Exam project work

Tecnologie e applicazioni web, a.a. 2021/2022

The goal is to develop a full-stack web application, comprising a REST-style API backend and a SPA Angular frontend to let the users play the game of “Battleship” (See https://en.wikipedia.org/wiki/Battleship_(game) for the general rules).

A player starts by signing-in to the application. When logged in, the player can decide to:

- Play against a particular user in the player’s friend list
- Start playing against a random opponent
- Observe a game in progress

In the first case, an invitation is displayed to the challenged player that can decide to accept or refuse the match. In the second case, the system will automatically arrange a match between two waiting players and the game can begin. In the last case, a user can observe a match by choosing it from a list.

Game logic

Battleship is a turn-based guessing game for two players. Each player has two grids of 10x10 cells. On one grid (primary) the player arranges ships and records the shots by the opponent. On the other grid (secondary) the player records their own shots.

A match is divided in two phases. In the first, each player arranges their ships onto the primary grid. In the second phase the game proceeds in a series of rounds. In each round, a player takes a turn selecting a target cell in the opponent’s grid which is to be shot at. If an opponent’s ship is present at that location, the ship is “hit” and the corresponding cell in the secondary grid is marked as hit. On the contrary, if no ships are present, the cell is marked as “miss”. When all the cells occupied by a ship are marked as “hit”, that ship is destroyed.

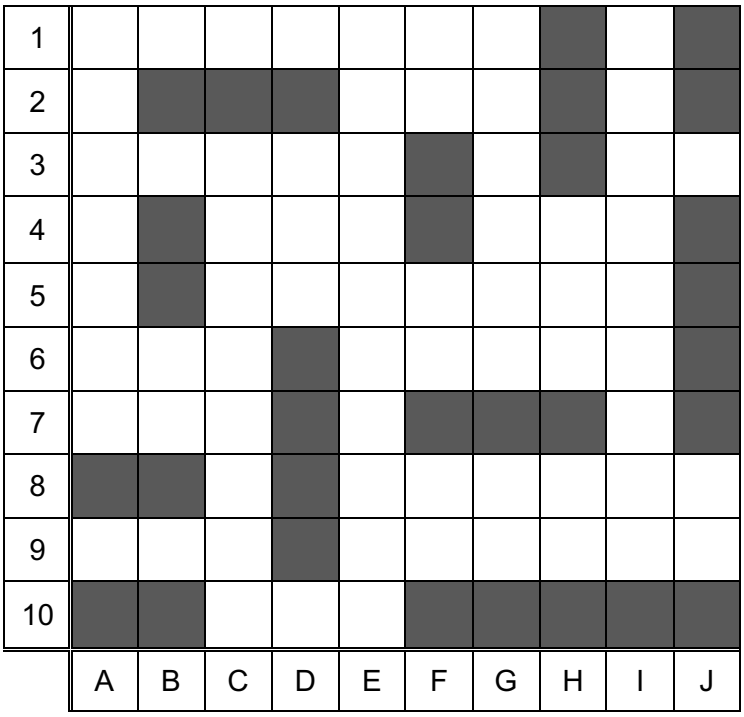
The player who destroys all the opponent’s ships wins the match.

Positioning phase

The game allows the positioning of the following ships for each player:

Type	Dimension (rows x cols)	Quantity
Destroyer	1x2 o 2x1	5
Cruiser	1x3 o 3x1	3
Battleship	1x4 or 4x1	2
Carrier	1x5 o 5x1	1

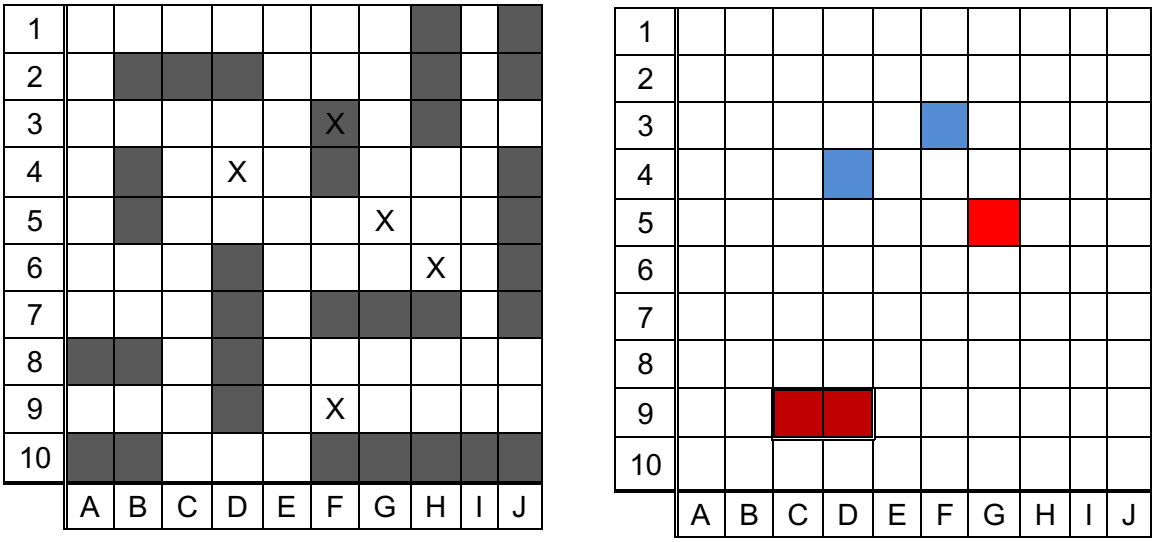
Each ship can be positioned either horizontally or vertically and must not be adjacent to any other ship. An example of possible ship positioning is the following:



Game phase

Player starting the first round is randomly selected. During its turn, a player must shoot to the enemy ships by selecting a cell in the secondary grid. Each cell can be selected only once, ie. a player cannot fire more than once at each location. The system verifies if that cell is occupied by an enemy ship and notifies the player if the shoot was a hit or a miss by marking that cell in the secondary grid. The shot is also notified in the opponent's primary grid.

An example of the match status after 5 turns can be as follows. Note that the graphic layout is just an example and can be changed (improved) in your implementation:



In the image above, destroyed ships are marked in dark red, hits in light red, misses are marked in blue and enemy shots are marked as X in the primary grid.

Additional requirements

The application must keep track of the statistics of each player, including the number of wins and losses, the number of games played, etc.

Users are either standard players or moderators. Moderators can forcefully delete standard players from the system and send messages to any other user. Standard players can only send messages to players in their friend list. The two players involved in a match can always chat regardless their presence in the respective friend-lists.

Architecture

The system must comprise:

- A REST-style **backend** webservice, implemented in TypeScript and running on Node.js. The service must use:
    - MongoDB DBMS for data persistence
    - Express.js for routing

- A web **front-end** implemented as a Single Page Application (SPA) using the Angular framework.
- A **mobile application** created by scaffolding the web front-end in the native container provided by Apache Cordova.

Requirements

Your application must (at least) implement the following features:

1. User management
    a. Signup of standard players. A new player can register by choosing a username, password and entering his details (name, surname, etc.).
    b. Signup of new moderators. Moderators cannot register themselves but must be added by another moderator, specifying a username and a temporary password. At the first login, the new moderator must forcefully set his own credentials (password, name, surname, etc.)
    c. Deletion of standard players by moderators
2. Friend list and chat
    a. A player can add another player in its own friend list. The other player must be notified and accept the request before being added
    b. A player can send a message to another player in its own friend list
    c. A player can chat with the opponent player during a match
    d. All the observers of a match can chat together. Such messages are not visible by the players involved in the match.
3. Gameplay
    a. During the positioning phase players cannot chat with each other. The two opponents must position all the ships on their respective primary grids, and click on a button when ready
    b. The system can optionally help the player in the positioning phase by randomizing the ship position (note that the positioning rules must still be satisfied).
    c. When both the opponents are ready the match is started.
    d. Once a match is started, two players can play the Battleship game as described before.
    e. The starting player is randomly selected.
    f. At any time during the match, the two opponents can chat to each other
    g. Any other user can observe the game in progress and the chat. Observers can only chat among themselves

4. Match arrangement
    a. The system can automatically arrange a match between two players. A user can enter a “wait state”, expressing his will to play against a random opponent. If at least two users are in “wait state”, random pairs are formed, and the corresponding games are started. When matching players, the system should consider the player’s statistics to avoid creating unbalanced matches. Note that players matched in this way are not required to be friends. Chat is possible in this case, but only during the game.
    b. A player can invite a user in his friend list to play a game. If the other player accepts, the game is started.
5. Statistics
    a. A standard player can see his own statistics and of any other player in his friend list
    b. A moderator can see the statistics of any other user

You are welcome to creatively enrich the project with features not expressly indicated. This does not imply to get honors (Lode) but concur to define a positive grade.

Submission

Projects must be submitted via Moodle at the following page:
https://moodle.unive.it/mod/assign/view.php?id=446165

If working on a group, each group member must submit the same file named:
`<group_name>.zip`.

The package must contain:

- **The report of each student**, in PDF format, named `<student_firstname>_<student_surname>_<matriculation_number>.pdf`
- A README.txt file with instructions on how to run the entire application. For example, you can briefly summarize the shell commands to compile the sources, start backend and frontend, etc.
- All the application source code, possibly divided in backend and frontend.

NOTES:

- Do not submit any external library. In other words, remember to delete all the `node_modules` directories before submission.
- Students are responsible for group creation. Moodle will not help nor enforce group submission. Individual students can simply use their surname as group name.

Report

Together with the application, you must submit a report describing the system architecture, the software components used the way in which they contribute to achieving the required functionalities. The report is **strictly individual** and will serve as a basis for the oral discussion of the project.

The report must contain:

- A description of the system **architecture**, listing all the different components and their relationship (i.e. How they work together to implement the application requirements).
- A description of the **data model**, including the collections and the structure of the documents inside each collection.
- A precise description of the **REST APIs**. Such description must contain the list of endpoints together with their parameters and what data is exchanged (give examples in JSON format whenever possible).
- A description of how the user **authentication** is managed, with the associated workflow.
- A description of the Angular frontend, listing the of **components, services, routes**, etc.
- Some examples, possibly with **screenshots**, showing the typical application workflow.

Q&A

For any question about the project work (or the course in general) don't hesitate to contact the teacher at:
filippo.bergamasco@unive.it

I'll be happy to arrange a meeting if needed!

Additional information about the exam is available at:
https://moodle.unive.it/mod/page/view.php?id=446162

Filippo Bergamasco,
Tecnologie e Applicazioni Web, a.a. 2021/2022