MelonJS - The First Few Days

In the ‘melonJS-gh-pages’ (which is the documentation pages) there are examples. In investigating the framework, I could not get through the ‘platformer’ tutorial so I started looking at the examples. There are 20 examples at this time and the current build for MelonJS is 6.3.0. I am also using a Windows environment (but it that shouldn’t matter) and Chrome 71.0.* (which could matter but probably doesn’t).

Below is a series of Lessons, that I had to figure out and discover. I stopped on Lesson 8, because after I found the example files, I switched gears to writing a simplified Tutorial that builds a MelonJS from the ground up instead of starting in the middle and working outwards.

Lesson 1:

The version of the Framework is important. I ran into problems with one example because the framework was implementing an experimental feature with Cameras. You can find the version at the top of the MelonJS file.

Lesson 2:

The MelonJS framework can fit inside of one file. That’s the idea anyway. Then you can include it in your HTML file with a simple script line

```
<script type="text/javascript" src="../../build/melonjs.js"></script>
```

This is a common way of working with pre-build frameworks such as React and JQuery. There are programs that strip out all the human readable fluff and make a minified file. In theory the two should work the same.

Lesson 3:

MelonJS is a build from a series of files. Even though we see one framework file, the source code is organized into a series of classes (and there accompanying separate files). There is a build process that assembles all these files into one file. I believe (but subject to verification) that the current suggestion is to use the grunt tool to assemble the MelonJS.js file and create its associated minified file. The grunt tool was a little unclear in its documentation, but essentially the grunt tool takes a series of tasks (of course written in the way that ‘grunt’ understands) and executes them. That is why there is a ‘Gruntfile.js’ in the example directory. It contains the associated tasks lists for the build of whatever example you are working on.

Lesson: 4

The melonJS-gh-pages are the document pages. It can be downloaded so you have them off line. It's pretty thorough with an explanation of classes and namespaces. There is also a wide series of examples found in the ./melonJS-gh/pages/examples.

Collision_test

Collision test is a simple example in the melonJS-gh-pages. The following lessons are learned from examining these prebuilt examples.

Lesson 5:

WebStorm seems to serve up the files fixing the CORS problem.

JetBrain's WebStorm IDE was used to investigate the first example. And when it right click the index.html and open in a browser the browser address points to the served address… which for me was something like: "http://localhost:63342/Melon%20Tutorial%201/melonJS-gh-pages/melonJS-gh-pages/examples/collision_test/index.html?_ijt=5v4e8sv72v1poitfqp9islvkav"

Vs when you just navigate to the directory using windows explorer it navigates to "file:///C:/Users/Daniel/Source/Repos/Melon%20Tutorial%201/melonJS-gh-pages/melonJS-gh-pages/examples/collision_test/index.html"

This suggests that WebStorm IDE is acting like a 'server' when it populates the browser. In either case it does not seem to be affected by the CORS issues outlined in the platformer tutorial. But using the WebStorm IDE might be an alternative to using the 'grunt serve' program indicated by the platformer tutorial.

In collision test there is a single JavaScript file called 'main.js'. There are only 3 variables game, PlayScreen, and Smilie and the script file ends with a call to me.device.onReady(...)

Lesson 6:

How does the onReady get called?

If you look in the MelonJS.js file the very first function creates and calls itself. It adds me to the global variable. So now we see how 'me' is added… Line 438 (melonJS

6.3.0) shows me.device created which assigns api.onReady (1011) but returns api (line 1826) to assign to device.

Here is an example function that parallels a lot of the instant create and execute functions similar to the way me is added to the global variable. I have node.js installed so I just pasted this at the node.js prompt… but you can also open the dev tools console and enter the line there and it should execute as well. I know this sounds like basic stuff and maybe it is, but it seems that it has taken me quite a long time to understand the significance of calls like this, and their use seems ubiquitous so I am putting it in here.

```
(function(A){ console.log("Hello, you sent "+ A);return 1;})("someStuff")
---> outputs the following immediately.
Hello, you sent someStuff
1
```

Lesson 7:

Adding global variables makes them accessible anywhere without any reference.

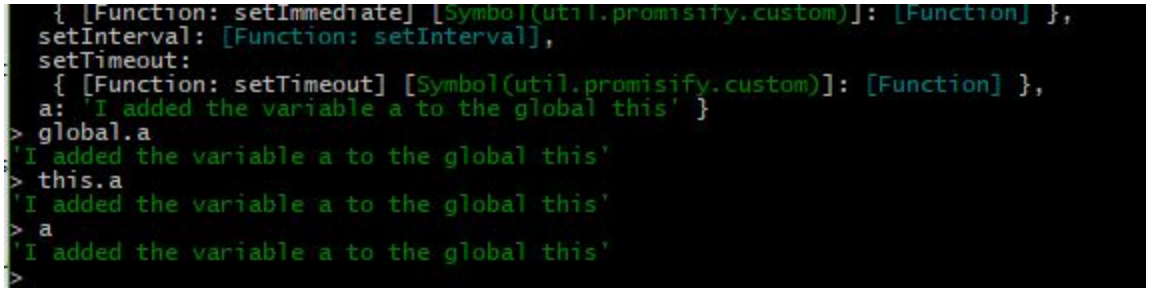
Ok this seems like a no-brainer, and I feel stupid even writing it. But I can't seem to find a better way to write this. So all throughout the mellonJS we find the me object accessed. The reason it is accessible is because it is actually added to the global variable in the very first few lines. I guess why it was so novel to me is that I took it for granted that in order to access a global variable it actually had to be stored somewhere.

Do the following exercise and do it in both node.js prompt and in chrome dev tools (ctrl-shift-I). Type a="I added the a variable to the global this";

Then you can see the variable by typing either 'a', 'this.a'. However in dev tools you can't type 'global.a' because it adds the 'a' to the 'window' object. In node.js though at the node.js prompt you can type 'global.a' to see the variable.

```
> a = "I added the variable a to the global this";
< "I added the variable a to the global this"
> this.a
< "I added the variable a to the global this"
> this
< ▶ Window {postMessage: ƒ, blur: ƒ, focus: ƒ, close: ƒ, parent:
    Window, …}
>
```

And in Node.js



The significance of the above discussion now lets us follow the execution tree for how the MelonJS Framework starts and implements its functionality.

- HTML File is Loaded
- Script File for MelonJS is loaded *AND Executed*
    - me object is created and added to the global variable and because it is HTML the global variable happens to be the window object.
- Script file main.js is loaded *AND Executed* (it may be useful to pull up the main.js file now as you are reading this)
    - Game.onload function is executed
        - me.loader.preloader is called (which we now can see this is from melonJS because it is part of this global object me. Which passes an array of all the game.assets.
        - But there is this call
          ```
          me.loader.preload(game.assets, this.loaded.bind(this));
          ```
          Which is clarified by this video: https://youtu.be/tMhJ4dXbmCM
          But I still find it a bit confusing, but I think when the preload is finished it calls the game.loaded function.

    - Game.loaded: So the execution goes out, loads things then calls the same loaded function of the game object we are looking at.
    - At this point we are just going to assume that the string in game.assets is loaded and the type of asset is determined by its file extension. I might return to this in a bit.
- The me.state sets the playScreen and the state changed to PLAY to indicate the game is running.

Lesson 8: Some Examples are Easier than Others

I'm trying to find a step by step approach to understanding the framework in terms of building blocks. But each example keeps relying on miles of code that I'm just not prepared to understand.

So As I'm looking through the example code I'm an sorting the examples in levels of complexity.

Example	Complexity (1-100)	Notes
collision_test	25	Pretty straight forward… Not to many moving pieces. Only one graphics directory. The game object is placed inside the main.js.,
devicetest	45	No image resources to load but uses the me.Renderable class to write some words on the screen.
drag-and-drop	55	
font_test		
frame_prediction		
graphics	2	This is the perfect introduction because all the code is in the index.html <script> tag. Illustrates the Polygon class, Ellipse
hexagonal		
isometric_rpg		
lineofsight		
multitouch		
particles		
platformer		
shapes		
shoebox		
sticker-knight		

texturepacker		
tiled_example_laoder		
UI		
whack-a-mole		