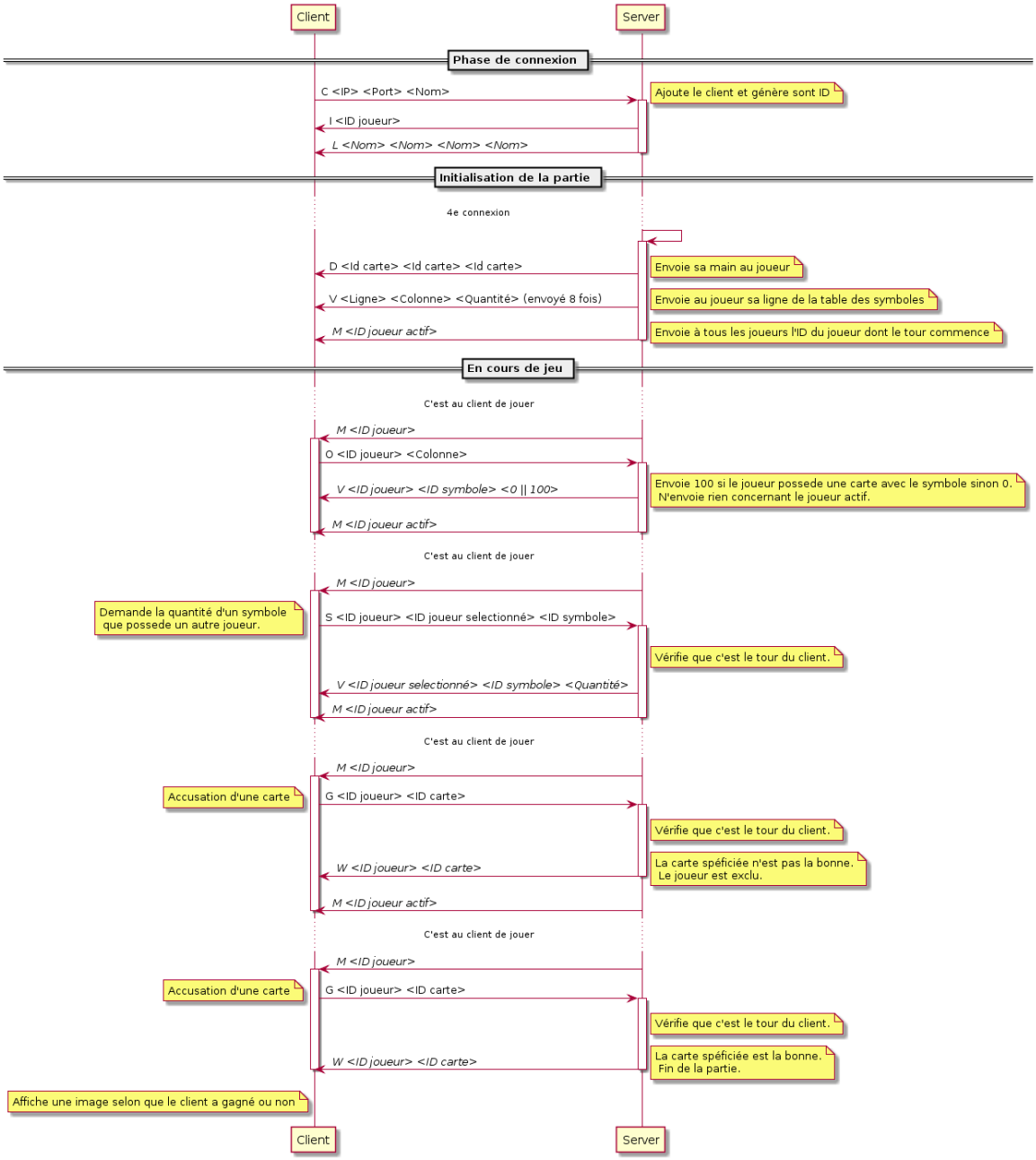
SHERLOCK 13

Florian Cormée et Hugo Duarte

Table des matières

1 Diagramme UML de séquence



M <ID joueur actif> : Commande envoyée à tous les clients connectés.
<ID joueur> : Argument obligatoire d'une commande.

FIG. 1 : Diagramme UML de séquence

2 Guide utilisateur

2.1 Instructions de compilation

La compilation est réalisé par l'intermédiaire de fichiers Makefile. Il suffit d'utiliser la commande suivante pour compiler le client et le serveur.

```
make all
```

Il est possible de ne compiler que le client ou le serveur avec respectivement, les deux commandes suivantes :

```
make client
make server
```

La compilation génère des fichiers objets. La commande suivante permet de les supprimer :

```
make clean
```

Il est possible de supprimer en même temps les exécutables du client et du serveur grâce à la commande suivante :

```
make msproper
```

2.2 Utilisation du client

2.3 Utilisation du serveur

En tant qu'utilisateur, les intéractions avec le serveur, se limite à l'exécuter et à l'arrêter. Pour lancer le serveur, il suffit d'entrer la commande suivante :

```
./server <port>
```

Le port est un argument obligatoire. Le port spécifié doit être disponible. Par exemple, le port 32000.

Afin de ne pas interrompre le chat en fin de partie, le serveur ne s'arrête pas. Pour l'arrêter, sélectionnez sa console et appuyez sur les touches `Ctrl + C`.

3 Fonctionnement général du code

Cette section n'a pas pour ambition d'expliquer en détail le fonctionnement du code du client et du serveur. Néanmoins, elle présente le principe de fonctionnement et l'agencement du code qui a été remanier pour rendre le code plus lisible.

3.1 Fonctionnement du client

3.2 Fonctionnement du serveur

Tout d'abord, le serveur initialise ses variables. Les adresses des clients sont `localhost` et le port est initialisé à une valeur impossible. Le serveur mélange les cartes et rempli sa table de vérité. Ainsi, le coupable est tiré au sors et le programme comptabilise la quantité de chaque symbole que possèdera chacun des joueurs.

Suite à cela, le programme prépare un socket en tant que serveur lié au port passé en argument. Dès lors, le programme attend la réception d'un message.

À la réception d'un message le programme acceptera différentes commandes selon son état. Comme le montre la figure 1, à l'état initiale, le serveur n'accepte que des demandes de connection. Une fois les quatres connections établies, le serveur envoie ses cartes et sa ligne de la table de vérité à chaque joueur. Puis il annonce le début du tour du premier joueur. Enfin, le programme change d'état pour gérer les commandes liées au déroulement du jeu.