Documentation for Colour Memory

1. Live Sample

I have deployed a working copy of the game on my personal site at the following url: http://www.killerslime.com

As requested, the game is operated by the **Up, Down, Left, Right** and **Enter** keys. I have also added in support for the **F1** key to enable a *'colourblind'* mode in case the player, like myself, has problems distinguishing colours of similar shades.

Upon completion of the game, the user is presented with a form. Navigation between the form fields and buttons are performed with the **Up, Down, Left, Right** keys too. In addition, I have disabled the use of the **Tab** key.

Note: An actual email will be sent using PHP's mail function upon submission of the form. (There are 3 different messages depending on your submitted score)

2. Installation

a. For convenience's sake, download and install the latest version of the WAMP (http://www.wampserver.com/en/). If you already have Apache, PHP and MySQL installed in your localhost, then that's ok too.
b. Unzip the contents of the archive into the root folder of your localhost directory.
c. A MySQL database and its corresponding user should be created from the credentials listed in **./includes/backend.php** (lines 14 ~ 17).
d. Inside the sql folder, there is a dump of the MySQL database from my website at http://www.killerslime.com. This sql file should be imported into the newly created database.
e. Navigate to http://localhost in your browser and the game should load.

Note: If the game is loaded in a subdirectory, then the Rewrite rule in **.htaccess** file will need to be edited to ensure that the restful API used by the AJAX calls works properly.

3. Frontend Code walkthrough

index.html

line 14:

One thing that I always do is to check whether the user has turned on Javascript in his/her browser. This tiny little snippet of native Javascript is placed at the very top to change the class in the <html> tag if JS is on.

This is helpful as I can then use the CSS to style the page differently depending on whether JS is on or not.
e.g. displaying a '*Please turn your Javascript on*' message.

line 56:

I like to keep template code in my HTML files so that I don't need to type any html markup inside my Javascript code. Makes the code easier to maintain as the JS and HTML are segregated.

line 91:

My Javascript file is placed at the very bottom of the page to ensure that all the DOM is loaded before any scripts are parsed. Note that I've appended a **?v=20160230** to the end of the filename. This helps to prevent browser caching as well as remind me which date I last changed the file.
I do the same for the CSS file at the top also.

_script.js

line 609:

I have a personal style to where I try to play nice with other script libraries that may be attached to the onload and onresize events of the browser. The code within onLoadFunc (line 588) is what gets run after the site finishes loading.

line 557: *startNewGame*

I broke down the steps required in starting a new game, and they were:

a. Clearing the playing area and resetting the score to the starting value of 0
b. Recreating the cards required for a new game
c. Assigning random colours to the cards
d. Adding logic to the cards so that the game can be played
e. Initializing the keyboard controls and targeting cursor

Additional steps like refreshing the leaderboards in the Info area and checking to see if the colourblind mode was enabled are extras.

line 286: *makeGameCards*

The number of cards in the game are generated procedurally and are not hardcoded. The **gridHeight**, **gridWidth** and **matchAmt** values can be changed to create as many (or as few) cards as required in a game.
I had originally planned to include a feature where the user could select from a range of difficulty settings. The grid dimensions and no. of cards to match are listed in the unused **gameDifficulty** variable. This feature was commented out as I wasn't confident I could make it work well with the cursor. It worked well with a mouse though.

line 309: *makeRandomColours*

I used 6 letters (0, 3, 6, 9, C, F) to generate the random permutations of colours used by the cards. The total number of possible colours is 720.
I do know that sometimes, sets of very similar colour may be generated (e.g. #0033CC, #0033FF) so I added the '*colourblind mode'* in.
Unfortunately, the '*colourblind mode'* ended up making the game so much easier I was tempted to change its name to '*easy mode'* instead.
If I had more time, I'd definitely try to refine this feature more to create more distinct colours.

line 65: *cursor* object

The logic for the operation of the game cursor is contained entirely in this object, with methods for each key used in the game as well as all conditions to cater to all the possible interactions by the user.
The cursor will change its size and position according to the item it is hovered over. The values are procedurally generated for the cards but hardcoded for the other elements.

For example:

1. Pressing **Right** at the right edge of the array of cards (regardless of which card you're at) will move the cursor to the restart button. Moving **Left** from the the restart button will send the cursor to the card at the bottom left.
2. The cursor shapes and positions for the score submission form have something I'd like to touch on: **submitCursor** (line 38) contains hardcoded values for the cursor position and size, stored in the form of a 2D array of objects.
   By passing in the cursor's x and y values in that screen, I retrieve the corresponding values in the array to resize and reposition for the corresponding element.

4. Backend Code walkthrough

backend.php

Database access credentials are listed at the top of the file. (The commented ones are the setting for my localhost db and can be ignored)

I also defined constant variables for all the various error messages that would appear. These messages are passed back to the frontend inside the response JSON strings.

The 2 functions in this file are self-explanatory: **db()** returns a pointer to the MySQL database, whereas **sendScoreEmail** does exactly what its name says and send the user a congratulatory email after he/she submits the score at the end of a game.

api.php

There are 2 APIs used for the game:

1. **http://killerslime.com/api/getrank**
   This GET request returns a JSON containing data that will be displayed in the score ranking table. It should be noted that the values in the **rank** column are not stored in the table but are generated when the **SELECT** query is run.

   While the query gets the jobs done now, performance will definitely get worse the more records are stored in the database. A possibly better solution could make use of cron jobs running late at night to calculate the ranking values and store them, but that would mean the rankings table would only be updated once a day.

2. **http://killerslime.com/api/addscore**
   This POST method is called by the frontend AJAX to insert a new record into the database. It is not practical to rely on Javascript to perform error checking so sanitisation of the user's **name** and validation of the **email address** are done at this point.

   If the user input values are problem-free and the user hits the Submit button, all this will be inserted into a record in the database.

   To calculate the user's ranking after the insertion, I use mysqli's **insert_id** method to retrieve the value of the last auto-increment primary key value from my highscores table, and I use this id value in another **SELECT** query that calculates out the user's ranking based on the updated date in the db.

4. Final Thoughts

The code test was challenging and I feel happy that I had an opportunity to really test myself. While I feel that I managed to get the job done, I'm also aware that there is a lot that can be improved.

I did not make use of Gulp, Npm, Bower, Sass or Angular during the process of producing the game and it felt liberating.

All in all, I had a fun time.

Oh yes, before I forget, the PSD for the logo is in the **./img** folder.