TP OS User – Sherlock 13

EISE4

CHEN Pascal

KO Roqyun

Table des matières

1 Présentation

```
inconnu@DESKTOP-5HG6GQK:/mnt/c/Users/inconnu/Documents/sh13/Sherlock13$ ./server 32000
0 Sebastian Moran
1 irene Adler
2 inspector Lestrade
3 inspector Gregson
4 inspector Baynes
5 inspector Bradstreet
6 inspector Hopkins
7 Sherlock Holmes
8 John Watson
9 Mycroft Holmes
10 Mrs. Hudson
11 Mary Morstan
12 James Moriarty
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
7 Sherlock Holmes
2 inspector Lestrade
8 John Watson
11 Mary Morstan
12 James Moriarty
4 inspector Baynes
1 irene Adler
6 inspector Hopkins
9 Mycroft Holmes
5 inspector Bradstreet
3 inspector Gregson
10 Mrs. Hudson
0 Sebastian Moran
02 01 02 01 01 00 02 00
00 02 00 01 01 01 00 01
02 02 00 01 01 01 01 01
01 00 02 02 01 01 00 00
```

Figure 1: Partie Serveur

Pour lancer le serveur, il faut définir le numéro de port du serveur en passant en argument. L'adresse IP du serveur est par défaut « localhost », 127.0.0.1. On peut se connecter à distance au serveur en utilisant l'adresse IP de la carte Ethernet.

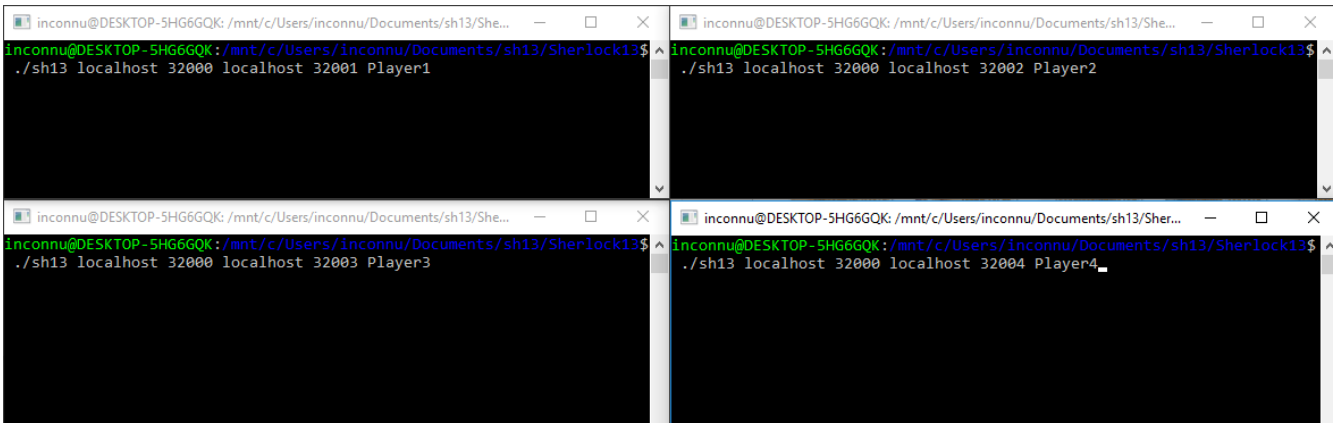

Figure 2: Lancer les clients

Pour lancer les clients, il faut définir le numéro de port des client comme le serveur et les noms des clients. Ils ont également besoin du port et de l'adresse IP du serveur.

Enfin, le jeu est lancé :

Le bouton « GO » en vert est activé / désactivé en fonction du tour des clients. Il y a 12 cartes distribuées aux joueurs parmi 13 cartes au total. La dernière carte est cachée. Les cartes sont représentées par les objets et il faut trouver la carte cachée en devinant les cartes dans les mains des joueurs.

Il y a trois actions disponibles : Accuser le coupable (déclarer la carte cachée), demander de signaler la possession d'un certain objet à tout le monde et demander le nombre d'un objet à un seul joueur.

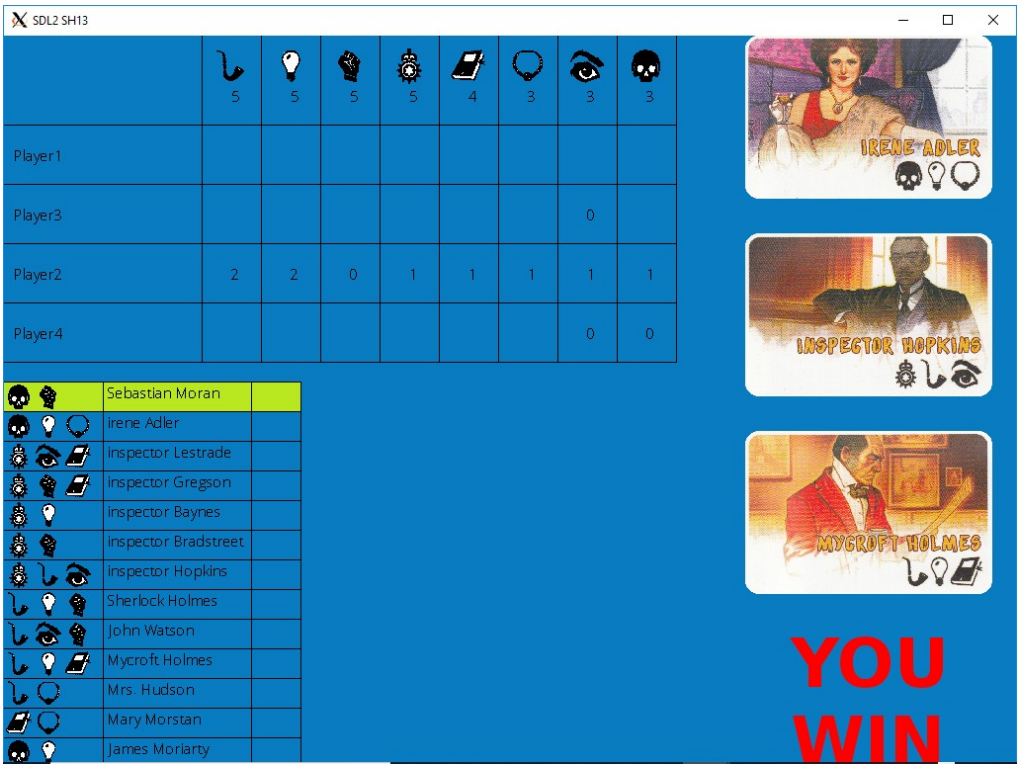

Figure 3: Accusation (Sebastian Moran).

Figure 4: demander de signaler la possession d'un certain objet à tout le monde (oeil).

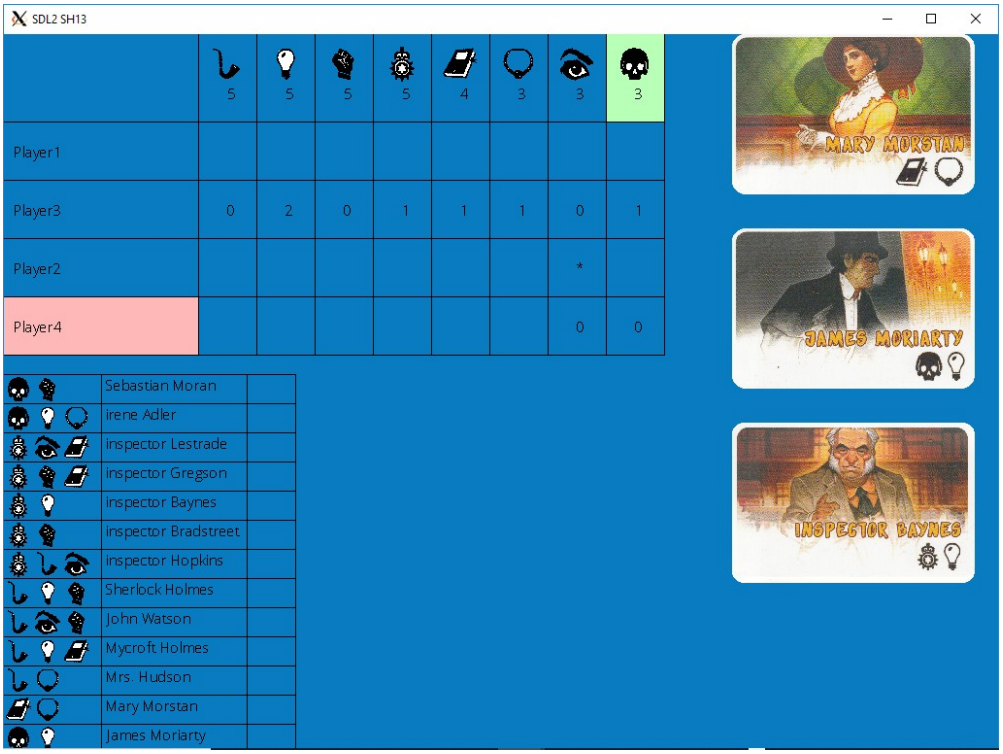

Figure 5: demander le nombre d'un objet à un seul joueur (crâne au joueur 4).

2 Partie – Réseau

2.1 Protocole TCP

1. Socket
   - Cela définie :
     - la famille de protocole utilisé qui est, dans ce cas, AF_INET (l'adresse IP est sur 4 octets).
     - le type de connexion (orienté connexion ou non).
     - le protocole : TCP/IP.
2. Bind
   - Lier le socket et un point de communication défini par une adresse IP et un port.
3. Listen
   - Mettre le socket en attente.
4. while(1) – boucle infinie
5. Accept
   - Accepter la connexion en acceptant l'appel.
6. Read
   - Lire les données envoyées par le côté client.
7. Send
   - Envoyer un acknowledgement par le côté serveur.
8. Close
   - Fermer la connexion en fermant le socket.

2.2 Protocole local

2.2.1 UML

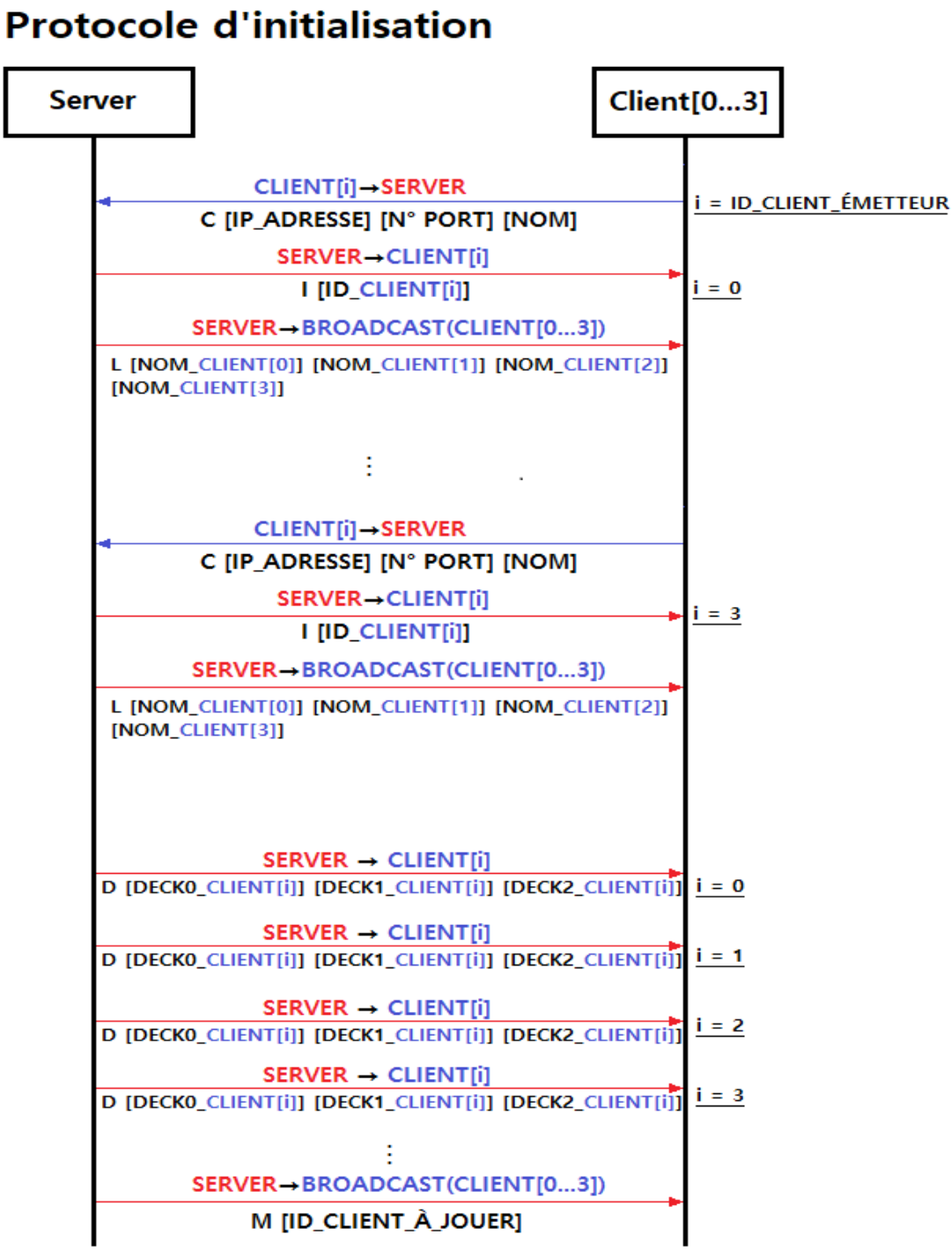

Figure 6: Le protocole d'initialisation : la première communication entre le serveur et les client.

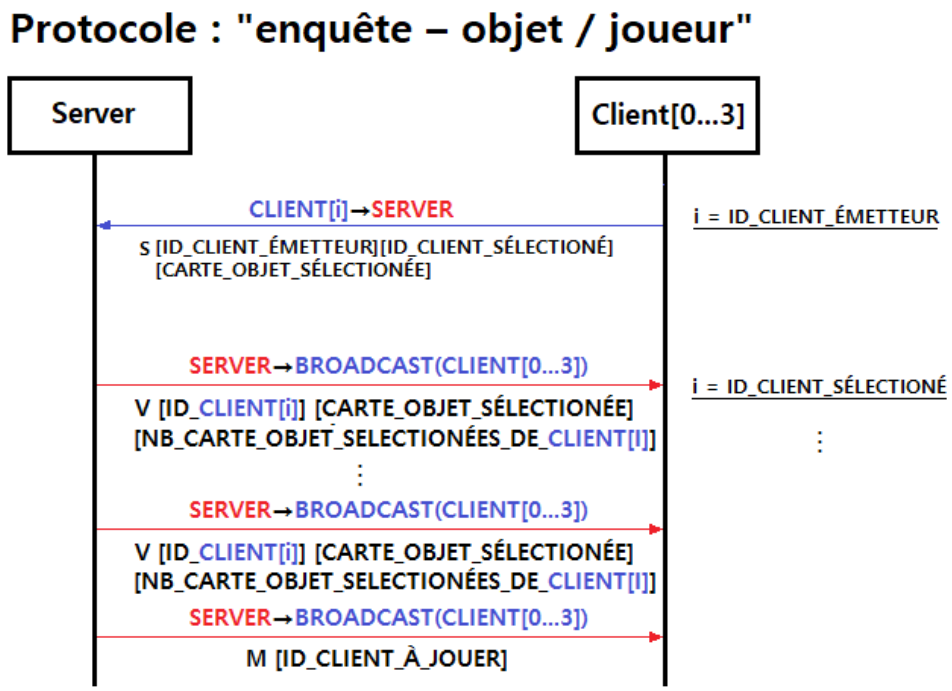

Figure 7: Protocole "enquête – objet / joueur" : signaler le nombre d'un objet en possession.

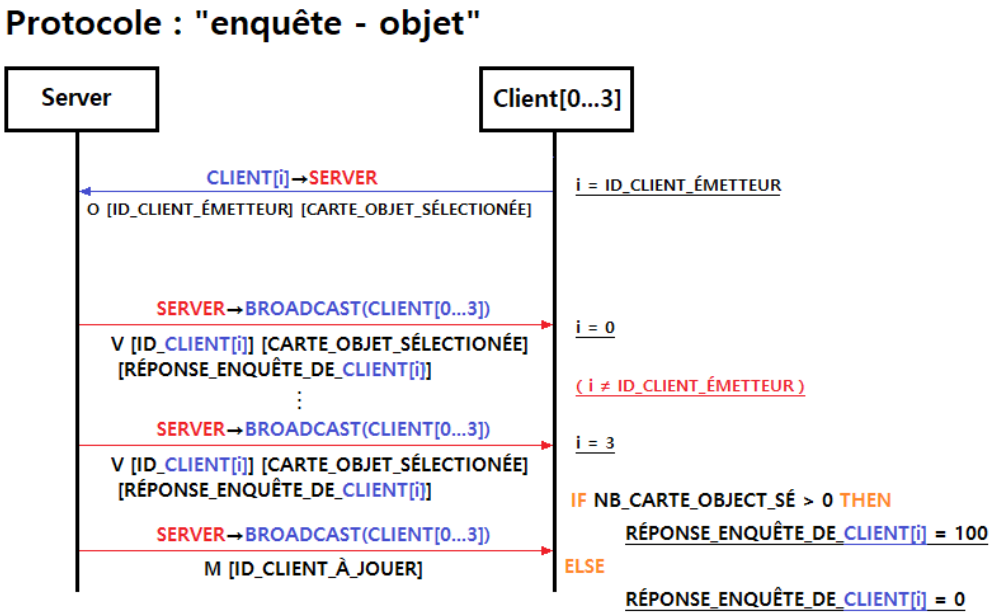

Figure 8: Protocole "enquête - objet" : signaler si les joueurs possèdent l'objet sélectionné

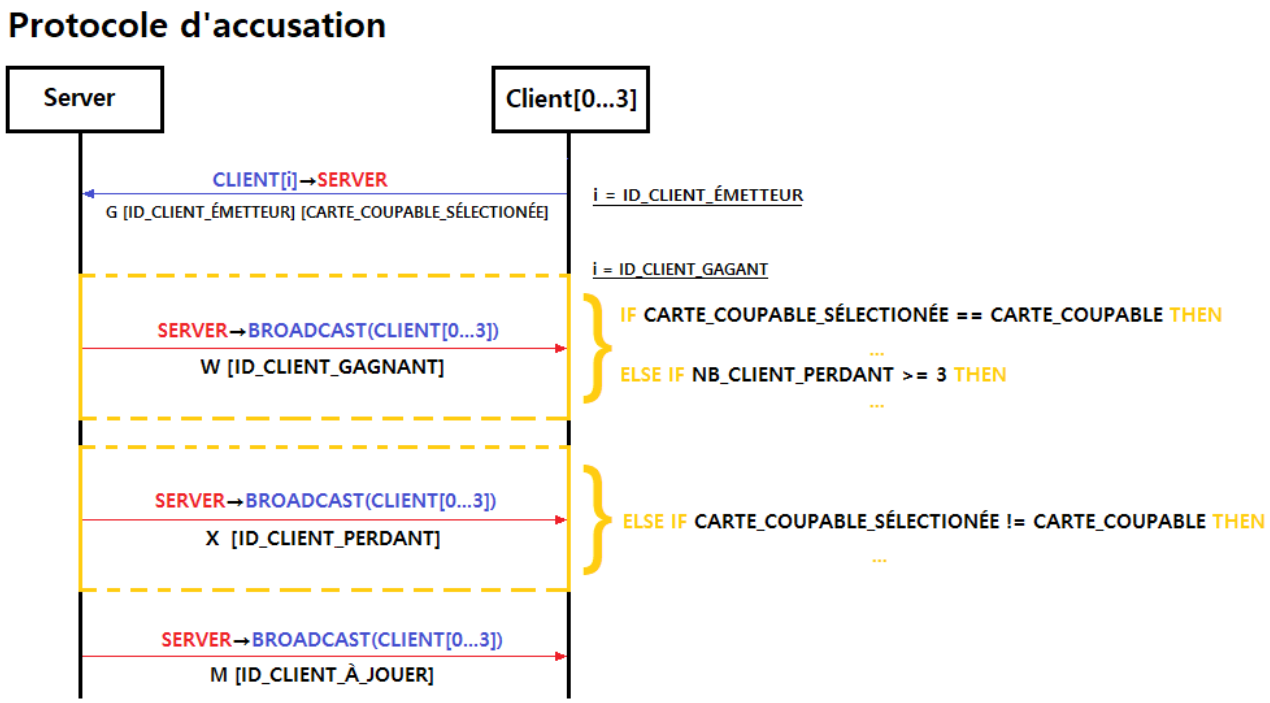

Figure 9: Accusation : déclarer la carte cachée.

2.2.2 Détails

- Finite State Machine : État 0
  - Protocole d'initialisation
    - Le « finite state machine » (machine à état) est à l'état 0.
    - C'est la première communication entre le serveur et les clients. La communication est initialisée par le côté client.
    - Les clients se connectent aux serveur par le processus décrit dans le **Protocole TCP**
    - **Message « C »** : les clients envoient leur adresse IP, leur port et leur nom du joueur. Le serveur les reçoit et enregistre ces informations. Il envoie le **message « I »** vers les clients.
    - **Message « I »** : Le serveur distribue l'identifiant unique de chaque client. Cet identifiant sert à déterminer le gagnant / perdant et l'émetteur d'une enquête parmi les clients. Le serveur diffuse tout de suite le **message « L »** à tout le monde.
    - **Message « L » :** Le serveur envoie les noms de client vers tous les clients. Les clients les reçoivent et ils les affichent dans leur fenêtre.
    - Les 3 messages précédents doivent être envoyé 4 fois pour avoir 4 joueurs. Une fois où les 4 joueurs sont enregistrés, la partie commence et le serveur envoie le **message « D »**

        - **Message « D » :** Le serveur mélange l'ordre des 13 cartes et il distribue 3 cartes à chaque client. Les **messages « D »** contiennent les informations sur ces cartes. Les **messages « D »** sont visés et envoyés en privé. Après envoyer 4 messages, le serveur diffuse tout de suite le **message « M »**.
        - **Message « M » :** Le serveur décide les tours des clients en fonction du temps de la réception du **message « C ».** Le serveur diffuse l'identifiant du client du premier tour. Les clients qui reçoivent ce message active et désactive le bouton « GO ».
        - Le « finite state machine » (machine à état) passe à l'état 1.
- Finite State Machine : État 1
    - Protocole « Enquête – objet / joueur »
        - Le client à jouer sélectionne un objet et un joueur pour demander le nombre d'un objet que le joueur sélectionné possède.
        - **Message « S »** : Le client envoie un message contenant son identifiant, le joueur sélectionné et l'indice de l'objet sélectionné vers le serveur
        - **Message « V » :** C'est la réponse du serveur diffusée à tout le monde. Cette réponse contient la réponse de l'enquête envoyé en **message « S »**. Le **message « V »** contient l'indice de l'objet, le nombre de l'objet du joueur sélectionné et l'identifiant du joueur.
        - **Message « M »** : Le serveur passe le tour au prochain joueur. Le comportement du serveur et des clients restent identique.
    - Protocole « Enquête – objet »
        - Le client à jouer sélectionne un objet pour demander si les joueurs possèdent l'objet.
        - **Message « O »** : Le client envoie un message contenant son identifiant et l'indice de l'objet sélectionné vers le serveur.
        - **Message « V » :** C'est la réponse du serveur diffusée à tout le monde. Cette réponse contient la réponse de l'enquête envoyé en **message « O »**. Au lieu d'envoyer le nombre exacte de l'objet sélectionné, le **message « V »** envoie soit 100 soit 0 pour affirmer la réponse « qui possède l'objet ? ». Le **message « V »** contient également l'indice de l'objet sélectionné et l'identifiant du joueur qui répond à la question. Ce message est envoyé 3 fois (1 questionneur + 3 répondeurs)
        - **Message « M »** : Le serveur passe le tour au prochain joueur. Le comportement du serveur et des clients restent identique.
    - Protocole d'accusation
        - Un joueur déclare le coupable (la carte cachée) et le client envoie cette information au serveur. Si le joueur a correctement deviné, alors il gagne. Sinon, il perd.

- **Message « G »** : Le client envoie un message contenant son identifiant et l'indice d'une carte vers le serveur.
- Le serveur vérifie si le joueur a raison ou pas. Il envoie un des messages ci-dessous.
  - **Message « W » :** le joueur a correctement deviné la carte. Le serveur diffuse l'identifiant du gagnant à tout le monde. Les clients désactivent les boutons « GO » et ils affichent les messages appropriés (« You Win » / « You Lose ») dans leur fenêtre.
  - **Message « L »** : le joueur n'a pas correctement deviné la carte. Le serveur diffuse l'identifiant du perdant à tout le monde. Le client du perdant désactive son bouton « GO » et il affiche le message « You Lose » dans sa fenêtre. S'il y a 3 perdants, le dernier joueur a automatiquement gagné et le serveur envoie le **message « W »**
- **Message « M »** : Dans tous les cas, le serveur passe le tour au prochain joueur. Le comportement du serveur reste identique mais les clients des perdants ne raniment plus les boutons « GO ».

3 Thread

Le thread est implémenté dans la partie client. Le thread permets de tourner les tâches en parallèle. On implémente le thread pour 2 raisons. Premièrement, cela permet de dessiner l'interface graphique et reçoit les messages transmis par les clients. La parallélisation est critique parce que les deux fonctionnement tournent en boucle. Deuxièmement, la réception des message ne doit pas perturber le traitement du message.

Les variables pour utiliser le thread sont créées dans l'entête **sh13.h**. La syntaxe volatile permet de ne pas utiliser le cache pour accéder à la variable synchro. Cela permet un accès plus rapide.

./src/sh13.h, ligne 20 - 21 :

```
pthread_t thread_serveur_tcp_id;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

./src/sh13.h, ligne 45 :

```
volatile int synchro;
```

On constate que :

1. La création du thread est faite dans la fonction main.

   ./src/sh13.c, ligne 46 – 48, la fonction main :

   ```
   printf ("Creation du thread serveur tcp !\n");
   synchro= 0;
   ret = pthread_create ( & thread_serveur_tcp_id, NULL, fn_serveur_tcp, NULL);
   ```

2. Le thread entre dans critique lorsque un message des clients est capté. La fonction attend que la variable synchro soit remise à 0. La variable est remise à 0 si et seulement si le traitement du message reçu est fini. ***cf. 3***

   ./src/sh13.c, ligne 46 – 50, la fonction fn_serveur_tcp :

```
pthread_mutex_lock(&mutex);
synchro=1;
pthread_mutex_unlock(&mutex);

while (synchro);
```

3. Le thread entre dans la section critique pendant le traitement du message. La variable synchro est remise à 0 à la fin.

   ./src/sh13.c, ligne 250 – 312, la fonction main :

```
if (synchro==1) {
        pthread_mutex_lock(&mutex);
        printf("consomme |%s|\n",gbuffer);
        switch (gbuffer[0]) {
                // Message 'I' : le joueur recoit son Id
                case 'I':
                …
                break ;
                case 'X':

                …
                 break ;
        }
        synchro=0;
        pthread_mutex_unlock(&mutex);

}
```

Lorsqu'on entre dans la section critique, cela signifie que cette région du code reste bloquée pour tous les autres threads.

Lorsque un message est reçu, la valeur de la variable synchro devient 1 et puis la fonction ***fn_serveur_tcp*** se fait être en attente par la boucle (***cf. 2***). (La variable synchro ne peut pas être modifiée autrement parce que cette région du code est bloqué.)

Puis, alors que la fonction ***fn_serveur_tcp*** est en attente, le traitement du message est déclenché (***cf. 3***). Cette région devient également bloquée et aucun thread ne peut remettre la variable synchro à 0. Il faut attendre jusqu'à la fin.

Donc les threads sont bien synchronisés dans la manière qu'ils ne se perturbent pas.