INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN

**Projet informatique**
**GM3-2017/2018**

# Mini-Projet POO : La Bataille Navale


**Étudiants :**
David Albert, Henri Durozay,
Matthieu Bellucci, Thomas Anquetil

**A l'attention de :**

M.Abdulrab

# Table des matières

# Partie 1

# Description du sujet

## Description générale

*BattleshipNET* est un logiciel de jeu de bataille navale se jouant en réseau. La principale faculté du jeu est de pouvoir choisir son adversaire parmi la liste des joueurs en attente de partie. Dans cette première version, le joueur en partie bénéficie d'une grille de bateaux placés aléatoirement. Les prochaines versions, si le développement du logiciel se poursuit, feront apparaître :

1) La possibilité de choisir ses bateaux parmi une liste de bateaux (*)
2) La possibilité de les positionner à son souhait
3) Les communications réseaux seront protégées et ne seront plus uniquement dans un réseau local

(*) Certains bateaux seront standards et d'autres auront des particularités. Exemple : Le Cuirassé (IronClad en anglais) a une taille de 3 et attend de se faire touché partout 2 fois avant de couler, le voilier est un tout petit bateau et le sous-marin n'est découvert que lors du 2ème tire dessus.

## Description technique

Concernant la programmation du projet, nous avons tenté de respecter au mieux les principes de la programmation par objet. De plus, nous avons tenté de mettre en place un modèle MVC (Modèle-Vue-Contrôleur), tout en rajoutant une partie communication réseau entre la vue, coté client, et le modèle, coté serveur. La combinaison du schéma MVC et du réseau n'étant pas paru comme instinctive, la solution apportée au problème peut ne pas être la meilleure.

### 1) Le réseau

D'un point de vue global, nous avons décidé d'avoir une communication réseau centralisée autour d'un serveur tournant sur une machine d'un réseau local (pour le moment). En considérant que, l'IP du serveur et le port de communication, sont communs à tous les utilisateurs de BattleshipNET, on peut jouer sans problème à plusieurs sur le réseau WI-FI de l'INSA ou sur un réseau créé par une box personnelle.

Plus spécifiquement, quand le programme serveur est lancé, il crée un thread chargé d'attendre les connexions clientes. A chaque connexion d'un client, le programme va créer un nouveau thread propre au client, qui vient de se connecter. Ce processus (*ServerCommunicationThread*) représente le lien de communication entre le client et le server. La communication entre le programme client, tournant sur l'ordinateur du joueur, et le programme serveur se fait grâce à une communication socket en mode connecté, utilisant le protocole TCP. La communication socket permet d'avoir des données qui transitent aussi bien, du client vers le serveur, que l'inverse. De plus, le mode connecté permet une connexion durable entre nos deux processus *ClientCommunicationThread* et *ServerCommunicationThread*.

## 2) Le schéma MVC

Le schéma que nous avons tenté de suivre est celui du modèle MVC. Nous ne l'avons réellement mis en place que côté serveur. L'interface graphique se situant forcément coté client, notre "Vue" est représentée par le thread lancé coté serveur et communiquant avec le client (ie *ServerCommunicationThread*).

Ainsi, à chaque événement réalisé sur l'interface graphique, si l'action nécessite la mise à jour des données ou la récupération d'informations, un message est envoyé au serveur.

Le message reçu est traité et envoyé au "Contrôler", qui s'occupe de renvoyer des informations du Modèle ou de vérifier que la mise à jour des données est possible puis demande au "Modèle" de mettre à jour ses données.

Dans notre cas, le Modèle est *DataServer* lors du choix d'un adversaire et *Game* pendant une partie.

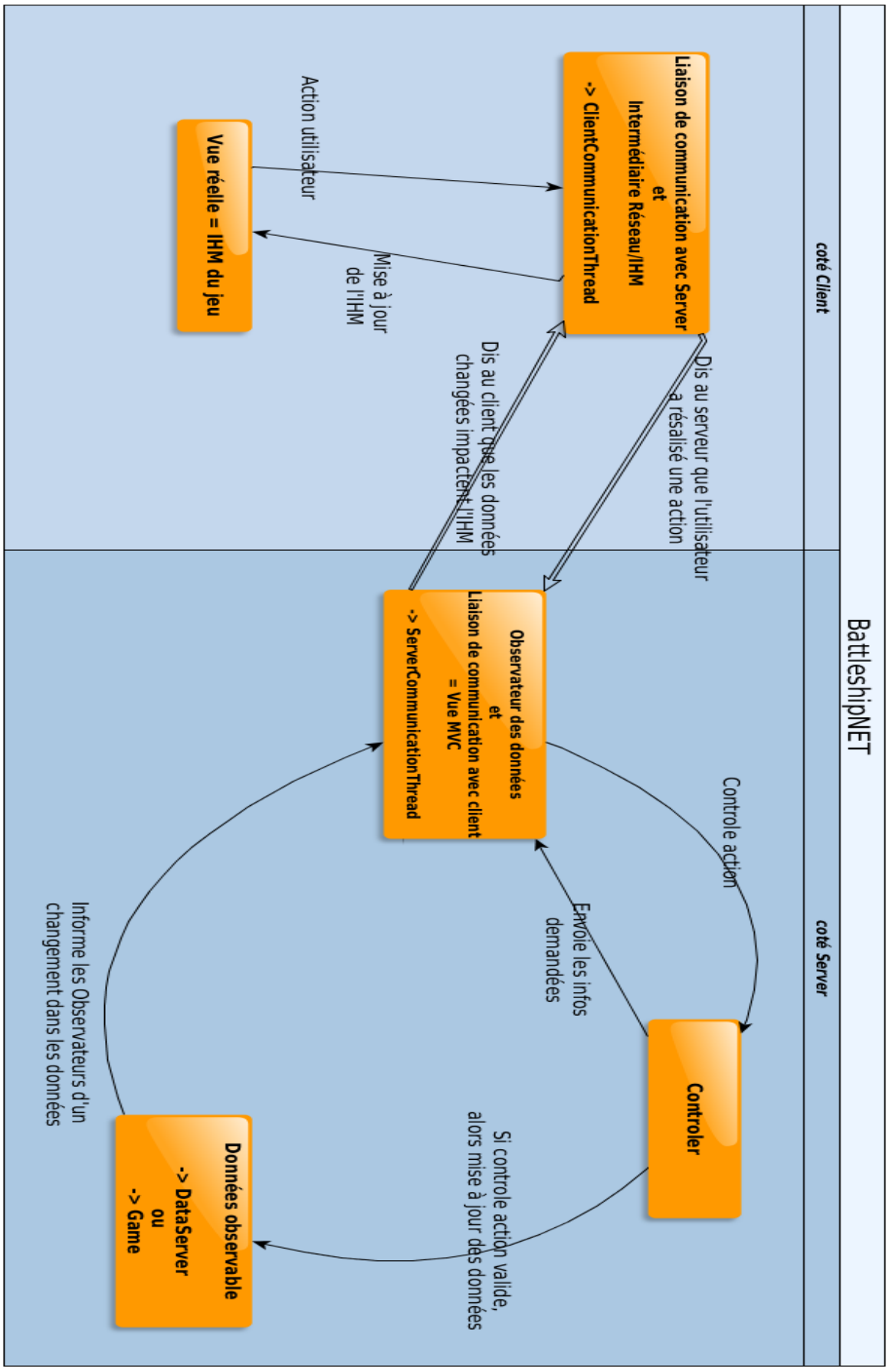
Enfin, le modèle, après avoir modifié ses données, l'indique à tous ses Observateurs (les Vues = *ServerCommunicationThread* dans notre cas).

## 3) L'Interface Homme-Machine (IHM)

L'IHM est la dernière partie du projet que nous avons implémentée. Nous avons utilisé les paquetages SWING pour l'interface graphique et AWT pour la gestion des événements. Le but était de faire une interface graphique et une gestion d'événement fonctionnelle rapidement. Le peu d'expérience en ce qui concerne la création d'IHM ne nous a pas facilité la tâche.

Le schéma ci-après résume le fonctionnement global du système.

## Schéma global du système

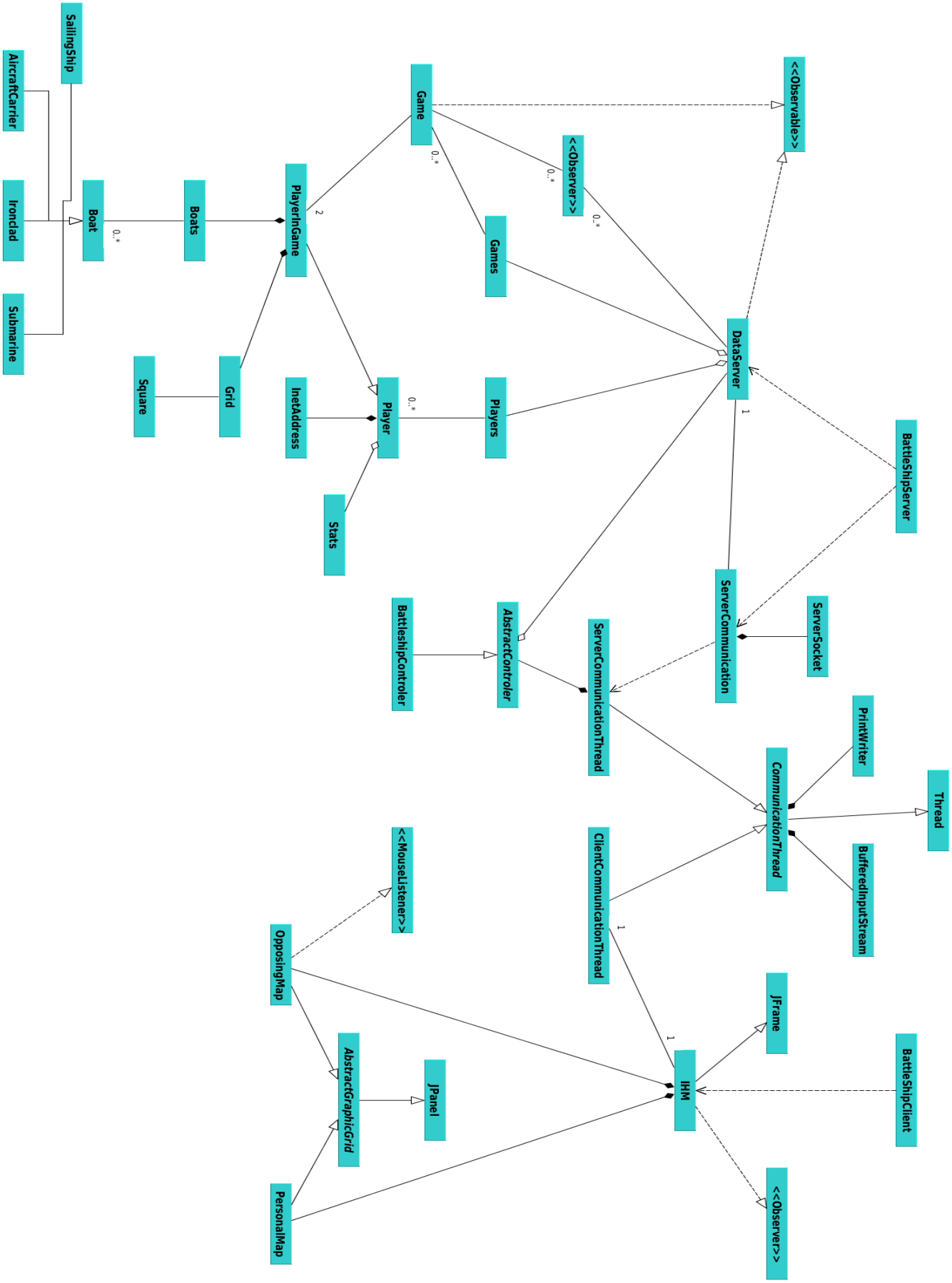


# Partie 2

# Modèle UML

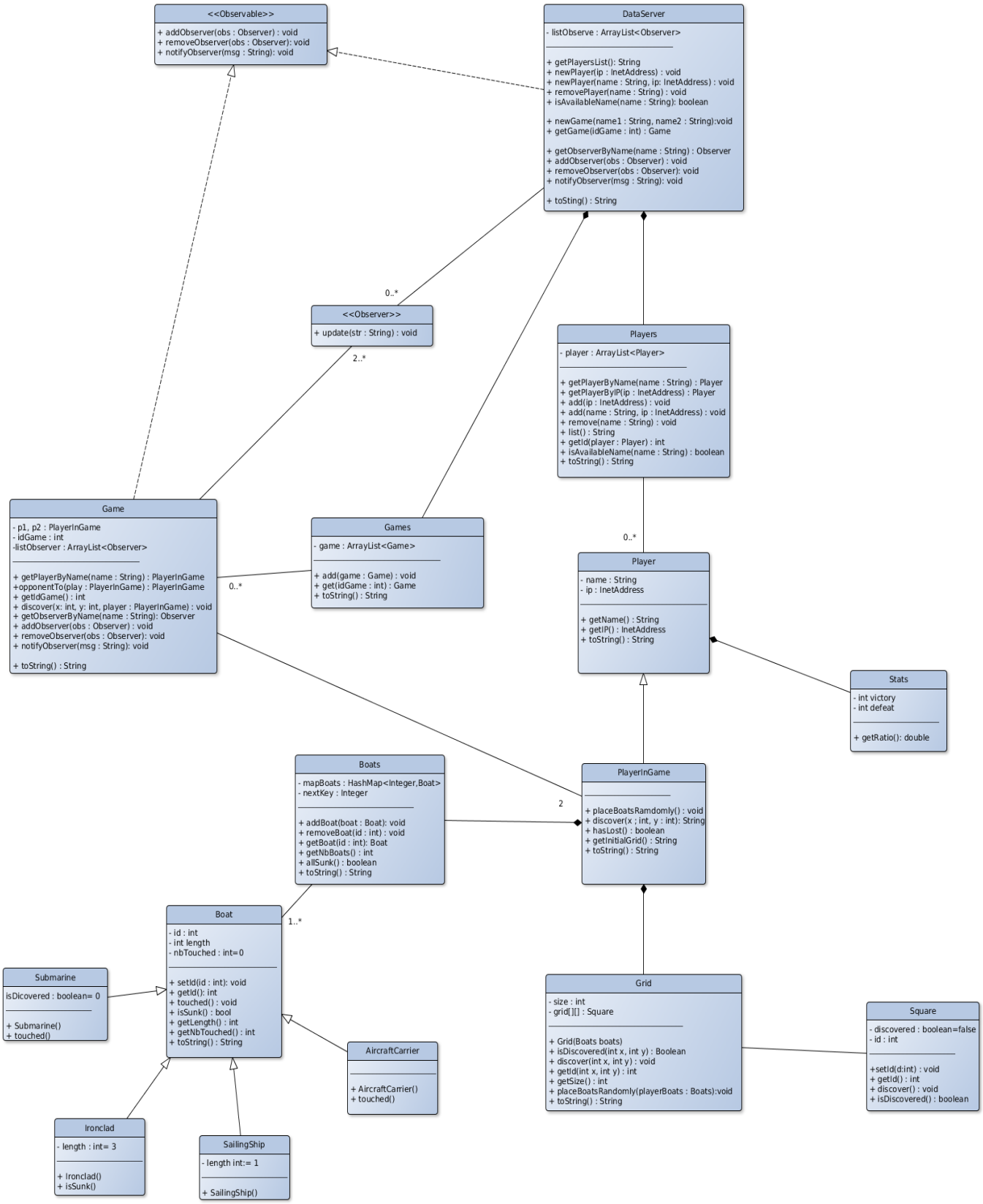
Nous avons séparé notre projet en un ensemble de 5 paquetages. Ces paquetages sont divisés selon le schéma Modèle-Vue-Contrôleur, schéma que nous avons essayé de respecter pour le mieux malgré quelques difficultés.

# Modèle UML général



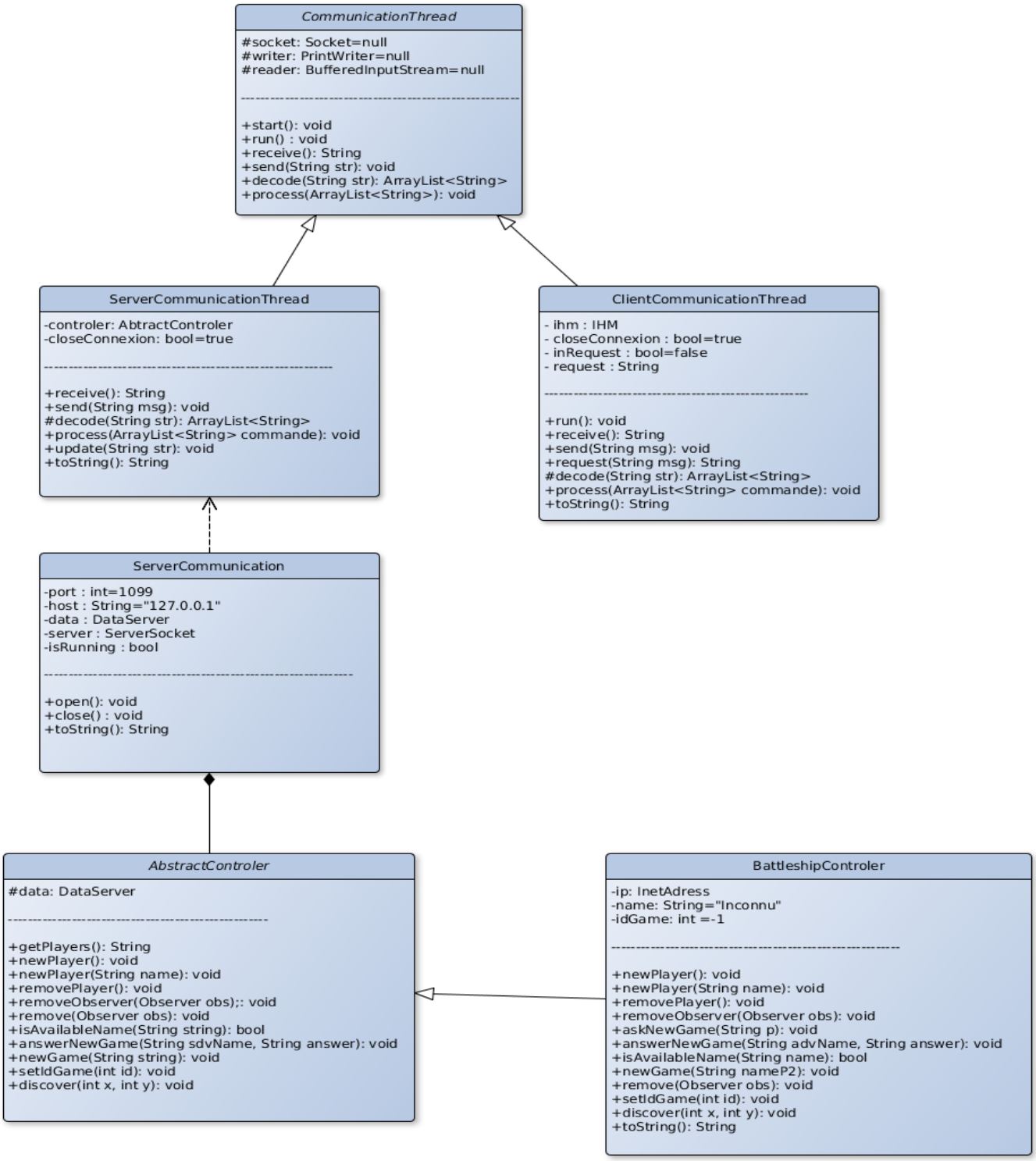
# Modèle UML complet des paquetages Model et Observer

.



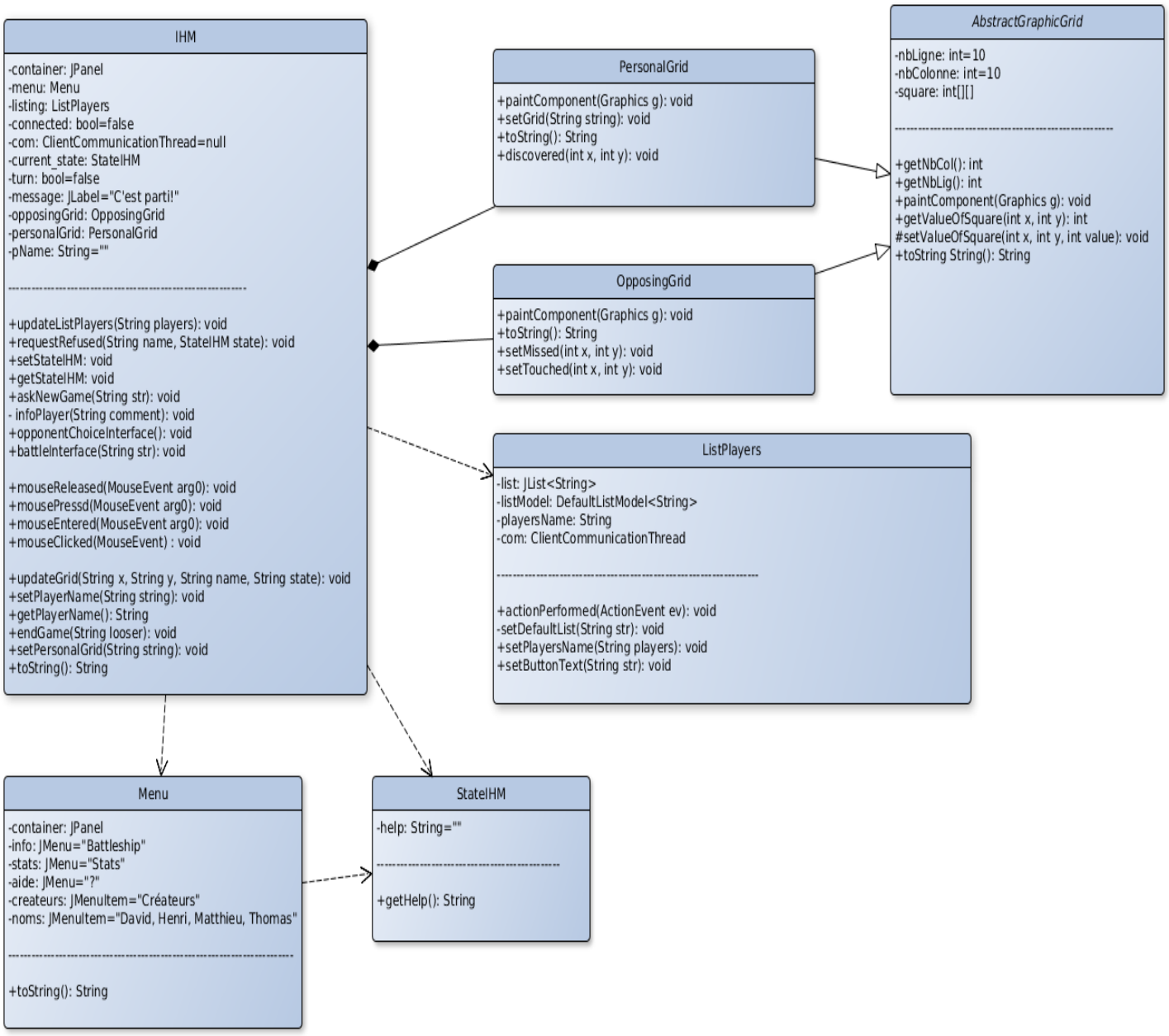
# Modèle UML complet des paquetages Net et Controler

.



# Modèle UML complet du paquetage View

.



# Partie 3

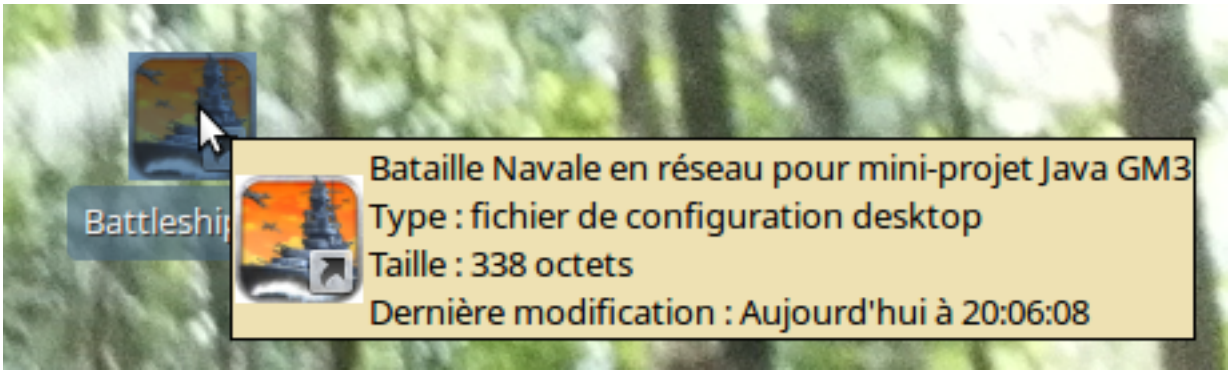
# Code

Voir le lien ci-après : https://github.com/shamrodia74/Battleship

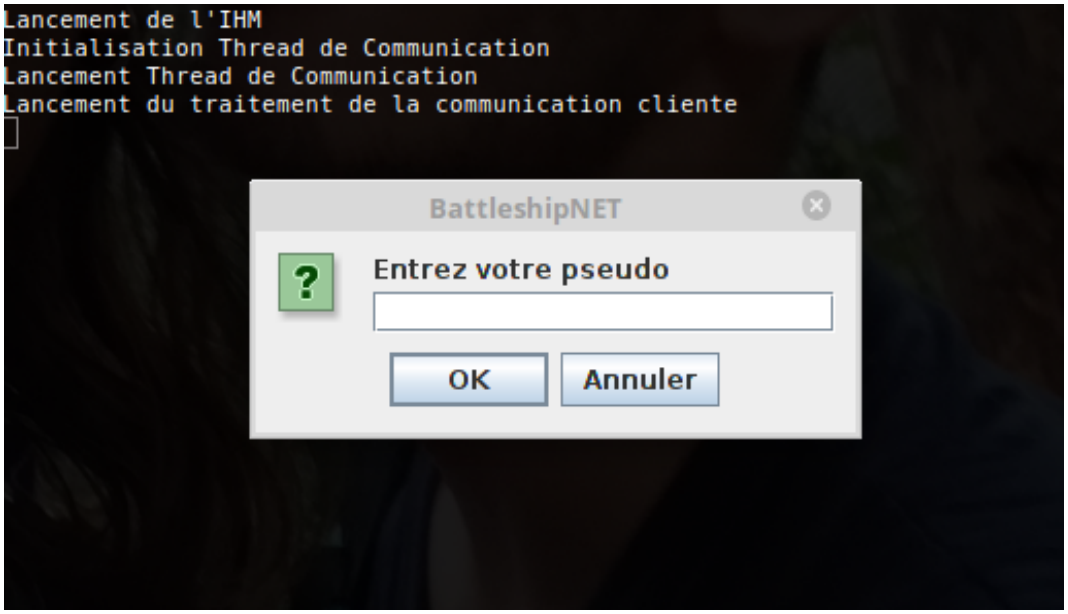
# Partie 4

# Déroulement du programme

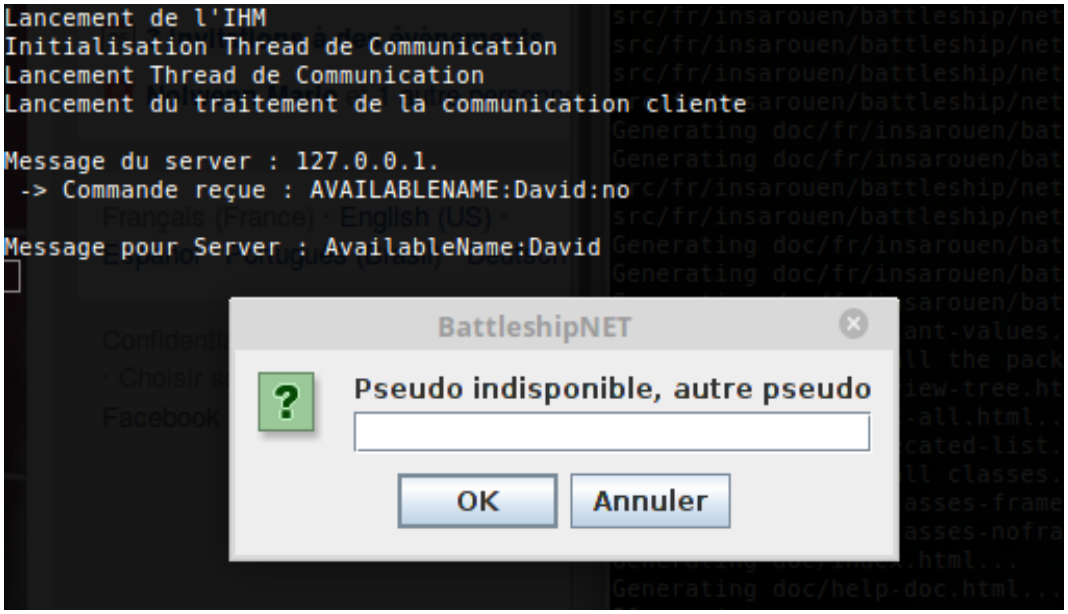
Tout d'abord, l'utilisateur du programme Client doit le lancer le programme, soit depuis le terminal en se plaçant dans le dossier du projet et en tapant
*java -classpath bin/ fr.insarouen.battleship.BattleshipClient* ou par l'intermédiaire d'une archive JAR ou d'un lanceur de programme créé à cet effet.



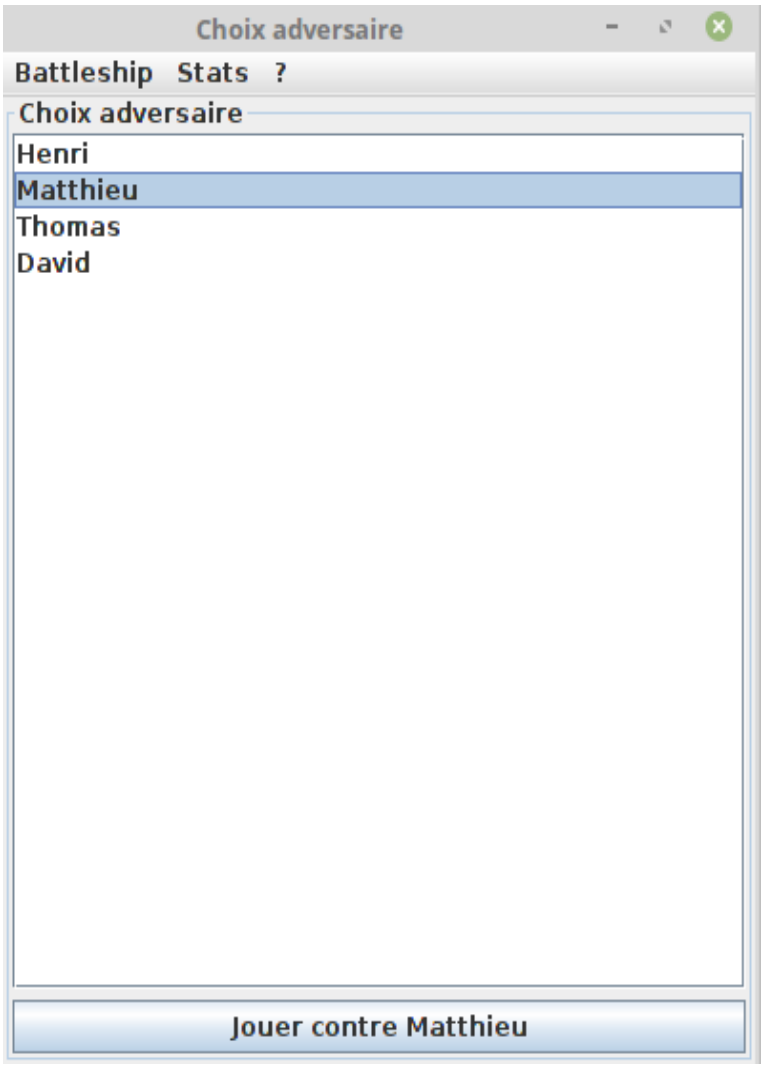

!! ATTENTION !! Si le programme serveur n'a pas été lancé auparavant sur la machine utilisée à cet effet, le programme ne se lancera pas et affichera dans le terminale que le serveur est indisponible. Dans ce cas, il faut d'abord lancé le programme serveur grâce à la commande *java -classpath bin/ fr.insarouen.battleship.BattleshipServer*.
Dans le cas contraire, la fenêtre suivante apparaît.



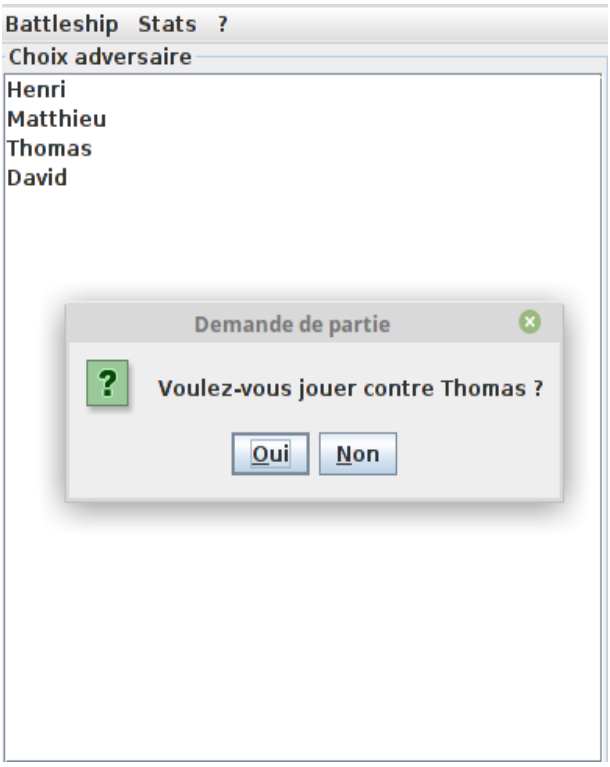
Ensuite, une demande de validation du pseudo est envoyée au serveur. Si le pseudo est déjà utilisé par un joueur, la fnêtre l'affiche.




Sinon on arrive sur la fenêtre de choix du joueur adverse.

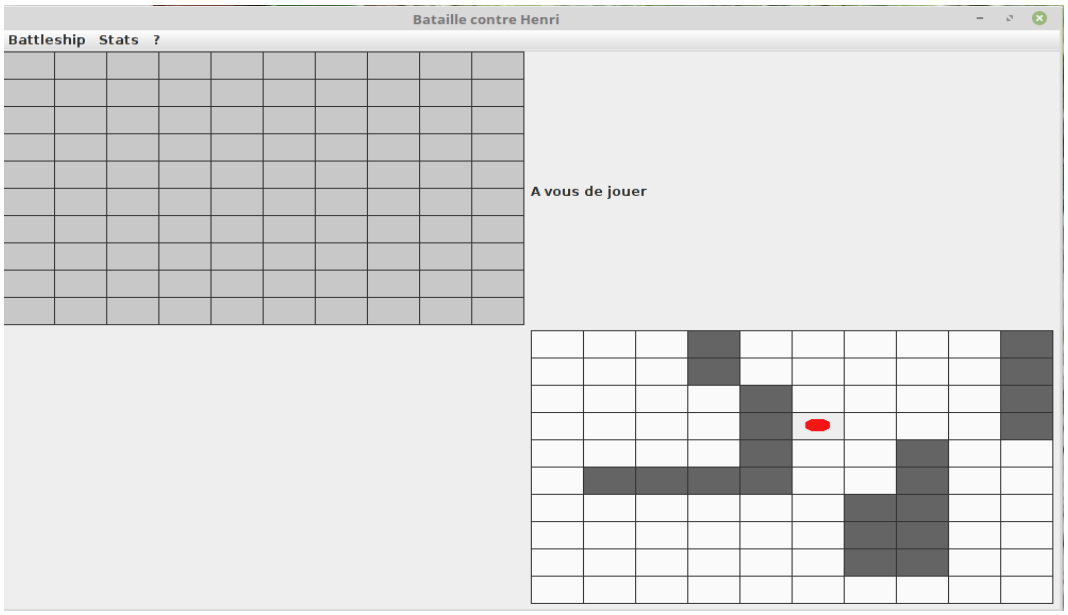


Après avoir cliqué sur le bouton indiquant "Jouer contre *NomDuJoueur*", une demande est envoyée. Celle-ci se fait par l'intermédiaire d'une boîte de dialogue qui apparaît sur l'écran du joueur distant.




Si ce dernier accepte, les 2 joueurs sont redirigés vers la page de jeu. C'est le joueur qui a reçu la demande qui commence. On rappelle que le placement des bateaux dans cette première version se fait aléatoirement. Ensuite les joueurs sont amenés à jouer à tour de rôle jusqu'à ce qu'il y ait un perdant.

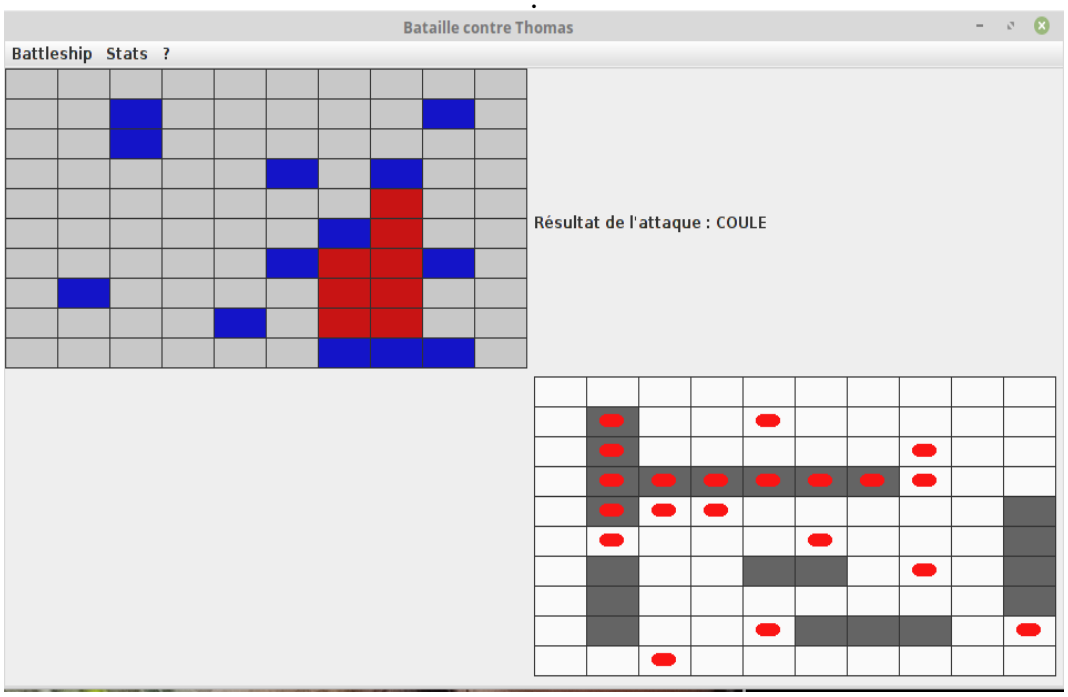
Les images ci-dessous présentent différents états de la partie des deux joueurs.




*Début de partie, côté Thomas*

.

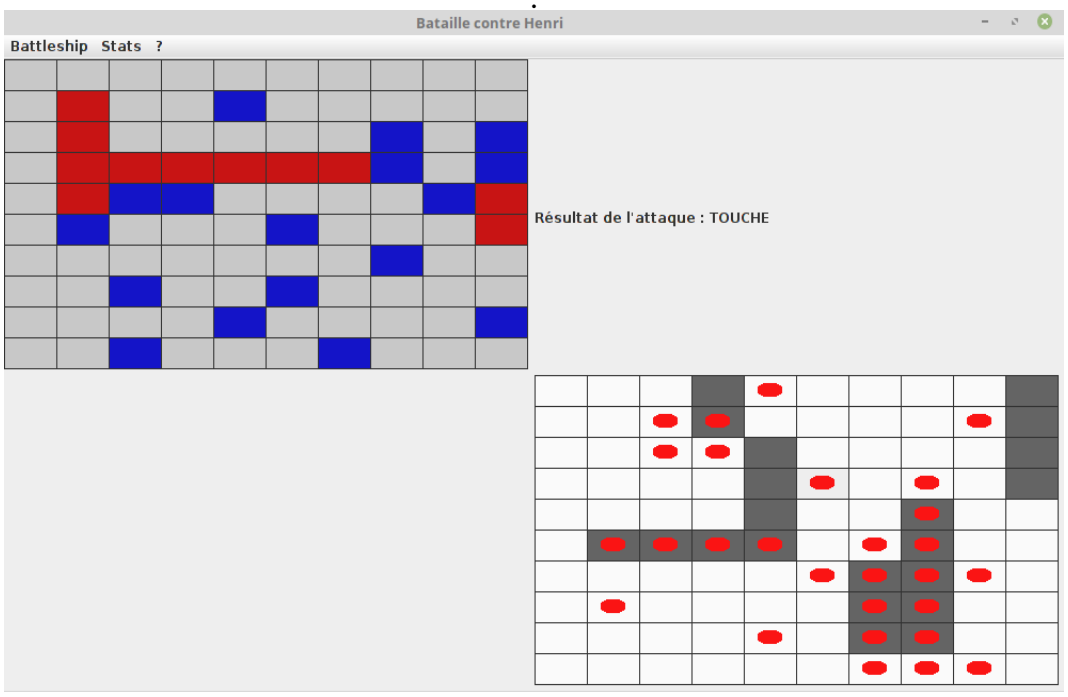
.




*Coulé, côté Henri*

.

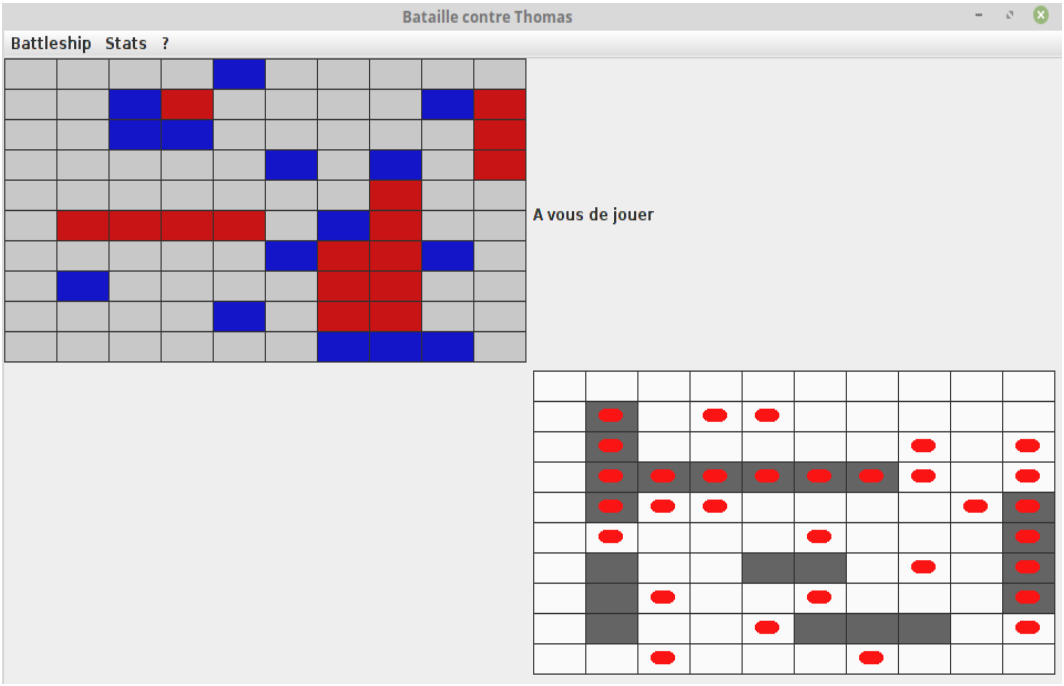
.




*Touché, côté Thomas*

.

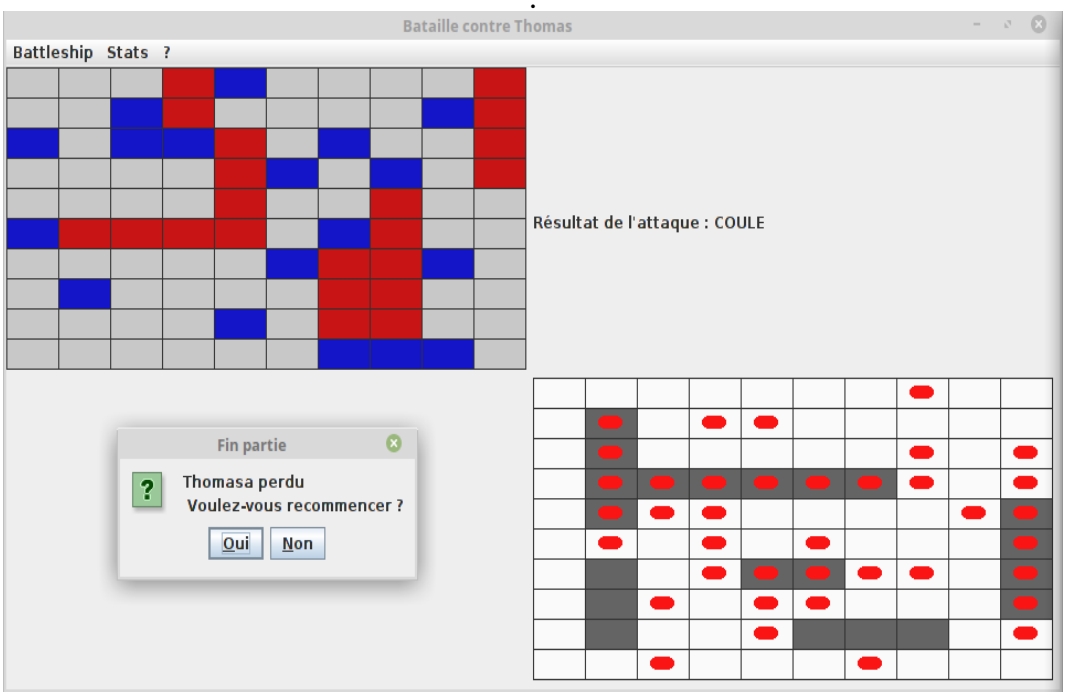
.




*En attente du choix d'une case, côté Henri*

.

.




*Fin partie, côté Henri*

.

.