## **Challenge: Battleship**

### **Introduction**

It’s time to really flex your muscles. Test Driven Development can certainly feel uncomfortable at first, but becomes more natural with practice. We’re going to implement the classic game ‘Battleship’. If you’ve never played it, or need a refresher you can read about it [here](https://en.wikipedia.org/wiki/Battleship_(game)) and you can play an online version [here](http://en.battleship-game.org/).

Since we’re doing TDD, it’s important that you don’t get overwhelmed. Simply take it one step at a time. Write a test, then make it pass.

We have not yet discussed testing the appearance of a webpage. Doing this requires a separate set of tools, and it is outside the scope of this unit. For this assignment do your best to isolate every bit of application functionality from the actual DOM manipulation bits. You can use mocks to make sure that DOM methods like appendChild are being called, but try your best to keep those things outside of the app logic.

### **Assignment**

1. Begin your app by creating the Ship factory function.
   1. Your ‘ships’ will be objects that include their length, where they’ve been hit and whether or not they’ve been sunk.
   2. REMEMBER you only have to test your object’s public interface. Only methods or properties that are used outside of your ‘ship’ object need unit tests.
   3. Ships should have a hit() function that takes a number and then marks that position as ‘hit’.
   4. isSunk() should be a function that calculates it based on their length and whether all of their positions are ‘hit’.
2. Create Gameboard factory.
   1. Note that we have not yet created any User Interface. We should know our code is coming together by running the tests. You shouldn’t be relying on console.logs or DOM methods to make sure your code is doing what you expect it to.
   2. Gameboards should be able to place ships at specific coordinates by calling the ship factory function.
   3. Gameboards should have a receiveAttack function that takes a pair of coordinates, determines whether or not the attack hit a ship and then sends the ‘hit’ function to the correct ship, or records the coordinates of the missed shot.
   4. Gameboards should keep track of missed attacks so they can display them properly.
   5. Gameboards should be able to report whether or not all of their ships have been sunk.
3. Create Player.
   1. Players can take turns playing the game by attacking the enemy Gameboard.
   2. The game is played against the computer, so make ‘computer’ players capable of making random plays. The AI does not have to be smart, but it should know whether or not a given move is legal. (i.e. it shouldn’t shoot the same coordinate twice).
4. Create the main game loop and a module for DOM interaction.
   1. At *this* point it is appropriate to begin crafting your User Interface.
   2. The game loop should set up a new game by creating Players and Gameboards. For now just populate each Gameboard with predetermined coordinates. You can implement a system for allowing players to place their ships later.
   3. We’ll leave the HTML implementation up to you for now, but you should display both the player’s boards and render them using information from the Gameboard class.
      1. You need methods to render the gameboards and to take user input for attacking. For attacks, let the user click on a coordinate in the enemy Gameboard.
   4. The game loop should step through the game turn by turn using only methods from other objects. If at any point you are tempted to write a new function inside the game loop, step back and figure out which class or module that function should belong to.
   5. Create conditions so that the game ends once one players ships have all been sunk. This function is appropriate for the Game module.
5. Finish it up
   1. There are several options available for letting users place their ships. You can let them type coordinates for each ship, or investigate implementing drag and drop.
   2. You can polish the intelligence of the computer player by having it try adjacent slots after getting a ‘hit’.
   3. Optionally, create a 2 player option that lets users take turns by passing the device back and forth. If you’re going to go this route, make sure the game is playable on a mobile screen and implement a ‘pass device’ screen so that players don’t see each others boards!

Examples

* <https://benders-battleship.netlify.app/>
* <https://michalosman.github.io/battleship/>
* <https://igorashs.github.io/vue-battleship/>
* <https://eribloo.github.io/battleship/>
* <https://jcarloslucio.github.io/battleship/>