CSIS 3380

First Session – Project 1

Introduction:

This is a fairly easy project but a good warmup practice. I have given some detailed instructions below so you could build the app on your own. I will also cover some of the topics that you require to know for this project over the class session.

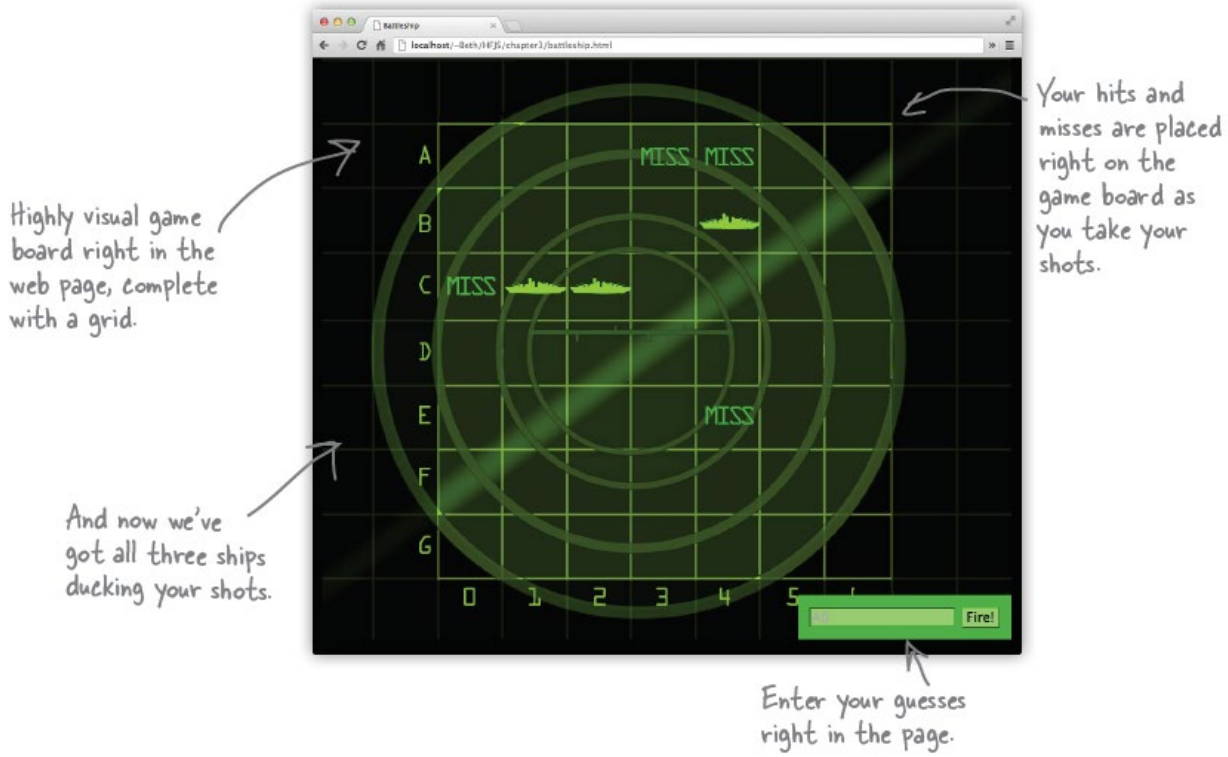
Your Tasks:

- Complete the project and submit it to "CSIS 2300 – Lab 1 – Assignment"
    - Submission: 2 marks
    - Working app: 3 marks

Introduction to the app:

Player does not see the ships but make a guess of the whereabout of the ships by the squares and if the guess hits a part of the ship it will show hit otherwise miss. The player need to continue until all ships destroyed.

In other words, you'll want to create something like this:



During this app, you will build a game called Battles of the Ships. It is an old game. you'll need a visual game board, snazzy battleship graphics, and a way for players to enter their moves right in the game (rather than a generic browser dialog box).

Topics required to know:

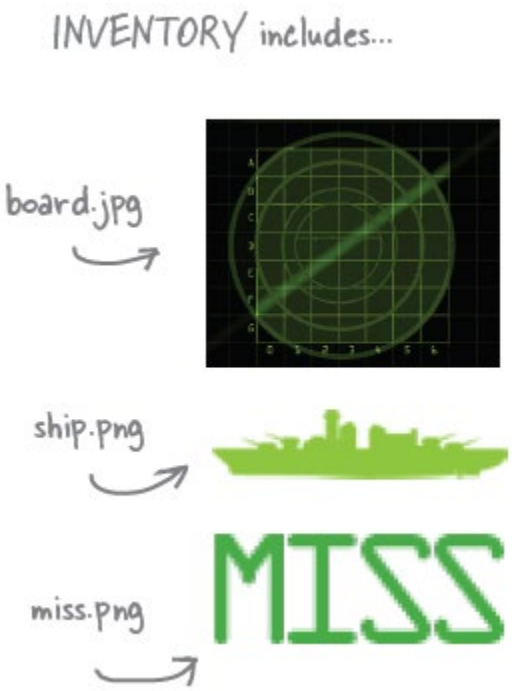
- **HTML Forms**
- **DOM introduction**
- **Arrays**
- **Objects**

Instructions to build the app:

You need to start with a solid HTML and CSS to build the visuals. You will need to program the game through JS (Javascript).

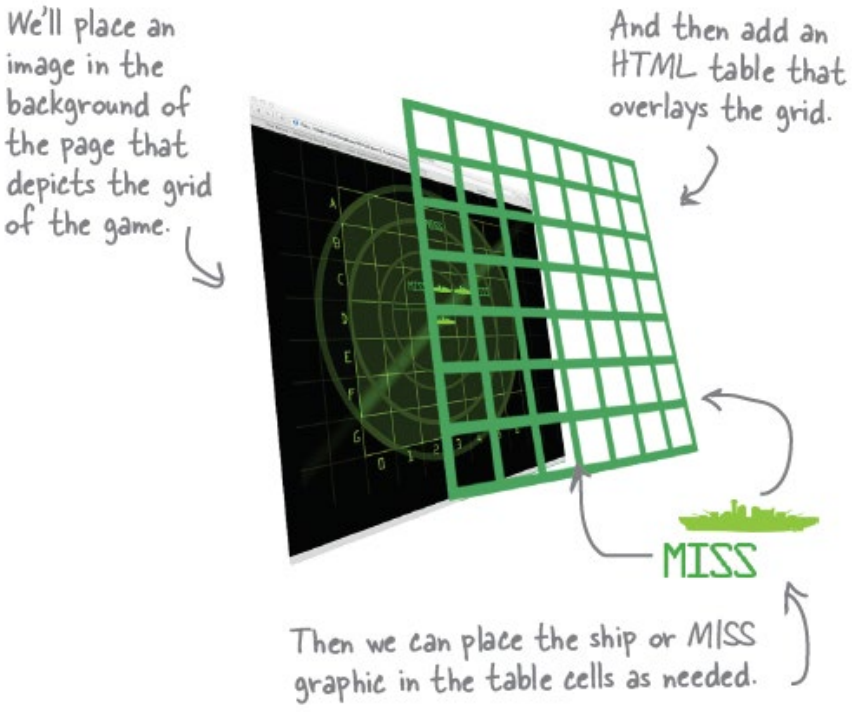
The look of the game is already created, and you can find it in the HTML and CSS files supplied for the project. You will also have the required images.

Here's a toolkit to get you started on this new version of Battleship.



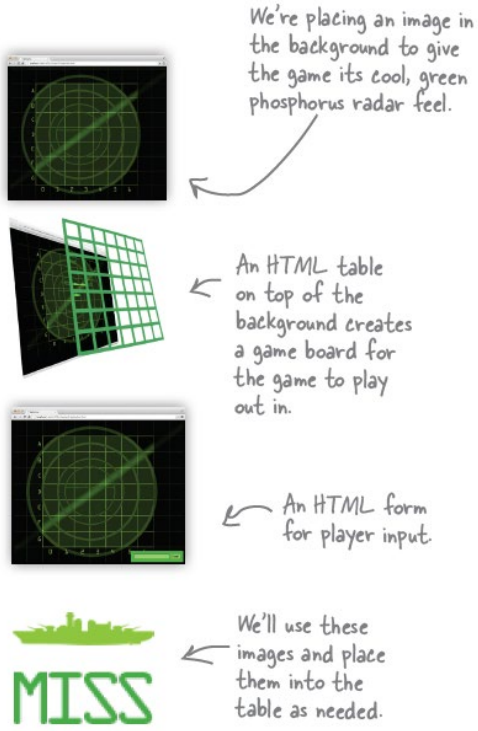
This toolkit contains three images, "board.jpg", which is the main background board for the game including the grid; "ship.png", which is a small ship for placement on the board—notice that it is a PNG image with transparency, so it will lay right on top of the background—and finally we have "miss.png", which is also meant to be placed on the board. True to the original game, when we hit a ship we place a ship in the corresponding cell, and when we miss we place a miss graphic there.

Download everything you need for the game:



So, let's build this game. We're going to take a step back and spend a few pages on the crucial HTML and CSS, but once we have that in place, we'll be ready for the JavaScript.

planning the html for the game Creating the HTML page: the Big Picture

Here's the plan of attack for creating the Battleship HTML page: First we'll concentrate on the background of the game, which includes setting the background image to black and placing the radar grid image in the page. Next we'll create an HTML table and lay it on top of the background image. Each cell in the table will represent a board cell in the game. Then we'll add an HTML form element where players can enter their guesses, like "A4". We'll also add an area to display messages, like "You sank my battleship!" Finally, we'll figure out how to use the table to place the images of a battleship (for a hit) and a MISS (for a miss) into the board.	

Step 1: The Basic HTML

Let's get started! First we need an HTML page. We're going to start by creating a simple HTML5-compliant page; we'll also add some styling for the background image. In the page we're going to place a <body> element that contains a single element. This element is going to hold the game grid.

- Check the code in stuHTML1.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Battleship</title>
    <style>
      body {
        background-color: black;
      }

      div#board {
        position: relative;
        width: 1024px;
        height: 863px;
        margin: auto;
        background: url("board.jpg") no-repeat;
      }
    </style>
  </head>
  <body>
    <div id="board">

    </div>
    <script src="battleship.js"></script>
  </body>
</html>
```

Just a regular HTML page.

And we want the background of the page to be black.

We want the game board to stay in the middle of the page, so we're setting the width to 1024px (the width of the game board), and the margins to auto.

Here's where we add the "board.jpg" image to the page, as the background of the "board" element. We're positioning this relative, so that we can position the table we add in the next step relative to this .

We're going to put the table for the game board and the form for getting user input here.

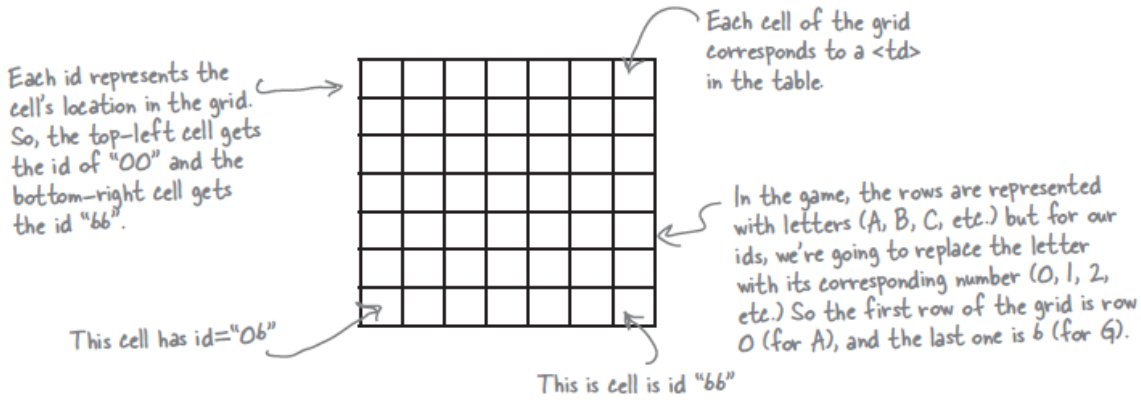
We'll put our code in the file "battleship.js". Go ahead and create a blank file for that.

Step 2: Creating the table

Next up is the table. The table will overlay the visual grid in the background image, and provide the area to place the hit and miss graphics where you play the game. Each cell (or

if you remember your HTML, each <td> element) is going to sit right on top of a cell in the background image. Now here is the trick: we'll give each cell its own id, so we can

manipulate it later with CSS and JavaScript. Let's check out how we're going to create these ids and add the HTML for the table:



Here's the HTML for the table. Go ahead and add this between the tags:

- Stu_HTML2.html

We're nesting the table inside the "board" .

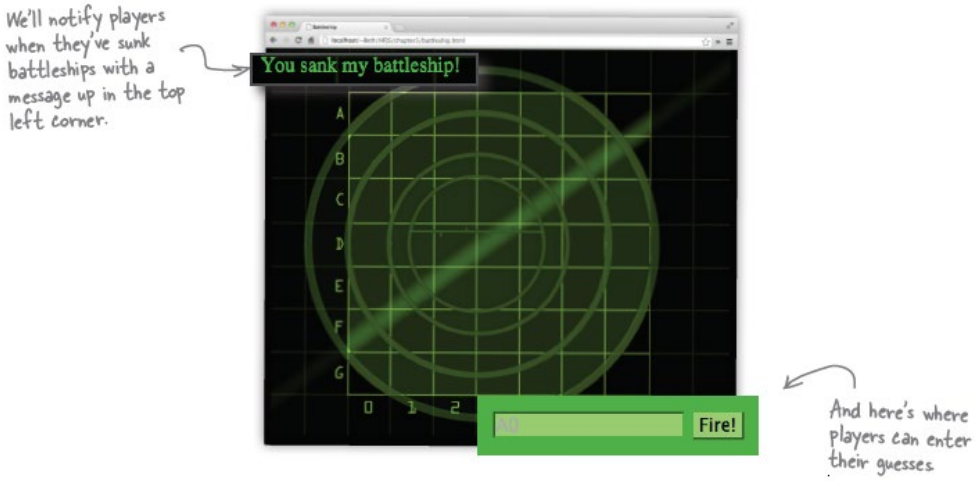
```
<div id="board">
  <table>
    <tr>
      <td id="00"></td><td id="01"></td><td id="02"></td><td id="03">
</td><td id="04"></td> <td id="05"></td><td id="06"></td>
    </tr>
    <tr>
      <td id="10"></td><td id="11"></td><td id="12"></td><td id="13"></td>
<td id="14"></td> <td id="15"></td><td id="16"></td>
    </tr>
    ...
    <tr>
      <td id="60"></td><td id="61"></td><td id="62"></td><td id="63"></td>
<td id="64"></td><td id="65"></td><td id="66"></td>
    </tr>
  </table>
</div>
```

Make sure each <td> gets the correct id corresponding to its row and column in the grid.

We've left out a few rows to save some trees, but we're sure you can fill these in on your own.

Step 3: Player interaction

Okay, now we need an HTML element to enter guesses (like "A0" or "E4"), and an element to display messages to the player (like "You sank my battleship!"). We'll use a <form> with a text <input> for the player to submit guesses, and a to create an area where we can message the player:



Add these:

```
<div id="board">
       <div id="messageArea"></div>
       <table>
              ...
       </table>
       <form>
              <input type="text" id="guessInput" placeholder="A0">
              <input type="button" id="fireButton" value="Fire!">
       </form>
</div>
```

```
<div id="board">
  <div id="messageArea"></div>
  <table>
    ...
  </table>
  <form>
    <input type="text" id="guessInput" placeholder="A0">
    <input type="button" id="fireButton" value="Fire!">
  </form>
</div>
```

The messageArea will be used to display messages from code.

The <form> has two inputs: one for the guess (a text input) and one for the button. Note the ids on these elements. We'll need them later when we write the code to get the player's guess.

Notice that the message area , the <table>, and the <form> are all nested within the "board" . This is important for the CSS on the next page.

Adding some more style

If you load the page now (go ahead, give it a try), most of the elements are going to be in the wrong places and the wrong size. So we need to provide some CSS to put everything in the right place, and make sure all the elements, like the table cells, have the right size to match up with the game board image.

To get the elements into the right places, we're going to use CSS positioning to lay everything out. We've positioned the "board" element using position relative, so we can now position the message area, table, and form at specific places within the "board" to get them to display exactly where we want them.

Let's start with the "messageArea" . It's nested inside the "board" , and we want to position it at the very top left corner of the game board:

```
div#messageArea {
    position: absolute;
    top: 0px;
    left: 0px;
    color: rgb(83, 175, 19);
}
```

```
body {
    background-color: black;
}
div#board {
    position: relative;
    width: 1024px;
    height: 863px;
    margin: auto;
    background: url("board.jpg") no-repeat;
}
div#messageArea {
    position: absolute;
    top: 0px;
    left: 0px;
    color: rgb(83, 175, 19);
}
```

The "board" is positioned relative, so everything nested within this can be positioned relative to it

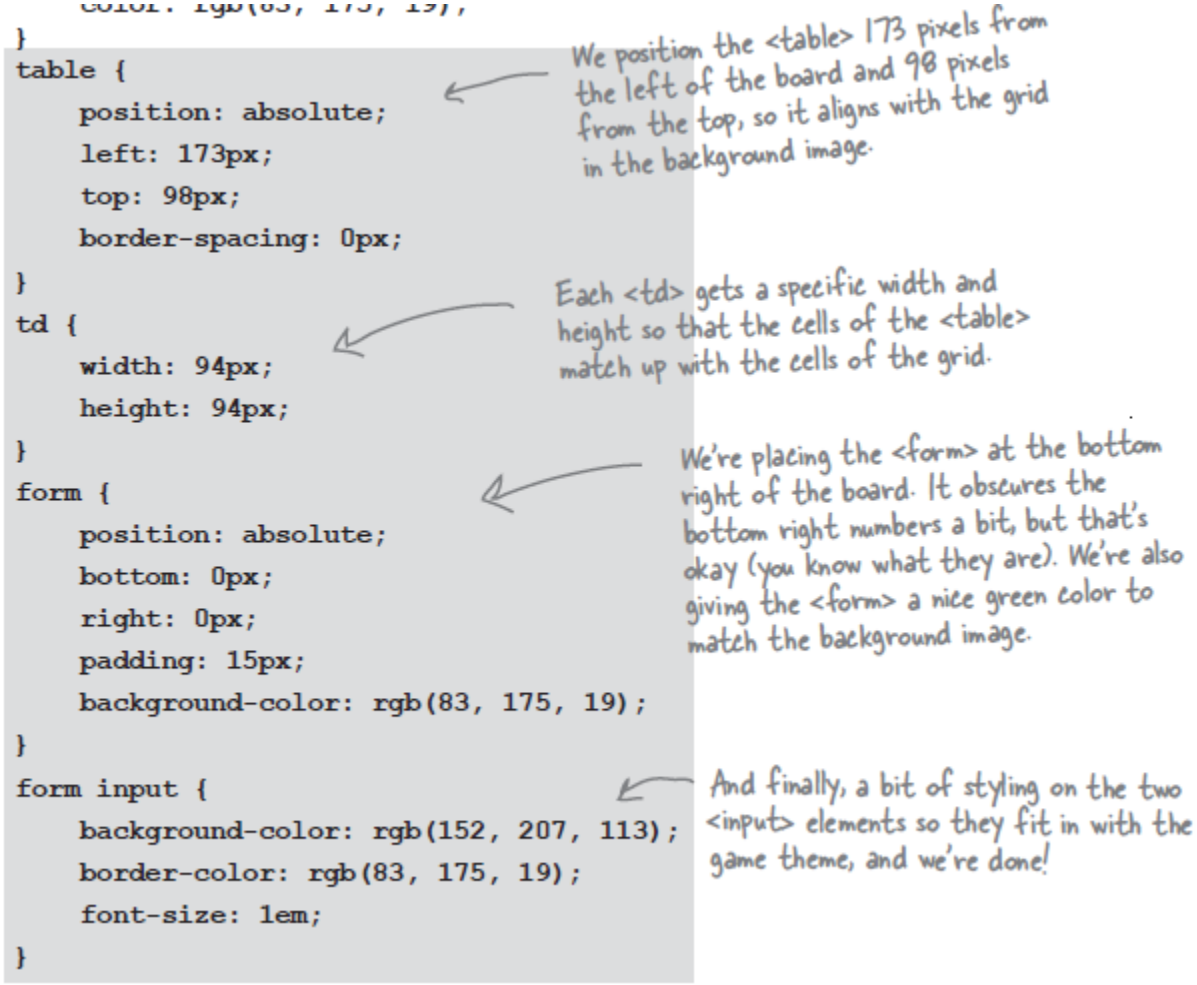
We're positioning the message area at the top left of the board.

The messageArea is nested inside the board , so its position is specified relative to the board . So it will be positioned 0px from the top and 0px from the left of the top left corner of the board .

We can also position the table and the form within the "board" , again using absolute positions to get these elements precisely where we want them.

Here's the rest of the CSS:

```css
table {
    border-spacing: 0px;
    /* could use border-collapse instead */
    /* border-collapse: collapse; */
    position: absolute;
    left: 173px;
    top: 98px;
}
td {
    width: 94px;
    height: 94px;
}
form {
    position: absolute;
    bottom: 0px;
    right: 0px;
    padding: 15px;
    background-color: rgb(83, 175, 19);
}
form input {
    background-color: rgb(152, 207, 113);
    border-color: rgb(83, 175, 19);
    font-size: 1em;
}
```



Get all the HTML and CSS entered into your HTML file and then reload the page in your browser.

Step 4: Placing the hits and misses

We still need to figure out how to visually add hits and misses to the board—that is, how to add either a "ship.png" image or a "miss.png" image to the appropriate spot on the board for each guess. Right now we're only going to worry about how to craft the right markup or style to do this, and then later we'll use the same technique in code.

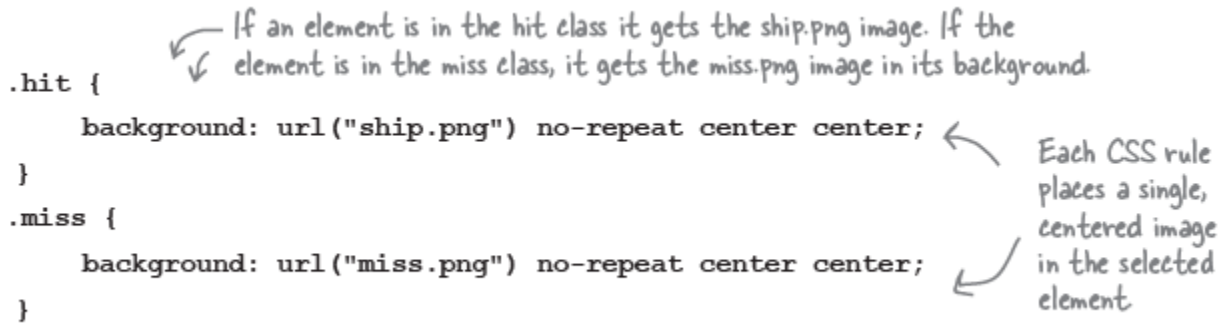
So how do we get a "ship.png" image or a "miss.png" image on the board? A

straightforward way is to add the appropriate image to the background of a <td> element using CSS. Let's try that by creating two classes, one named "hit" and

the other "miss". We'll use the background CSS property with these images so an element styled with the "hit" class will have the "ship.png" in its background,

and an element styled with the "miss" class will have the "miss.png" image in its background. Like this:

```
.hit {
    background: url("ship.png") no-repeat center center;
}
.miss {
    background: url("miss.png") no-repeat center center;
}
```



Using the hit and miss classes

Testing:

Make sure you've added the hit and miss class definitions to your CSS. You may be wondering how we're going to use these classes. Let's do an experiment to see: imagine you have a ship hidden at "B3", "B4" and "B5", and the user guesses "B3"—a hit! So, you need to place a "ship.png" image at B3. Here's how you can do that: first convert the "B" into a number, 1 (since A is 0, B is 1, and so on), and find the <td> with the id "13" in your table. Now, add the class "hit" to that <td>, like this:

<tr>

<td id="10"></td> <td id="11"></td> <td id="12"></td> <td id="13" class="hit"></td> <td id="14"></td> <td id="15"></td> <td id="16"></td>

</tr>

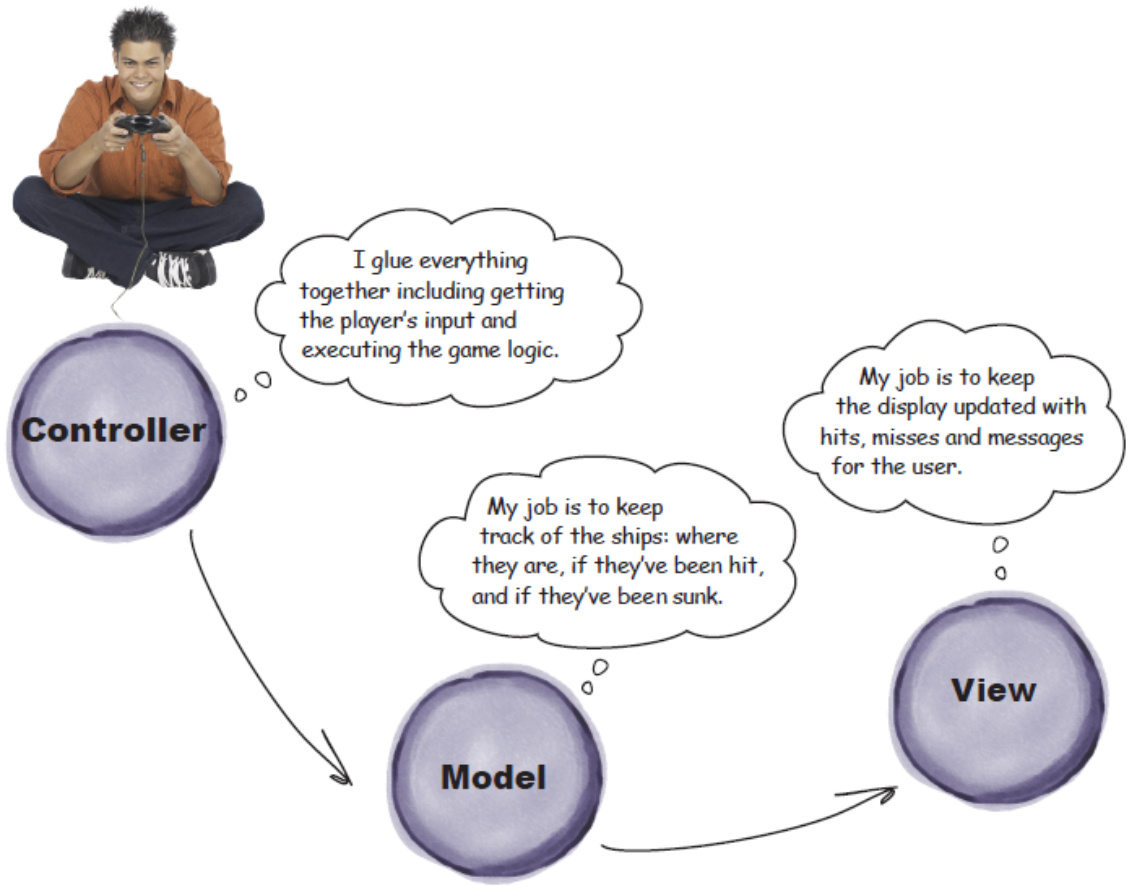
Now when you reload the page, you'll see a battleship at location "B3" in the game board.

This project can get quite long time to go to all details. For the scope of this project it is required to work with HTML, CSS, and javascript.

So, the modules are summarized, you just need to put them together.

MCV model

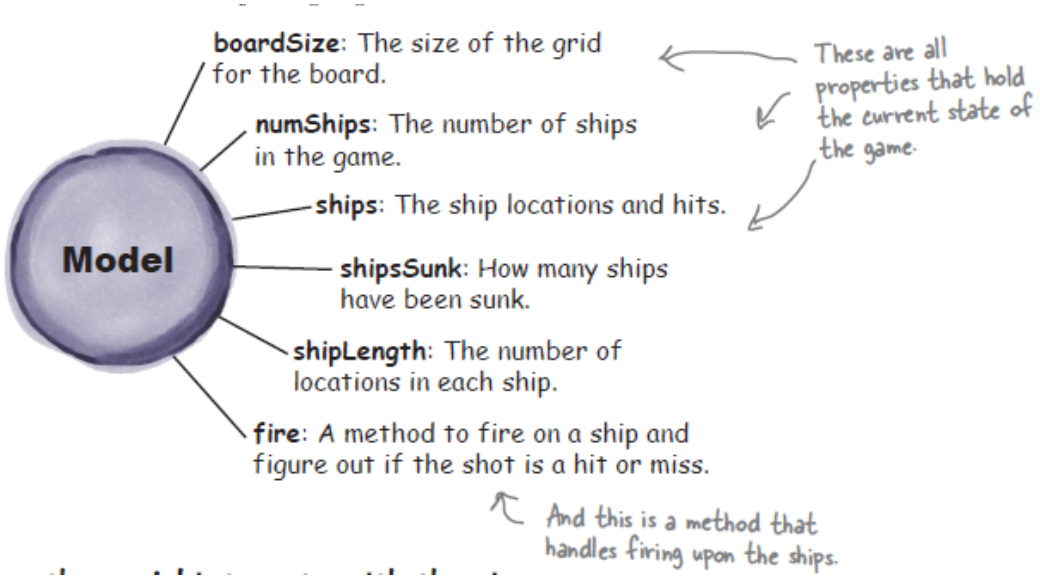
We will have a Model, Controller, and View modules.



I just list the code for each module. You just need to put them together.

The Model:

The model looks like:



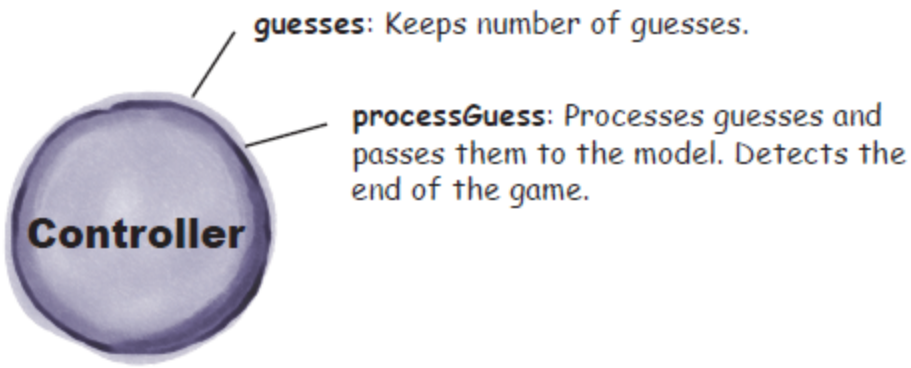
```
var model = {
    boardSize: 7,
    numShips: 3,
    shipLength: 3,
    shipsSunk: 0,

    ships: [
        { locations: [0, 0, 0], hits: ["", "", ""] },
        { locations: [0, 0, 0], hits: ["", "", ""] },
        { locations: [0, 0, 0], hits: ["", "", ""] }
    ],

// original hard-coded values for ship locations
/*
    ships: [
        { locations: ["06", "16", "26"], hits: ["", "", ""] },
        { locations: ["24", "34", "44"], hits: ["", "", ""] },
        { locations: ["10", "11", "12"], hits: ["", "", ""] }
    ],
*/

    fire: function(guess) {
        for (var i = 0; i < this.numShips; i++) {
            var ship = this.ships[i];
            var index = ship.locations.indexOf(guess);

            // here's an improvement! Check to see if the ship
            // has already been hit, message the user, and return true
.
            if (ship.hits[index] === "hit") {
                view.displayMessage("Oops, you already hit that locati
on!");
                return true;
            } else if (index >= 0) {
                ship.hits[index] = "hit";
                view.displayHit(guess);
                view.displayMessage("HIT!");

                if (this.isSunk(ship)) {
```

```
                    view.displayMessage("You sank my battleship!");
                    this.shipsSunk++;
                }
                return true;
            }
        }
        view.displayMiss(guess);
        view.displayMessage("You missed.");
        return false;
    },

    isSunk: function(ship) {
        for (var i = 0; i < this.shipLength; i++)  {
            if (ship.hits[i] !== "hit") {
                return false;
            }
        }
        return true;
    },

    generateShipLocations: function() {
        var locations;
        for (var i = 0; i < this.numShips; i++) {
            do {
                locations = this.generateShip();
            } while (this.collision(locations));
            this.ships[i].locations = locations;
        }
        console.log("Ships array: ");
        console.log(this.ships);
    },

    generateShip: function() {
        var direction = Math.floor(Math.random() * 2);
        var row, col;

        if (direction === 1) { // horizontal
            row = Math.floor(Math.random() * this.boardSize);
```

```
            col = Math.floor(Math.random() * (this.boardSize - this.sh
ipLength + 1));
        } else { // vertical
            row = Math.floor(Math.random() * (this.boardSize - this.sh
ipLength + 1));
            col = Math.floor(Math.random() * this.boardSize);
        }

        var newShipLocations = [];
        for (var i = 0; i < this.shipLength; i++) {
            if (direction === 1) {
                newShipLocations.push(row + "" + (col + i));
            } else {
                newShipLocations.push((row + i) + "" + col);
            }
        }
        return newShipLocations;
    },

    collision: function(locations) {
        for (var i = 0; i < this.numShips; i++) {
            var ship = this.ships[i];
            for (var j = 0; j < locations.length; j++) {
                if (ship.locations.indexOf(locations[j]) >= 0) {
                    return true;
                }
            }
        }
        return false;
    }

};
```

The Controller Object:

Looks like:




```
var controller = {
    guesses: 0,

    processGuess: function(guess) {
        var location = parseGuess(guess);
        if (location) {
            this.guesses++;
            var hit = model.fire(location);
            if (hit && model.shipsSunk === model.numShips) {
                    view.displayMessage("You sank all my battleships,
in " + this.guesses + " guesses");
            }
        }
    }
}
```

The Code for the View Object:

```
var view = {
    displayMessage: function(msg) {
        var messageArea = document.getElementById("messageArea");
        messageArea.innerHTML = msg;
    },

    displayHit: function(location) {
        var cell = document.getElementById(location);
        cell.setAttribute("class", "hit");
    },
```

```
    displayMiss: function(location) {
        var cell = document.getElementById(location);
        cell.setAttribute("class", "miss");
    }

};
```

You will need the following functions to add to your JS file:

```
// helper function to parse a guess from the user

function parseGuess(guess) {
    var alphabet = ["A", "B", "C", "D", "E", "F", "G"];

    if (guess === null || guess.length !== 2) {
        alert("Oops, please enter a letter and a number on the board."
);
    } else {
        var firstChar = guess.charAt(0);
        var row = alphabet.indexOf(firstChar);
        var column = guess.charAt(1);

        if (isNaN(row) || isNaN(column)) {
            alert("Oops, that isn't on the board.");
        } else if (row < 0 || row >= model.boardSize ||
                   column < 0 || column >= model.boardSize) {
            alert("Oops, that's off the board!");
        } else {
            return row + column;
        }
    }
    return null;
}
// event handlers
function handleFireButton() {
    var guessInput = document.getElementById("guessInput");
    var guess = guessInput.value.toUpperCase();
```

```
    controller.processGuess(guess);

    guessInput.value = "";
}

function handleKeyPress(e) {
    var fireButton = document.getElementById("fireButton");

    // in IE9 and earlier, the event object doesn't get passed
    // to the event handler correctly, so we use window.event instead.
    e = e || window.event;

    if (e.keyCode === 13) {
        fireButton.click();
        return false;
    }
}
// init - called when the page has completed loading

window.onload = init;

function init() {
    // Fire! button onclick handler
    var fireButton = document.getElementById("fireButton");
    fireButton.onclick = handleFireButton;

    // handle "return" key press
    var guessInput = document.getElementById("guessInput");
    guessInput.onkeypress = handleKeyPress;

    // place the ships on the game board
    model.generateShipLocations();
}
```

Now put together all the js files and the HTML file that holds the CSS already. Link the HTML to the js file and test your program.

Put all the files in a zip folder and submit.