Computer networking :

Report - 2^{nd} project

Julien Bolland - Thomas MAZUR
s161622 - s162939

Prof G. Leduc

Academic Year 2018-2019

1 Software Architecture

In order to find a solution, we have implemented the following classes.

1.a WebServer.java

This is the main class of the project. It opens the connection on port 8018 and waits for clients to connect. We used pools so that the server can't be overwhelmed by to much clients. When calling the class, we choose the number of clients that can connect to the server at the same time.

1.b Worker.java

This class inherits from the class Thread, which means that we can use multi-threading on it. It is basically the door between the server and the client, because it handles the reponses of the server according to the requests of the client. These requests are parsed thanks to HttpReader.java. Then, according to the method sent in these request, we have several behaviour.

GET method

When the request use a GET method, then we look at the path we want to reach. If the path is "/", we redirect the page to "/play.html". There, we send a response containing all the html code that has to be computed by the client (*i.e.* the browser), so that our main page is displayed.

Then, when it is displayed, we can hit a tile of our game. As we are using GET method, this means that we are using JavaScript. There, the action is handled by actions.js, which is our JavaScript file containing some event listenners. Our worker sends the play as a JSON file, that informs whether the game is won, lost or if we have hit an unvisited tile. If we won, an alert appears saying that we won. If we d'ont, an alert shows a Game Over. If the tile hasn't already been visited, then JavaScript actualise the image of the tile (whether we hit the water or a ship).

If the request page is halloffame.html, then we only send a response containing the HTML code of the page. It is generated thanks to BattleshipHTML and the Hall of Fame is stored in a Ranking object (see later for further explanations).

POST method

In that case, it means that JavaScript is disabled. Therefore, we can't click on a tile to hit it. We handle this by creating a form, where the user can choose the line and the column to hit. Then he or she clicks on submit and the entire page is reloaded with the corresponding effect on the selected tile.

1.c HttpReader.java

As already said, this class parse the requests of the client. It first reads the entire request, and then breaks it into wanted pieces, *i.e.* the headers fields and the body. Then it instantiates the URL that will be used to access our pages.

1.d BattleshipHTML.java

This class handles the generation of the HTML code. With a certain game going on, we can retrieve all the states of the tiles and then display the corresponding image (hit, splash or not hit yet). Images are handled with the class BattleshipImages, which encodes them with Base64.

1.e Game.java

This class implements the behaviour of a game corresponding to a certain cookie. It keeps track of the battlefield and the state of the tiles.

1.f Ranking.java

This class is responsible for the memorization of the Hall of Fame. We chose to use an ArrayList containing HallOfFame objects, defined by a cookie and a score. When we instantiate a Ranking object, it automatically updates the ranking if it has to. The usage of this object is shared among the workers, so that we had to ensure the correctness of datas thanks to synchronization.

1.g BattleshipException.java

This class allow us to display the messages of the corresponding http error codes on the browser.

2 Multi-thread coordination

As already said, we used pools in WebServer in order to limit the number of connection on the server.

In addition to that, we have two variables that can be shared among the workers : the list of games and the ranking necessary for the Hall of Fame. To avoid several workers to access them and read or write on them, we enclosed the sensitive part in a `synchronized` close. Like that, each worker must quire the lock on these variable before being allowed to access them.

3 Limits

It is sure that if we allow to much clients to connect to the server, it can only crash, even with the pool mechanism. For example, it couldn't handle effectively DOS attacks. If a user instantiates 10000 games to a pool that allows 100 connections, then with our 2 minutes of socketTimeOut, the server would take :

$$\frac{10000}{100} \times 2 = 200\text{minutes}$$

to close all the connections. A new user will then wait more than 3 hours to be connected to the server, which is not really what we expect from a server.

4 Possible Improvements

Instead of simply displaying a new image when selecting a tile, we could also display which type of ship we hit. Moreover, adding a game status and the current score could be great, because it could help the user to know what's left to hit and how far it is to the end of the game.

In order to be more user-friendly, despite the fact that the style of the page could be remade, the images could be interactive, *i.e.* when we hit them, an animation is displayed.