T5 - Web Development Seminar

T-WEB-500

Day 08

Javascript

2.0

Today's goal is to **debug and achieve** an already-starting project (available in the resources of this subject) from someone else.

It is heavily advised to know and understand the entire code (there is multiple JS files).
It may also be smart to inform yourself about the libraries used.
The previous developer, even though he left you with an unachieved project, tried to use good practice and/or code tricks, inspire yourself from it!
Moreover, he also left a bit of documentation and comments in the code… Sadly he didn't follow proper etiquette and left the commentaries in his native language… Anyway, use all this information to succeed.

We also recommend you to regularly commit, as usual, it could allow you to turn back in an easier way if necessary.

You are only allowed to use the libraries already included in the project (except for the last exercise).

All of the code that you will turn-in (even if it wasn't yours to begin with) must be JSLint valid, without changing the directives present at the top of the files (if you need to do it, you must be able to justify it during your defense).

You must be able to explain every part of the code, even if you didn't code it initially.

Exercises don't always have a link between them, it is however heavily suggested to do them in the given order.

Exercise 00 — Opt

Open your navigator's web console (developers tools) and **correct the JS error** displayed: (`TypeError : player .setGame...`).

For this exercise, you do no need to modify the "game.js" file.

EXERCISE 01 — 1PT

As you probably guessed, the project is about a battleship…
Once ships have been correctly placed, they must be **erased from the big map**.
Make it happen.

For this exercise, you do no need to modify the “game.js” file

EXERCISE 02 — 1PT

Implement the “renderMiniMap” function.
It must **color the mini map cells** (on the left) to match the ships placements chosen by the player, with the color of the ship occupying it.

Ships positions are registered when the player clicks on the big grid.

EXERCISE 03 — 1PT

Prevent ships to collide (that is to say, prevent 2 ships to be on the same cells), or to **go out of the field**.
If one of these conditions is not respected, it must not be possible to valid this ship position (we must stay on the ship placement and not continue to the next one).

Exercise 04 — 1pt

Implement the possibility to **place ships vertically**, using a right click during the positioning phase to change the ship orientation.

Exercise 05 — 2pts

Implement the **computer ships placement**.
The computer must not cheat, it has to follow the same rules as the player (ships can't collide and can't go outside of the map).

For this exercise, you do not necessarily have to create new functions.

Exercise 06 — 1pt

After a shot, the cell shot must now **change color** on the bigger map: red for a hit, and grey for a miss.

Exercise 07 — 2pts

If a player chose a cell where he already shot, the **message must be different** from the classical one.
The information (miss or hit) must not change and be identical to the first shot that happened on this cell.

Exercise 08 — 1pt

Our computer is kind of stupid and always shoot in the top left corner.
Upgrade it a little.

Obviously, it must not cheat and its object must never access the grid property of the player.

Exercise 09 — 1pt

After the computer played, if the player has been hit (and only in this case) the **mini map must be updated** to reflect this lost.
If a ship has been totally sunk, the CSS class “sunk” must be attributed to the matching ship icon (above the mini map).

Exercise 10 — 1pt

After a computer turn, it should be possible for the player to start **playing his next turn**.

It is not yet possible because of a bug…

EXERCISE 11 — 2PTS

Implement the **end of game** detection (function “gameIsOver”).

EXERCISE 12 — 1PT

Implement the possibility to choose **who starts the game** (among human player, computer or random).

EXERCISE 13 — 1PT

Add **a sound for each shot**: a normal sound played after every shot, followed by a second sound depending on the result (one for a successful hit, and another one for a missed hit).

EXERCISE 14 — 3PTS

Add **animations** in the grid cells, following the player’s shot: one for a successful shot and one for a missed one.

EXERCISE 15 — 3PTS

Implement **multiple difficulty levels** for the AI (for example: *easy* could be random shots, and *hard* could be a real algorithm).

EXERCISE 16 — 2PTS

Implement the possibility to have a **hint** (if the player desire it, the computer must suggest a cell to play).

EXERCISE 17 — 5PTS

Implement a **multiplayer** version (a network one, not local).

For this exercise, and only for this exercise, the *socket.io* library is allowed.