Projet PC2R 2012 : Bataille Navale Royale

Alexis Deluze & Clément Barbier

18 décembre 2012

Sommaire

1 Serveur **3**
- 1.1 Lancement . . . 3
- 1.2 Introduction . . . 3
- 1.3 Présentation des classes . . . 3
  - 1.3.1 Package serveur . . . 3
  - 1.3.2 Package serveur.grille . . . 3
  - 1.3.3 Package serveur.persistant . . . 4
- 1.4 Structure du serveur . . . 4
  - 1.4.1 Structure globale . . . 4
  - 1.4.2 Connexion au serveur . . . 5
  - 1.4.3 Gestion d'une partie . . . 6
- 1.5 Concurence . . . 6

2 Client **7**
- 2.1 Lancement . . . 7
- 2.2 Introduction . . . 7
- 2.3 Présentation des fichiers . . . 8
- 2.4 Structure du client . . . 8
- 2.5 Concurence . . . 8

3 Extensions **9**
- 3.1 Discussion instantanée . . . 9
- 3.2 Comptes utilisateurs . . . 9
- 3.3 Mode spectateur . . . 9
- 3.4 Serveur HTTP de statistiques . . . 9
- 3.5 Gestion de plusieurs parties . . . 9
- 3.6 Ajout d'éléments au jeu . . . 9

1 Serveur

1.1 Lancement

Le serveur a été développé en **Java** sous **eclipse**. Nous voulions rendre une version `.jar` du serveur en plus de ses sources mais pour une raison inconnue les fonctions qui sont en rapport avec le système de session utilisateurs (`REGISTER/ LOGIN/` et le serveur de statistiques) ne fonctionnent pas correctement (impossible de lire les objets serializés dans le `.jar`).

1.2 Introduction

Le serveur implémente toutes les extensions (voir 3.1 à 3.5) ainsi qu'un élément supplémentaire au jeu (3.6). Les actions que le serveur execute sont lisibles sur le flux standard de sortie.

1.3 Présentation des classes

1.3.1 Package serveur

Ce package comprend les classes nécéssaires à la création du serveur et au déroulement des parties. La classe principale qui contient le `main()` est `Serveur`. Son fontionnement est simple ; elle se met en attente sur le port passé en paramètre (ou 2012 par défaut) et à chaque nouveau client qui se connecte, elle créé un onbjet `ThreadClient` qui va s'occuper de communiquer avec le client (ie : recevoir les commandes que le client envoie et les gérer). La classe `Partie` contient entre autre la liste des joueurs (`ThreadClient`) et s'occupe de gérer et maintenir dans un état cohérent la grille de jeu pour tous les clients. Les classes `Commande`, `Timer` et `ShipPlacement` sont des outils qui permettent respéctivement de parser une commande reçue par un client, attendre que des clients rejoignent une partie avant de la débuter et placer les sous-marins des clients de façon concurente sur la grille.

`Classes du package serveur :`

- `Commande.java`
- `Partie.java`
- `Serveur.java`
- `ShipPlacement.java`
- `ThreadClient.java`
- `Timer.java`

1.3.2 Package serveur.grille

Ce package décrit la grille de jeu et les différents objets qu'elle contient. Ces objets implémentent l'interface `ICoordonnees` qui fournit les méthodes `getX`, `getY`, `setX(int x)`, `setY(char y)` et `setCoords(int x, char y)`. Ces méthodes permettent à la partie de gérer une grille de jeu dans laquelle les joueurs vont placer leurs sous-marins et donner des ordres à leurs drone.

`Classes du package serveur.grille :`

- `Drone.java`
- `Grille.java`
- `ICoordonnees.java`
- `Ship.java`
- `ShipPart.java`

1.3.3 Package serveur.persistant

Ce package contient les données que le serveur peut sauvegarder sur disque, plus précisément les statistiques des joueurs. Un joueur enregistré sur le serveur (avec la commande `REGISTER/user/pass/`) est représenté par un objet `StatsJoueur` qui contient le nombre de victoires et de défaites du joueur. Cette classe est serializée sur le disque dans un fichier `./persistant/user.ser`.

`Classes du package serveur.persistant :`

- `StatsJoueur.java`

1.4 Structure du serveur

1.4.1 Structure globale

Au lancement du `main`, le programme créé deux threads serveur, l'un prévu pour gérer les parties et l'autre pour les statistiques des joueurs. Le serveur de statistiques se contente d'attendre un client, de lui répondre, et de mettre fin aussitôt à la connexion au client. Le serveur de jeu créé un `ThreadClient` pour chaque client qui se connecte. Ainsi après avoir accepté le client, le serveur peut se remettre immédiatement en attente d'une nouvelle connexion. La communication entre le serveur de jeu et le client est donc assurée par le `ThreadClient`.

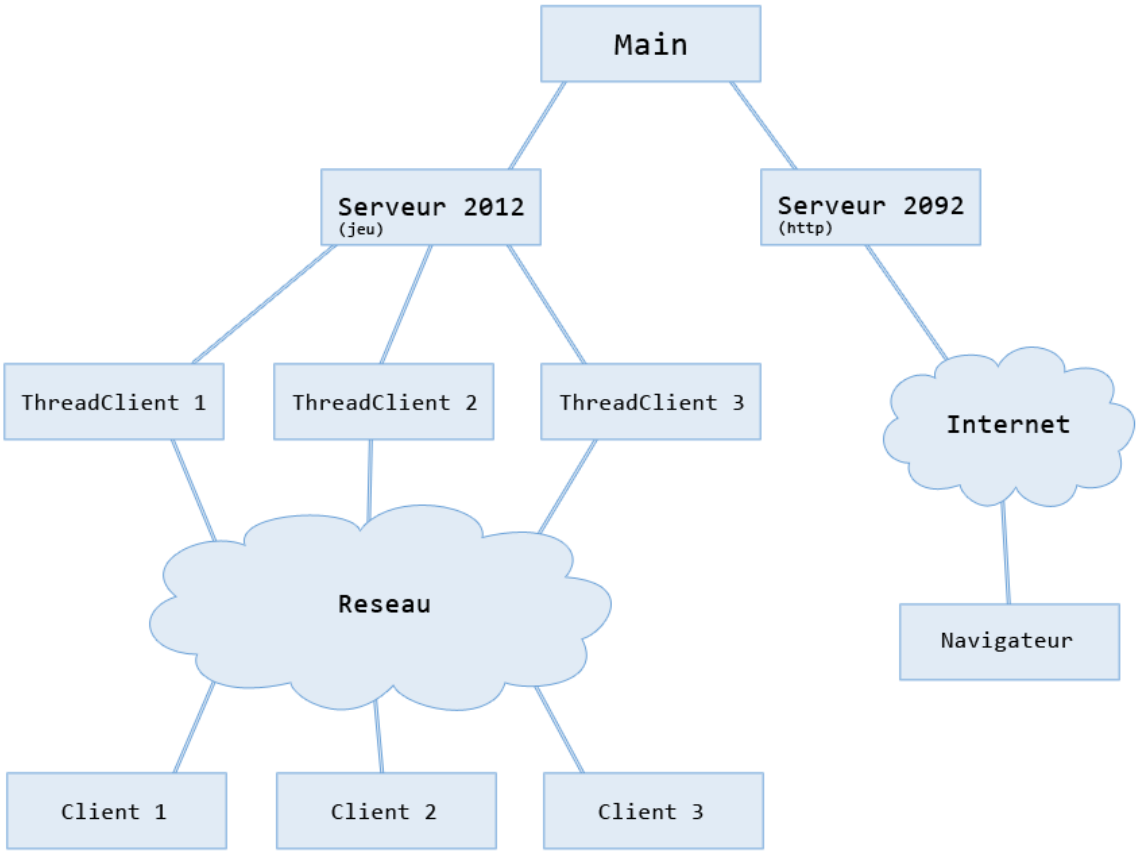

FIGURE 1.1 – Structure globale du serveur

1.4.2 Connexion au serveur

Le diagramme de séquence suivant détaille la connexion de deux clients au serveur de jeu et leur ajout à une nouvelle partie. Le serveur accepte un client, créé son `ThreadClient` associé, puis lance la méthode `gestionPartie()` qui s'occupe de trouver une partie libre à laquelle ajouter le joueur. Si aucune partie n'est libre (c'est le cas sur le diagramme) le serveur créé une nouvelle `Partie` et y ajoute le `ThreadClient` nouvellement créé. A l'ajout d'un second joueur, la partie démarre (voir 1.4.3).

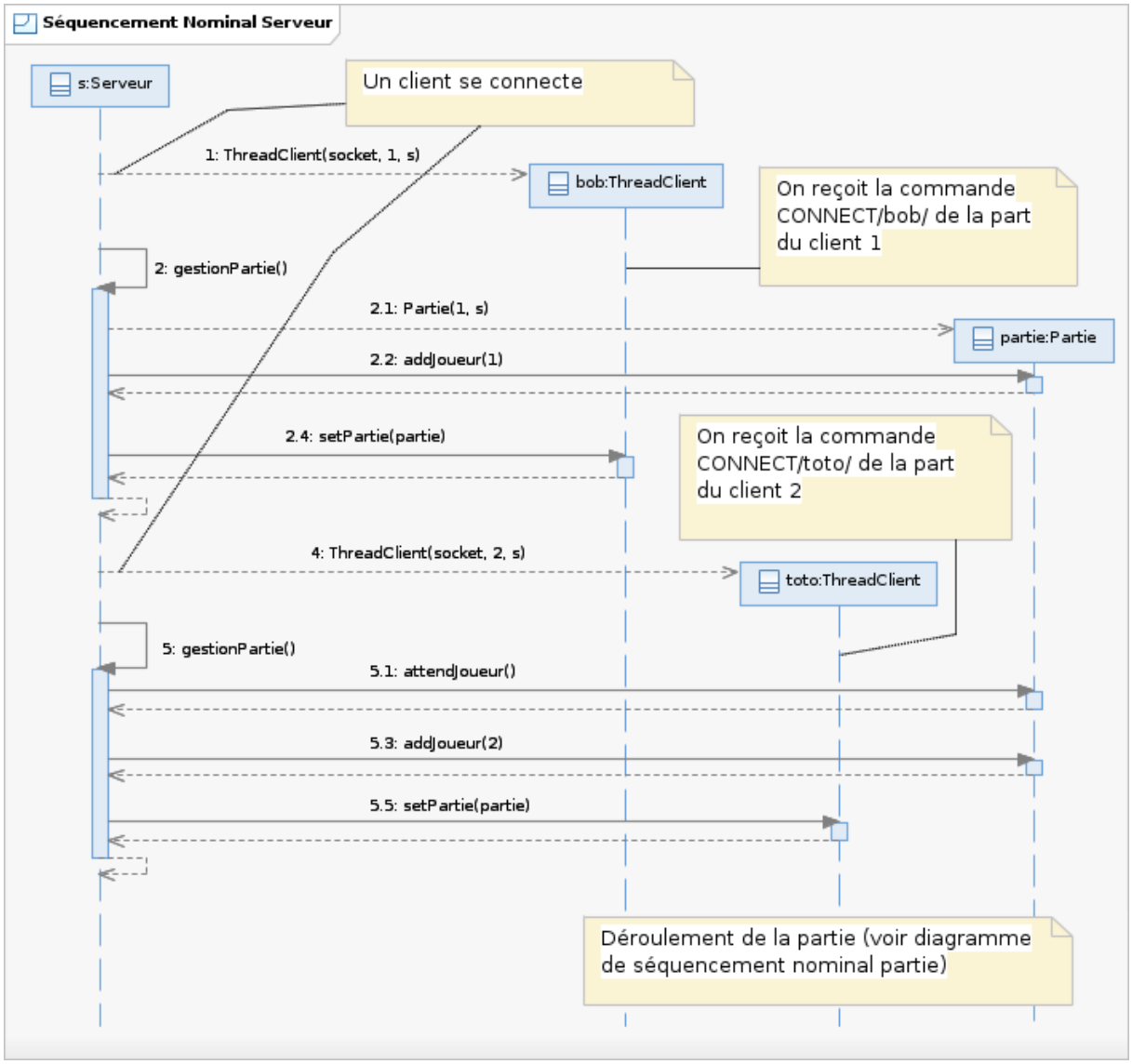

FIGURE 1.2 – Diagramme de séquence : Séquencement nominal serveur

1.4.3 Gestion d'une partie

Le diagramme de séquence suivant explique le déroulement d'une partie entre deux joueurs. Juste après l'ajout d'un second joueur à la partie (voir 1.4.2), celle ci créé un `Timer` qui va attendre 30 secondes la connexion de joueurs supplémentaires à la partie. Le timer se termine quand 4 joueurs sont dans la partie ou que le temps est écoulé ; il notifie alors la `Partie` pour qu'elle lance les préparatifs. Une fois la `Partie` notifiée par le `Timer`, celle ci créé autant de threads `ShipPlacement` que de joueurs dans la partie. Ces threads permettent aux joueurs de placer leurs sous-marins de manière concurente. La vitesse de placement des sous-marins par les joueurs determinera ensuite l'ordre de jeu.

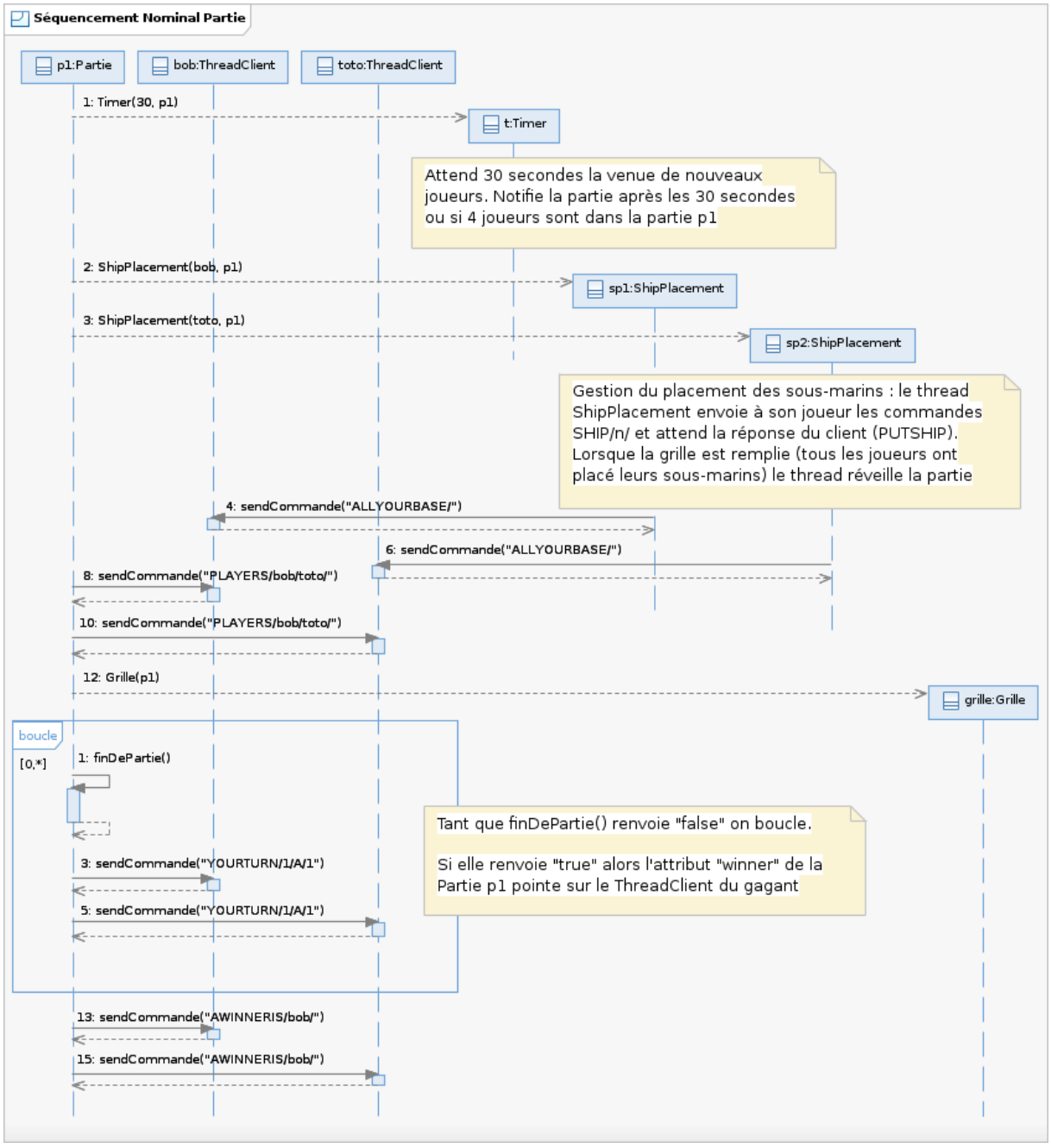

FIGURE 1.3 – Diagramme de séquence : Séquencement nominal partie

1.5 Concurence

La synchronisation entre threads est gérée grace aux méthodes `wait` et `notify` de la classe `Thread` Java. L'accès aux données partagées entre les threads est protégé par des fonctions `synchronized`, ou par l'utilisation du bloc `synchronized`.

2 Client

2.1 Lancement

Lancement du jeu :

- `./client <srv_address> <srv_port>`

Construction et netoyage du binaire exécutable :

- `make all` : construction des .o, liens avec librairies, construction de l'éxécutable.
- `make clean` : netoyage de tous les .o et du binaire.

L'affichage du plateau de jeu se fait dans la fenetre graphique, tous les autres affichages sont en console. La saisie des commandes doit également se faire dans la fenêtre graphique. L'affichage graphique s'appuie sur la SDL (Simple Directmedia Layer), librairie opensource écrite en c et portable (tournant sous Windows, Mac et Linux), simple d'installation, d'utilisation et de compréhension. N.B : SDL possède sa propre implémentation des threads et mutex (SDL_Thread et SDL_mutex) mais ils ne sont pas utilisés dans notre projet. Pour télécharger la librairie : `http://www.libsdl.org/download-1.2.php`. La SDL fait également partie des paquets standards : `libsdl1.2dev`, `libsdl1.2dbg`.

2.2 Introduction

Le client est actuellement capable d'interpréter et gérer toutes les commandes d'une partie, ainsi que celles du mode spectateur. L'aspect graphique étant la dernière chose travaillée, nous n'avons pas eu le temps d'aller jusqu'au bout de cet affichage graphique. Tous les données alphanumériques ne sont pas encore affichées mais elles le sont en console (telles que le nom du jouer, la liste des autres joueurs avec leurs états "Vivant" ou "Mort", le tchat, la saisie des commandes, les reperes sur la grille). La saisie des commandes se fait bien dans la fenêtre graphique mais le texte saisi n'apparait pas à l'écran : uniquement en console quand on presse "Entree". L'affichage de la grille est géré. Voici une capture d'écran avec le `joueur1` à gauche, le `joueur2` au centre et le `spectateur` à droite.

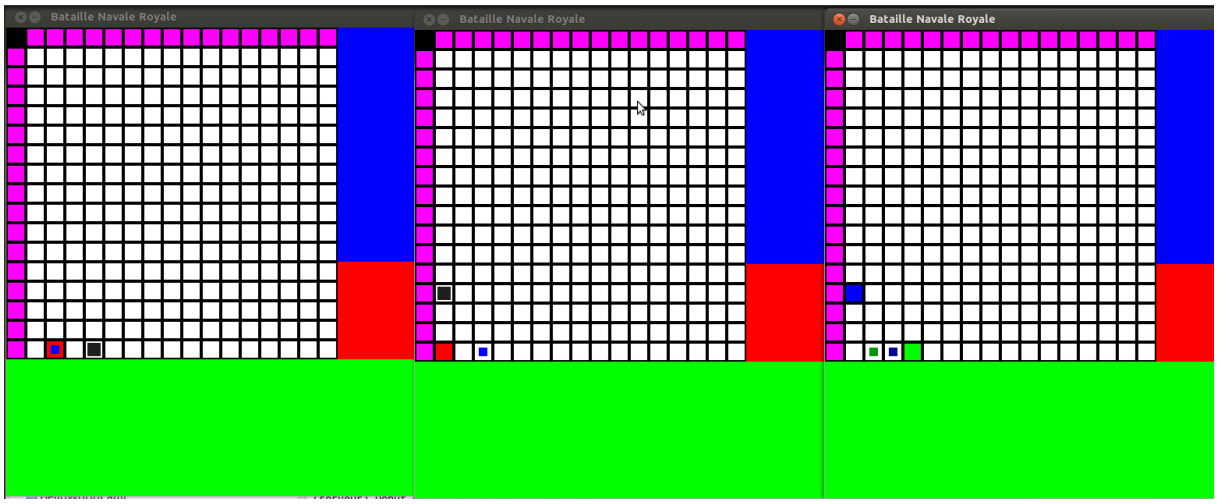

FIGURE 2.1 – Capture d'écran - 2 clients et 1 spéctateur

Découpage des zones non remplies :

- Bleu : Le tchat.
- Rouge : Infos (noms des joueurs, état de santé...)

– Vert : Commande (assistance saisie et affichage buffer d'écriture...)

Légende :

– Sur le client, comme sur le spectateur, une case blanche représente une case vide (sans sous-marin ou/et sans tirs).
– Sur le client, on colore en vert une case sur laquelle on a tiré et touché quelqu'un.
– Sur le client, on colore en rouge une case sur laquelle on a tiré sans toucher personne.
– Sur le spectateur, on colore en rouge une case contenant un ou plusieurs sous-marins ayant été touchés.
– Sur le client, un sous-marin est représenté par un carré gris dans la case ou il est situé, si celui ci est touché le carré sera orange.
– Sur le spectateur, chaque joueur a une couleur associée (j1=vert,j2=bleu,...), ses sous-marins seront représentés par carrés de sa couleur et les drones par de plus petits carrés de cette meme couleur un peu plus claire.
– Sur le client, un drone est représenté par un petit carré bleu.

2.3 Présentation des fichiers

Le client est développé en C sous eclipse et fait appel à la librairie pthread et la librairie `SDL` (qui doit etre installée) qu'il faudra linker lors de la compilation évidemment. Comme les affichages textes ne sont pas opérationnels il n'est pas nécessaire de linker la `SDL_ttf`.

Listing des fichiers :

– board.c et board.h
– command.c et command.h
– game.c et game.h
– player.h
– tchat.c et tchat.h
– utils.c et utils.h
– main.c

2.4 Structure du client

Au lancement, le client va créer deux threads : un thread de lecture, qui lira tous les messages du serveur, et un thread d'affichage/ecriture, qui affiche l'interface de jeu et récupère les saisies claviers pour les transférer au serveur. Le client stockera alors une partie (`game`) qui contiendra une liste de joueurs (`player`), un plateau de jeu (`board/case`), un tchat (`tchat`), un drone (`case`), les sous-marins (tableau de `case`)...etc.

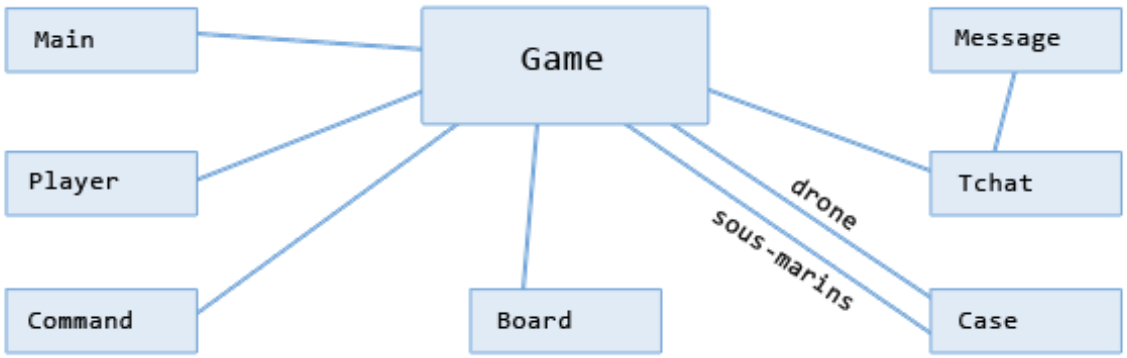

FIGURE 2.2 – Structure du client

2.5 Concurence

Avec deux threads, tous deux producteurs et consommateurs, il nous a fallu protéger les champs modifiés et/ou lus par des mutex. Pour cela nous avons fait appel aux `mutex` des `pthread`. Dans un soucis d'optimisation nous aurions également pu créer une variable pour demander le reaffichage de la partie graphique uniquement apres avoir détecter des modifications de celle ci.

3 Extensions

3.1 Discussion instantanée

Les clients (joueurs ou spectateur) qui sont dans une `Partie` peuvent à tous moment communiquer avec les autres clients en envoyant la commande `TALK/message/` au serveur. Ce dernier se charge ensuite d'envoyer la commande `HEYLISTEN/(pseudo) message/` à tous les clients de la partie.

3.2 Comptes utilisateurs

Un joueur peut réserver son pseudo sur le serveur grâce à la commande `REGISTER/user/pass/`. Le serveur créé alors un fichier `user.ser` (par serialization de la classe `StatsJoueur`) enregistré dans le dossier `./persistant/` qui contient le nom du joueur, son mot de passe et le nombre de parties qu'il a gagné et perdu.

3.3 Mode spectateur

Notre serveur peut gérer plusieurs parties en paralèle (voir 3.5). Le prototol de connexion pour le mode spectateur donné en énoncé n'est donc plus valide, celui ci ne permettant pas de choisir la partie que l'on veut observer. Nous avons donc changé le comportement de la commande `SPECTATOR/` comme suit : si la commande n'a pas d'argument, le serveur envoie au client qui demande la connexion en mode spectateur la commande `GAMESLIST/n1/j1/.../ni/ji/` où `ni` est le numéro d'une partie en cours et `ji` le nombre de joueurs qu'elle contient. Si la commande a un numéro de partie en argument, alors on ajoute le client en mode spectateur à la partie désignée.

3.4 Serveur HTTP de statistiques

Le serveur de statistiques écoute sur le port 2012 de la machine qui héberge le serveur. Il renvoie une page HTML contenant les statistiques de tous les joueurs qui possèdent un compte utilisateur sur le serveur.

3.5 Gestion de plusieurs parties

Nous avons choisi de pouvoir gérer plusieurs parties en paralèle. Lorsqu'un client se connecte pour jouer une partie, si aucune partie n'est libre ou en cours, le serveur créé une nouvelle partie et y ajoute le client.

3.6 Ajout d'éléments au jeu

Nous avons ajouté un élément au jeu : la bombe. Elle ne peut être utilisée qu'une seule fois par partie. Le joueur qui joue son tour de jeu peut envoyer dans sa liste d'actions le caractère 'B' qui représente la bombe. La bombe est effective dans un rayon d'une case autour de la cible (pour un total de 9 cases), toute partie de sous-marins détéctée dans ces 9 cases sera détruite.