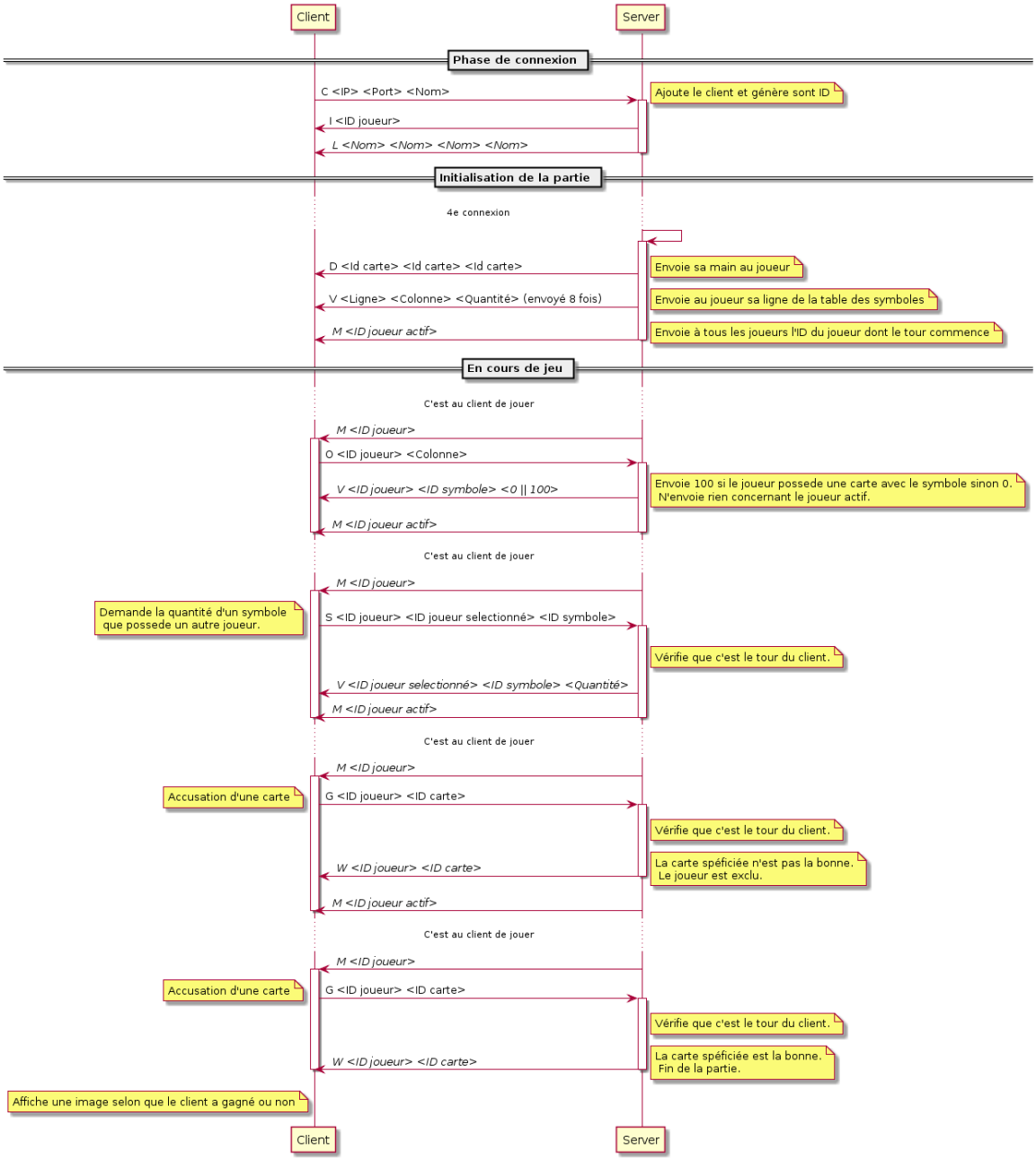
SHERLOCK 13

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

2.2.1 Lancement du client

Le client sert à générer l'interface graphique à l'utilisateur pour lui permettre de jouer. Pour lancer le client, il faut entrer la commande suivante :

```
./sh13.exe <server ip address> <server port> <client ip address> <client port> <player name>
```

Tous les arguments sont obligatoires. L'adresse ip et le port du serveur doivent être ceux utilisés par le serveur lancé. L'adresse ip du client spécifié peut être localhost. Le port client spécifié doit être disponible. Par exemple, les ports 32 001, 32 002, 32 003, etc. Un nom de joueur doit être renseigné.

2.2.2 Connection au serveur

Une fois la commande exécutée, une fenêtre graphique s'ouvrira :

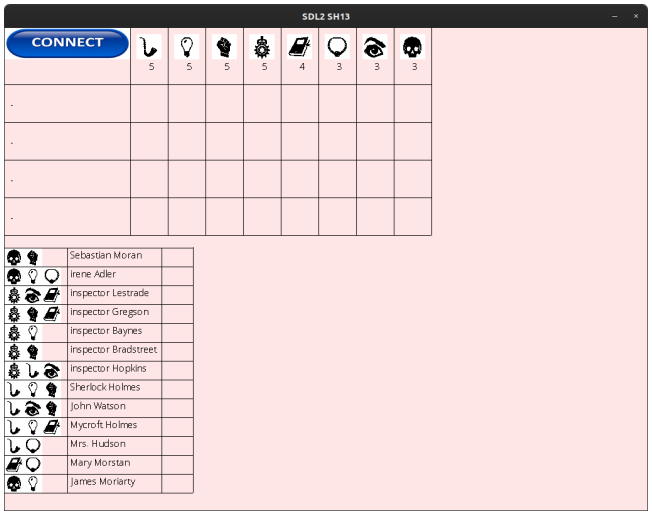


Fig. 2 : Fenêtre graphique avant connection

En cliquant sur le bouton «CONNECT», vous aurez accès aux noms des autres joueurs déjà connectés :

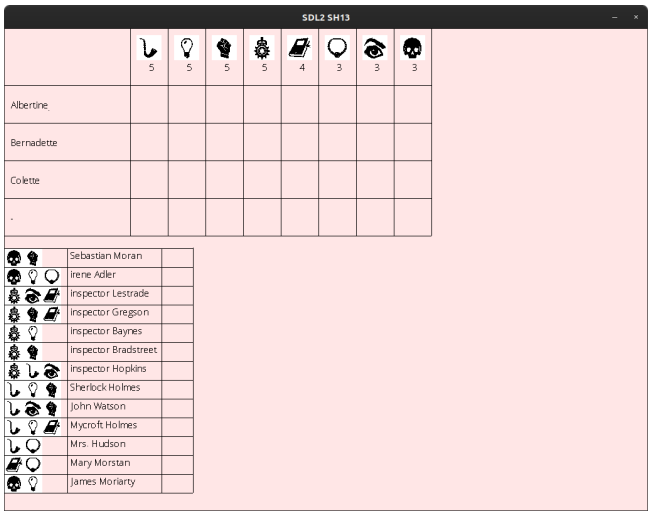


Fig. 3 : Fenêtre graphique après connection

Une fois que le quota de joueurs connectés est atteint, la partie peut commencer !

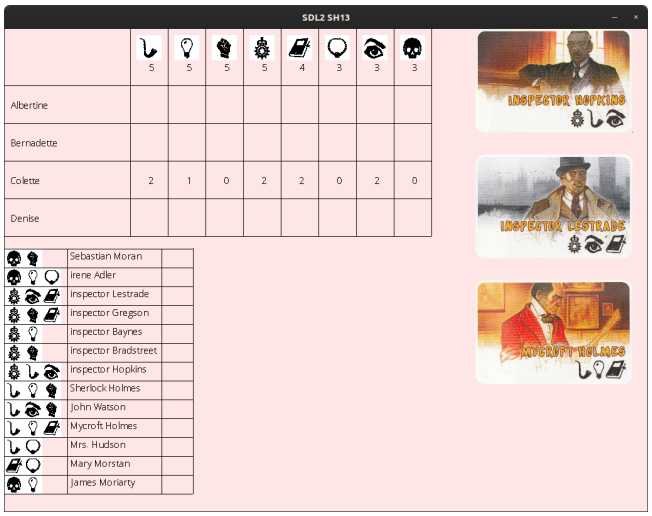


Fig. 4 : Début de partie

2.2.3 Actions disponibles

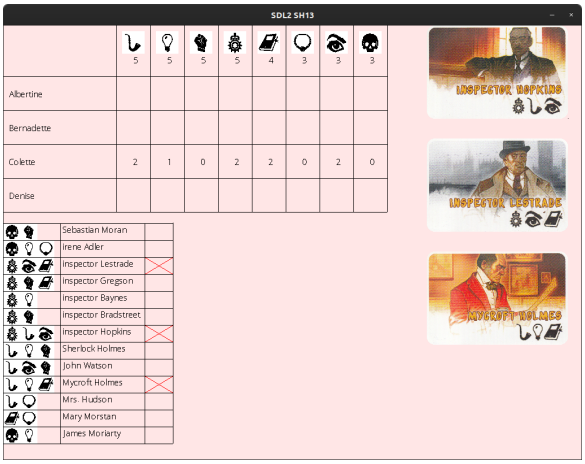
Une fois sur la partie lancée, plusieurs actions sont disponibles :

Utilisation du mémo : (figure 5a) En cliquant dans les cases à côté des noms des différents personnages, vous pouvez indiquer qui vous semble présent dans les mains des différents joueurs.

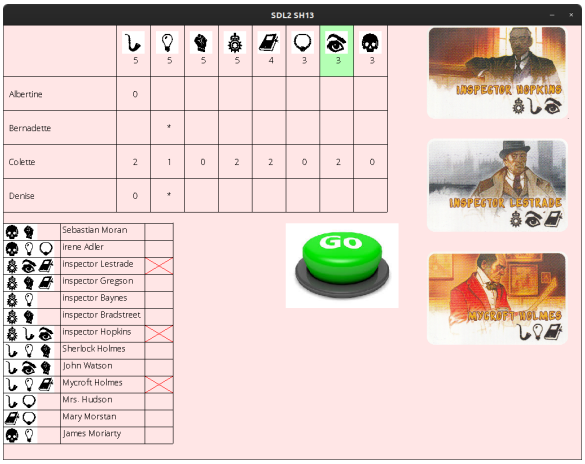
Savoir qui possède un symbole : (figure 5b) En cliquant sur la case du symbole en question et en confirmant son action (en cliquant sur le bouton go), vous pourrez savoir quels joueurs possèdent des personnages liés à ce symbole.

Savoir combien de fois un joueur possède un symbole : (figure 5c) En cliquant sur la case du symbole en question ainsi que la case du joueur souhaité et enfin en confirmant son action (en cliquant sur le bouton go), vous pourrez savoir combien de personnage sont liés à ce symbole dans la main de l'autre joueur.

Accuser un des personnages : (figure 5d) En cliquant sur le nom d'un personage et en confirmant son action (en cliquant sur le bouton go), vous lancez une accusation.



(a) Utilisation du mémo



(b) Savoir qui possède un symbole

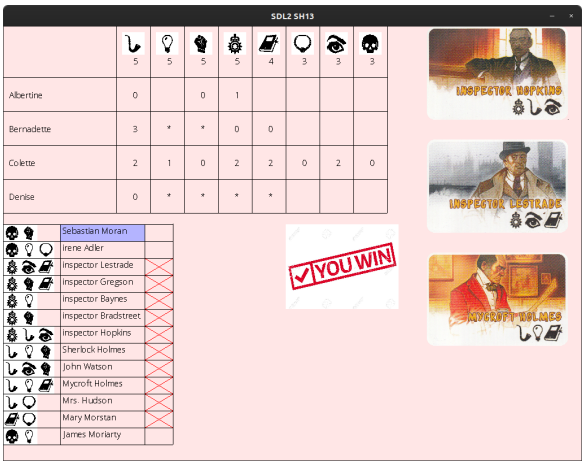
(c) Savoir combien de fois un joueur possède un symbole

(d) Accuser un des personnages

FIG. 5 : Captures des différentes actions

Pour ces trois dernières actions, il est nécessaire de cliquer sur le bouton go (et donc mettre fin à son tour) pour les valider. Pour annuler la sélection, il suffit de cliquer en dehors de tout encart.

Enfin, une fausse accusation entraîne la défaite du joueur tandis que la bonne accusation apporte sa victoire.



(a) Victoire

(b) Défaite

2.2.4 Fermeture du client

Pour mettre fin au programme, il suffit de cliquer sur la croix de fermeture de la fenêtre graphique ou de sélectionner sa console et appuyer sur les touches `Ctrl + C`.

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 remanié pour rendre le code plus lisible.

3.1 Fonctionnement du client

3.1.1 Présentation de l'algorithme

Au lancement, le client initialise ses variables, se connecte au serveur et génère l'interface graphique. On rentre ensuite dans la boucle principale qui est découpé en trois partie. Dans un premier temps, le programme analyse les interactions que l'utilisateur a avec la fenêtre graphique et envoie les commandes associées au serveur. Ensuite, si une requête est reçu du serveur, il exécute les commandes associées. Enfin, il met à jour la fenêtre graphique à l'aide des nouvelles informations reçu. La boucle est quittée quand la fenêtre est fermée et les différents objets graphiques sont détruits.

3.1.2 Répartition du code

Afin de rendre le code source du client plus lisible, nous l'avons découpé en de multiples fichiers et fonctions. Ce découpage est thématique. Nous avons aussi essayé de ne pas avoir des fonctions de plus de 100 lignes ainsi que des fichiers de moins de 500 lignes de code.

Ainsi, les fichiers `cartes.h` et `cartes.c` regroupent les fonctions et les variables globales liées aux cartes.

La communication a été abstraite avec les fichiers `com.h` et `com.c`. Ces fichiers proposent des fonctions d'envoi de message au serveur.

Les fichiers `gui.h` et `gui.c` gèrent les fonctions liées à l'interface graphique ainsi que toutes les variables liées à l'environnement graphique. De cette façon, `sh13.c` ne contient que la boucle principale.

3.2 Fonctionnement du serveur

3.2.1 Présentation de l'algorithme

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. sors et le programme comptabilise la quantité de chaque symbole que possédera 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.

Dans ce second état, le serveur accepte les commandes d'accusation et de questionnement. Quand l'une d'elles est réceptionnée, le programme vérifie que l'expéditeur est bien le joueur actif, dans le cas contraire, un message est affiché dans la console et la requête est refusée. Une fois la requête accomplie, le serveur annonce l'identifiant du joueur suivant. L'opération se répète jusqu'à se qu'un jour parvienne à démasquer le coupable ou que tous les joueurs ont été éliminé, ceci se produit lorsqu'une fausse accusation est lancée.

3.2.2 Répartition du code

Afin de rendre le code source du serveur plus lisible, nous l'avons découpé en de multiples fichiers et fonctions. Se découpage est thématique. Nous avons aussi essayé de ne pas avoir des fonctions de plus de 100 lignes ainsi que des fichiers de moins de 500 lignes de code.

Ainsi, les fichiers `cartes.h` et `cartes.c` regroupes les fonctions et les variables globales liées aux cartes.

La communication a été abstraite avec les fichiers `com.h` et `com.c`. Ces fichiers proposent des fonctions d'envoie de message à un client ou à tous les clients. Ils regroupent aussi les variables globales contenant les adresses des clients.

Les fichiers `msg.h` et `msg.c` ajoute de l'abstraction vis-à-vis du formatage d'un message. De cette façon, `server.c` ne contient que la boucle principale.