **July 7th, 2015 at 12:40 pm**

Thanks, this was helpful

aarthi • **October 10th, 2015 at 1:18 pm**

I am developing an app using phonegap. learning to pick up html5.
I have a canvas and two buttons.
on button 1 click, I do some freehand drawing (using touch events).
on button 2 click, I upload a image on it.
The result should be such that both the image and writing should be displayed.
i tried save and restore
but not getting? pl help

slk500 • **December 21st, 2015 at 1:35 am**

thanks a lot!

mishal153 • **March 10th, 2016 at 12:00 pm**

thanks

Srikanth Rangdal • **July 13th, 2016 at 2:10 pm**

But can we call the saved context out of order in which they are added to the stack?

It would be really helpful if they can be saved with an id or name which would help us recognise each of them individually.

Dave S • **November 12th, 2016 at 11:34 pm**

Thanks for a clear explanation, much appreciated!

Einar • **November 30th, 2016 at 7:45 pm**

Thanks! Interesting explanation.

Leave a Response

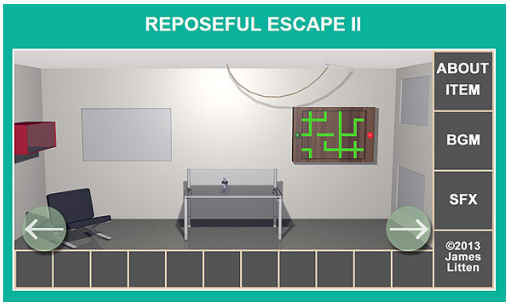
Name (required) Email (will not be published) (required) Comment

Submit Comment

« CSS 3.0 Series: The linear-gradient Object • Make a Maze Game on an HTML5 Canvas »

- James Litten has been doing computer stuff for over 30 years.
  Currently researching Algorithmic Processes of Social Systems Theory, Luhmann flavored (communicative)
  Available for all kinds of freelance work.

- # Can you solve this puzzle?



- **Contact Me**

Skype live:james_4345

Recent Posts

- Example of a WordPress Website Hack Attack
- Alarmism and Super Intelligent AI Vs Humans
- UPDATED How To Fix: External Disk Drive Suddenly Became RAW
- Another HTML Game Using Only Web Standards
- An HTML Game Using Only Web Standards

Search

Search the site: Search