Board and Card Game App Coding Standards

HTML

Use Lower Case Element Names:

HTML5 allows mixing uppercase and lowercase letters in element names.

It is recommended that you use lowercase element names because:

- Mixing uppercase and lowercase names is bad
- Developers normally use lowercase names (as in XHTML)
- Lowercase look cleaner
- Lowercase are easier to write

Example:

```
<section>
  <p>This is a paragraph.</p>
</section>
```

Close All HTML Elements:

In HTML5, you don't have to close all elements (for example the `<p>` element). It is recommended that you close all HTML elements.

Example:

```
<section>
  <p>This is a paragraph.</p>
</section>
```

Close Empty HTML Elements:

It is optional in HTML5, but recommended.

Example: `<meta charset="utf-8">`

Use Lower Case Attribute Names:

HTML5 allows mixing uppercase and lowercase letters in attribute names.

It is recommended that you lowercase attribute names because:

- Mixing uppercase and lowercase names is bad
- Developers normally use lowercase names (as in XHTML)
- Lowercase look cleaner
- Lowercase are easier to write

Example:

```
<div class="menu">
```

Example: `<img src="html5.gif" alt="HTML5" style="width:128px;height:128px">`

Spaces and Equal Signs:

HTML5 allows spaces around equal signs, but space-less is easier to read, and groups entities better together.

Spaces and colon in attributes:

Use no space before the colon and one space after the colon for the sake of readability.

Avoid Long Code Lines:

When using an HTML editor, it is inconvenient to scroll right and left to read the HTML code.

Try to avoid code lines longer than 80 characters.

Blank Lines and Indentation:

- Do not add blank lines without a reason.
- For readability, add blank lines to separate large or logical code blocks.
- For readability, add two spaces of indentation. Do not use the tabs or mix tabs and spaces.
- Do not use unnecessary blank lines and indentation. It is not necessary to indent every element.

When elements carry over more than one line of code, indent the contents of elements between the start tag and the end tag. This will make it easy to see where the element begins and ends.

Don't Omit the <html> and <body> Tags:

In the HTML5 standard, the `<html>` tag and the `<body>` tag can be omitted.

The `<html>` element is the document root. It is the recommended place for specifying the page language.

Example:

```
<!DOCTYPE html>
<html lang="en-US">
```

- Declaring a language is important for accessibility applications (screen readers) and search engines.
- Omitting `<html>` or `<body>` can crash DOM and XML software.
- Omitting `<body>` can produce errors in older browsers (IE9).

Don't Omit the <head> Tag:

In the HTML5 standard, the <head> tag can also be omitted.

By default, browsers will add all elements before <body>, to a default <head> element.

You can reduce the complexity of HTML, by omitting the <head> tag:

However, it is not recommended that you omit the `<head>` tag. Omitting tags is unfamiliar to web developers. It needs time to be established as a guideline.

Use Lower Case File Names:

Some web servers (Apache, Unix) are case sensitive about file names: "london.jpg" cannot be accessed as "London.jpg".

Other web servers (Microsoft, IIS) are not case sensitive: "london.jpg" can be accessed as "London.jpg" or "london.jpg".

If you use a mix of upper and lower case, you have to be extremely consistent.

If you move from a case insensitive to a case sensitive server, even small errors will break your web!

To avoid these problems, always use lower case file names.

Nested elements:

Nested elements must be nested appropriately.

Example:

```
<div>
  <p>Some text</p>
</div>
```

The <p> tag and its corresponding closing tag, </p>, are both nested inside the <div> and </div> tags.

Self-closing Elements:

All tags must be properly closed. For tags that can wrap nodes such as text or other elements, termination is a trivial enough task. For tags that are self-closing, the forward slash should have exactly one space preceding it:

```
<br />
```

Rather than the compact but incorrect form here:

```
<br/>
```

The W3C specifies that a single space should precede the self-closing slash (source).

Tag ID's:

These should be the first element within a tag.

CSS

Terminology:

Concise terminology used in these standards:

```
selector {
  property: value;
}
```

property: value makes a *declaration*. Selector and declarations makes a *rule*.

Write Valid CSS:

All CSS code must be valid CSS3.

Encoding of CSS files:

Encoding of CSS files should be set to UTF-8.

Naming Conventions:

Always use hyphens in class names. Do not use underscores or CamelCase notation.

```
/* Correct */
.sec-nav

/* Wrong */
.sec_nav
.SecNav
```

Use lowercase words.

Attribute selectors should use double quotes around values.

Some sources say to e single ('') rather than double ("") quotation marks for attribute selectors and property values.

Refrain from using over-qualified selectors, div.container can simply be stated as .container

ID and Class Naming:

Use meaningful or generic ID and class names.

Instead of presentational or cryptic names, always use ID and class names that reflect the purpose of the element in question, or that are otherwise generic.

Names that are specific and reflect the purpose of the element should be preferred as these are most understandable and the least likely to change.

Generic names are simply a fallback for elements that have no particular or no meaning different from their siblings. They are typically needed as “helpers.”

Using functional or generic names reduces the probability of unnecessary document or template changes.

```
/* Not recommended: meaningless */
#yee-1901 {}

/* Not recommended: presentational */
.button-green {}
.clear {}
/* Recommended: specific */
#gallery {}
#login {}
.video {}

/* Recommended: generic */
.aux {}
.alt {}
```

Numbers in class name and ID declarations SHOULD not be used.

Values:

Always define generic font families like sans-serif or serif.

```
/* Correct */
font-family: "ff-din-web-1", Arial, Helvetica, sans-serif;

/* Wrong */
font-family: "ff-din-web-1";
```

Shorten hexidecimal color values to 3 digits when possible:

Lowercase all hex values. It is easier to read.

```
background: #fff;
```

Do not use unit with 0.

```
/* Correct */
.nav a {
  padding: 5px 0 5px 2px;
}

/* Wrong */
.nav a {
  padding: 5px 0px 5px 2px;
}
```
Do not use default values if they are not necessary to override inherited values.

Space before the value, after the colon

No space before the colon

Do not pad parentheses with spaces

Always end in a semicolon

Use double quotes rather than single quotes, and only when needed, such as when a font name has a space or for the values of the `content` property.

Selectors:

Selectors should be on a single line, with a space after the selector, followed by an opening brace. A selector should end with a closing brace on the next line. Next selector related the previous one should be on the next line with one additional line space between them.

```
.nav li {
}

.nav a {
}
```

Properties:

Every declaration should be on its own line below the opening brace. Each property should:

- have a single soft tab with 2 spaces before the property name and a single space before the property value.
- end in a semi-colon.

```
.site-name span {
  position: absolute;
  top: 0;
  left: 0;
  z-index: 10;
}
```
Properties should be followed by a colon and a space.

Leading 0s:
Omit leading “0”s in values.

Do not put 0s in front of values or lengths between -1 and 1.

```
font-size: .8em;
```

80 Characters Wide:

Where possible, limit CSS files’ width to 80 characters. Reasons for this include

- the ability to have multiple files open side by side;
- viewing CSS on sites like GitHub, or in terminal windows;
- providing a comfortable line length for comments.

PHP

Line Length:

Try to avoid code lines longer than 80 characters.

Spacing:

Related lines of code should be grouped into blocks, separated from each other to keep readability as high as possible. The definition of "related" depends on the code :)

For example:

```
<?php

if ($foo) {
   $bar = 1;
}
if ($spam) {
   $ham = 1;
}
if ($pinky) {
   $brain = 1;
}
?>
```

is a lot easier to read when separated:

```
<?php

```

```
if ($foo) {
    $bar = 1;
}

if ($spam) {
    $ham = 1;
}

if ($pinky) {
    $brain = 1;
}
?>
```

Spacing:

Put spaces on either side of binary operators, for example:

```
// No:
$a=$b+$c;

// Yes:
$a = $b + $c;
```

Put spaces next to parentheses on the inside, except where the parentheses are empty. Do not put a space following a function name.

Assignment Expressions:

Using assignment as an expression is surprising to the reader and looks like an error. Do not write code like this:

```
if ( $a = foo() ) {
    bar();
}
```

Space is cheap, and you're a fast typist, so instead use:

```
$a = foo();
if ( $a ) {
    bar();
}
```

Naming:

- Always choose meaningful and specific names.
- Only alphanumeric characters

Use lowerCamelCase when naming functions or variables. For example:

```
private function doSomething( $userPrefs, $editSummary )
```

Use UpperCamelCase when naming classes: `class ImportantClass`. Use uppercase with underscores for global and class constants: `DB_MASTER`, `Revision::REV_DELETED_TEXT`. Other variables are usually lowercase or lowerCamelCase; avoid using underscores in variable names.

- Method names MUST be declared in `camelCase`.