Battleship Gameboard
S19 CSCI470 Web Science

Assigned: 2019-01-25
Reference: JavaScript, CSS, HTML5, Ajax, JSON Due Date: 2019-02-04

Summary:

Abstract:	Battleship Gameboard
Objectives:	1. HTML5 Semantic Tags - Structural 2. Simple CSS3 for Presentation 3. JavaScript for Behaviors 4. IIFE and Scoping 5. JSON and AJAX 6. Progressive Enhancement
Grading:	45 pts - A $\geq$ 41.85; A- $\geq$ 40.50; B+ $\geq$ 39.15; B $\geq$ 37.35; B- $\geq$ 36; C+ $\geq$ 34.65; C $\geq$ 32.85; C- $\geq$ 31.75; D+ $\geq$ 30.15; D $\geq$ 28.35; D- $\geq$ 27
Outcomes:	R2 (CAC-a,c,i, j, k; EAC-a, b, c, e, k, 1, 2); R5 (CAC-a, i; EAC-a, k) (see syllabus for description of course outcomes)

Project Description:

You are to use the files provided to create the Battelship Gameboard that must consist of the following components:

1. A Header area
   (a) Title of the Game, Authors, and Revision Number/Date
   (b) Toolbar area for controls
2. Main area
   (a) A Dock for the Ships - on the left or right side of the two grids
   (b) Two 10x10 grids, side by side (the left being the *ocean* grid and the right being the *target* grid.)
3. Footer area
   (a) Status area
   (b) copyright and help information

The user will select a ship from the *dock* area and then drag it to an unoccupied tile on the *ocean* grid and then drop the ship. The corresponding ship will be placed on the *ocean* grid in such a way that the different parts of the ship will occupy the corresponding tiles in either a horizontal or vertical fashion - you may decide how the user can change the orientation of the ship.

The events generated from the user interaction will not directly change the layout of the *ocean* grid. Instead, the events will modify a *game state model* - implemented as a JSON object. Every time the game state model is updated, a *redrawGrid* method must be called to render the tiles on the *ocean* grid to correspond to the game state model. A game state model JSON object has been provided to you in the file `bsState.json`. You may copy its contents into a state variable for this project.

The goal of this project is (a) to allow the user to place ships on their *ocean grid* and (b) keep the game state model JSON object and the tiles comprising the *ocean* grid in sync.

Obtaining Project Files and Setting up the Initial Web App:

1. Logon to `gitlab.cs.mtech.edu` and locate the project `bsGameBoard` under the |S19 CSCI470| (sub)group, and then fork this project into your own account.
2. Navigate to your own account and locate the `bsGameBoard` project to just forked, and copy the project url
3. Next, logon to `csdept16.mtech.edu` using your Department username and password.
4. Execute the command `ls -l` and you should see a long listing of the files and directories in your home directory. You should see a particular directory `public_html` in your home directory. Files in this directory are available to be served by the web server running on the system.
5. Change into your `public_html` directory by executing `cd public_html` at the command prompt
6. Issue the command `git <project_url>`, where `<project_url>` is the url you copied in the above step. This will create the project folder inside your `public_html` directory,
7. Once the clone is complete, go to your lab machine, open a web browser, and go to the url `http://csdept16.mtech.edu/~<username>/bsGameBoard`, where `<username>` is your username you used to logon to `csdept16.mtech.edu`.
8. You should see the starter web app in your browser.

Project Activities:

Please perform the following activities in the completion of the assignment.

1. Create the following structure layer:
   - A header area that shows the title of the application; space for a toolbar; and an additional area we will later populate with additional form controls.
   - A footer that has the course, term, your name, and the version of your application and a status area for providing the user feedback from the application.
   - The main area that will house
     - the dock on the left (or right) that will hold the five (5) ships comprising a players' fleet
     - two 10 x 10 grids, side-by-side to the right (or left) of the dock
2. The left 10x10 grid shall be the *ocean grid* on which the player may place his/her ships using a drag and drop operation with the mouse from the ships in the dock
3. The right 10x10 grid shall be the *target grid* on which the player may call out attacks on his opponent's ocean grid. This grid will also display the results of the attack - hit, miss, sunk. (more to come on this grid)
4. Each grid tile must consist of three (3) overlapping layers
   - layer-1 background layer - may display anything appropriate
   - layer-2 ship layer - should display the ship component
   - layer-3 effects layer - should display any effects as a result of attack
5. The toolbar should consist of an application menu with at least the following:
   - A Game drop down menu with the following submenu items
     - New Game - terminate existing game; reset game state; ready for Phase-I of game play
     - Load Board - Load from a url - the ocean grid layout - ship positions on tiles

  - Save Board - Save to a url - the ocean grid layout - ship positions on tiles
  - Exit Game - terminate the game and request to close the browser window
- A Help drop down menu with the following submenu items
  - A Help button showing game instructions - in a modal dialog box
  - An About button showing information about the author, limitation, frameworks used and their versions, and any other particulars the end user might need to know to be successful at loading and executing your game
- Feel free to add other menu items to the toolbar

6. Must provide appropriate presentation layer using CSS to provide a consistent look and feel to all application components - there are a lot of components to this seemingly simply gameboard. Make excellent use of CSS selectors, pseudo selectors and rules.
7. Must provide a behavior layer using JavaScript that minimally:
   - Provides behavior to all toolbar menu items
   - Allows the drag and drop of ships from the ship panel to tile locations on the ocean grid
   - Allows for the calling out of attacks to the opponent using button clicks on the target grid
8. Need to make use of at least one `<link rel="import" ...>` to load components of the application
9. Need to use at least one `<template>...</template>` to render content on the page; e.g. create a single time in a template, and then use the template to create the 200 tiles - not a requirement, but a suggestion.
10. Make sure you design your application with progressive enhancement in mind; keeping structure, presentation, and behavior separate.

Figure 1: Programming Project Grading Rubric

Attribute (pts)	Exceptional (1)	Acceptable (0.8)	Amateur (0.7)	Unsatisfactory (0.6)
Specification (10)	The program works and meets all of the specifications.	The program works and produces correct results and displays them correctly. It also meets most of the other specifications.	The program produces correct results, but does not display them correctly.	The program produces incorrect results.
Readability (10)	The code is exceptionally well organized and very easy to follow.	The code is fairly easy to read.	The code is readable only by someone who knows what it is supposed to be doing.	The code is poorly organized and very difficult to read.
Reusability (10)	The code could be reused as a whole or each routine could be reused.	Most of the code could be reused in other programs.	Some parts of the code could be reused in other programs.	The code is not organized for reusability.
Documentation (10)	The documentation is well written and clearly explains what the code is accomplishing and how.	The documentation consists of embedded comments and some simple header documentation that is somewhat useful in understanding the code.	The documentation is simply comments embedded in the code with some simple header comments separating routines.	The documentation is simply comments embedded in the code and does not help the reader understand the code.
Efficiency (5)	The code is extremely efficient without sacrificing readability and understanding.	The code is fairly efficient without sacrificing readability and understanding.	The code is brute force and unnecessarily long.	The code is huge and appears to be patched together.
Delivery (total)	The program was delivered on-time.	The program was delivered within a week of the due date.	The program was delivered within 2-weeks of the due date.	The code was more than 2-weeks overdue.

The *delivery* attribute weights will be applied to the total score from the other attributes. That is, if a project scored 36 points total for the sum of *specification*, *readability*, *reusability*, *documentation* and *efficiency* attributes, but was turned in within 2-weeks of the due date, the project score would be $36 \cdot 0.7 = 25.2$.

Project Grading:

The project must compile without errors (ideally without warnings) and should not fault upon execution. All errors should be caught if thrown and handled in a rational manner. Grading will follow the *project grading rubric* shown in figure 1.

Collaboration Opportunities:

You may collaborate with up to one other person on this project - but you must cite, in the code (html, css, js) each students' contribution.