

# *RAPPORT PROJET*

# *OS USER*

Sherlock Holmes

*Quentin Césard*



Rapport Projet OS User :

## Table des matières

<b>1. <i>Introduction</i> :</b> .....	<b>2</b>
<b>2. <i>Analyse et implémentation du code</i> :</b> .....	<b>2</b>
<b>A. Serveur :</b> .....	<b>2</b>
a) Structure : .....	2
b) Les fonctions : .....	3
c) Fonction principal (main) : .....	3
<b>B. Client :</b> .....	<b>5</b>
a) Les fonctions : .....	5
b) Fonction principale (main) : .....	5
<b>3. <i>Résultat</i> :</b> .....	<b>7</b>
<b>A. Lancement d'une partie</b> : .....	<b>7</b>
<b>B. Le jeu</b> :.....	<b>12</b>

## **1. Introduction :**

Le programme donné, décrit le jeu Sherlock Holmes. Un jeu de détective qui se joue, dans le programme donné, à 4.

Les règles sont simples, il y a 13 cartes qui représentent des personnages, 12 cartes vont être distribué, la dernière va être utilisée comme coupable ? Sur chaque carte, il y a différents objets (une pipe, un livre ...) qui peuvent être sur différentes cartes.



Exemple d'image de personnage avec le nom et les objets

Le but est simple pour les joueurs, il faut trouver le coupable. Pour cela les joueurs ont le droit d'effectuer 3 actions :

- Demander à tout le monde si quelqu'un a un certain objet.
- Demander à quelqu'un de spécifique, le nombre de fois qu'il a un objet.
- Dénoncer un personnage comme coupable.

Demander au joueur les différents objets qu'ils ont permis de réduire le champ des possibles sur l'identité du coupable. Une fois qu'un joueur pense avoir trouvé le coupable, il peut alors le dénoncer. Si l'accusation portée par le joueur est correcte alors le jeu s'arrête là et le joueur gagne en revanche si le joueur a tort alors le jeu continue mais les autres joueurs pourront enlever un suspect de leurs listes.

Pour réaliser cette application nous auront deux code le server.c (un serveur), le sh13.c (un client) et une multitude d'image, car oui dans ce jeu il y aura bien une interface graphique.

Ce jeu repose sur une architecture client-serveur, le serveur gère les connexions la distribution des cartes et les différents flux arrivant des joueurs.

Le client permet de gérer un client et de lui afficher une interface graphique.

## **2. Analyse et implémentation du code :**

### **A. Serveur :**

#### **a) Structure :**

Dans le code serveur on a une structure, cette dernière permet de définir un client avec différents paramètres, une adresse IP, un port et un nom.

Un tableau permet de stocker les informations des 4 joueurs qui se connecteront.

b) Les fonctions :

Nous avons plusieurs fonctions qui permettent chacune d'effectuer une action particulière :

- La fonction error permet d'afficher un message d'erreur
- La fonction melangerDeck, permet de mélanger de manière aléatoire le deck. Je me suis permis de changer cette fonction car en l'utilisant je me suis rendu compte que les cartes étaient toujours mélangées de la même manière et que le coupable était donc toujours le même. La nouvelle fonction résout ce problème, en utilisant l'heure actuelle, le programme génère un nombre aléatoire.
- La fonction createTable (), permet et de déterminer, pour tous les joueurs, les objets associer à leurs cartes pour pouvoir par la suite utiliser l'information dans le programme.
- La fonction printDeck, permet d'afficher le deck et la matrice tableCartes.
- La fonction printClient, permet d'afficher les différentes informations du client qui sont conservé dans le tableau qui stock les informations des 4 joueurs.
- La fonction findClientByName, permet grâce à un nom donné de retrouver un client.
- La fonction sendMessageToClient, permet d'établir la connexion avec un client en fonction de son IP et de son port de communication et elle permet comme son nom l'indique d'envoyer un message au client avec un write.
- La fonction broadcastMessage, permet d'envoyer un message à l'ensemble des joueur connectés.

c) Fonction principal (main) :

Dans ce programme nous constatons que nous utilisons un protocole de communication TCP (SOCK\_STREAM). Il a été décidé de définir les différents paramètres de cette communication de la manière suivante : utilisation d'une adresse IPV4, la possibilité du serveur d'accepter les connexions de n'importe quelle adresse IP et écoute sur un port donné lors de l'exécution du programme.

Intéressons-nous à la boucle infinie while(1).

Au début de cette boucle nous constatons que le serveur attend et accepte les connexions TCP de plusieurs clients. Il ne les accepte pas de manière simultanée, mais le serveur traite les demandes les unes après les autres. Une fois la demande de connexion accepté, le serveur reçoit un message provenant du client (il pourra lire jusqu'à 255 octets) qui va afficher avec adresse IP du client, qui vient de se connecter.

a) fsmServer à 0 :

- Nombre de joueur inférieur à 4 :

A partir de là nous avons deux possibilités, soit la variable fsmServer est à 0 qui est sa valeur initiale, et donc soit le nombre de joueurs est inférieur à 4 et dans ce cas nous sommes censés avoir un message commençant par un C, qui indique qu'un client c'est connecté. Ce message nous fournira les informations du joueur, et elles seront sauvegardé dans le tableau destiné au différent client.

On va attribuer, à ce nouveau joueur, un numéro (id) qu'on va lui transmettre en envoyant un message avec la fonction sendMessageToClient. Une fois le message contenant l'id est envoyé au bon joueur, on va indiquer tous les membres la nouvelle liste de joueur, en ajoutant le joueur qui vient de se connecter.

- Nombre de joueur égale à 4 :

Si le nombre de joueur est égale à 4, alors dans ce cas, on va devoir envoyer les cartes pour chacun des joueurs et la ligne correspondante dans tableCarte.

Pour commencer nous débutons notre message par un D (deck) pour pouvoir ensuite le décoder du côté client, puis on ajoute et envoi les 3 premières cartes pour le premier joueur, on enverra les 3 suivantes pour le second ect. Pour envoyer toute la ligne correspondante dans la tableCarte, on effectue une boucle for qui va faire chaque élément et l'envoyer à chaque fois sans oublier d'ajouter un V au départ de l'envoi pour le décodage.

Une fois toutes les cartes envoyées avec les lignes de la tableCarte envoyé, on passe la variable fsmServer à 1, ce qui nous permet d'entrer dans la partie de gestion des messages pour le jeu.

b) fsmServer à 1 :

Nous allons maintenant traiter les différentes actions que les clients peuvent effectuer.

Pour différentier ces action la lettre de départ du message reçu changera :

- Dans le cas où le message commencerait par un « G », alors cela signifiera qu'un joueur accuse quelqu'un. Nous vérifions alors si c'est la bonne réponse si c'est le cas nous affichons une phrase dans le terminal et nous envoyons une phrase aux joueurs qui indique que le joueur a gagné, dans le cas inverse, on affiche un message dans le terminal du serveur pour indiquer que le joueur n'a pas gagné et on passe au joueur suivant en
- Quand le message commence par un « O » alors l'un des joueurs veut des informations sur un objet, on va donc parcourir les cartes des joueurs pour trouver les cartes contenant cet objet, si certains sont trouvé alors elles

sont renvoyées à tout le monde, sinon on envoi 0 à tout le monde en faisant attention à ne pas oublier le V au début du message pour le décodage côté client.

- Dans le dernier cas si le message commence par un S, alors on peut poser une question cibler sur une personne.

## B. Client :

### a) Les fonctions :

Dans ce code client, nous avons deux fonctions qui permettent d'effectuer chacune une tache :

- La fonction fn\_serveur\_tcp, dans cette fonction un protocole TCP est créer comme du coté serveur, créations du socket TCP, paramétrage du protocole et écoute sur le port fourni.  
Dans cette fonction on constate une boucle infinie, elle permet d'accepter la requête du client (dans notre jeu ce sera le serveur principale), puis il lit le message reçu et active la variable synchro à 1 pour indiquer qu'un message est arrivé.
- La fonction sendMessageToServer, permet comme son nom l'indique d'envoyer un message au serveur. Dans cette fonction, on créer un nouveau socket, on récupère l'adresse IP du serveur, on paramètre le message, on établit enfin la liaison avec un connect et on termine par envoyé le message et fermer la connexion.

### b) Fonction principale (main) :

Dans les premières lignes du main du client, on nous indique les informations à entrer à l'exécution du code, ces informations sont ensuite stockées dans des variables.

### c) Initialisation :

On débute la fonction principale par initialiser la partie graphique en créant une fenêtre de 1024x768. Par la suite, on charge les différentes images qui se trouve dans notre dossier.

Une fois toutes les images chargées, on met tous les paramètres aux états de base :

- On définit les noms des joueurs avec des « - » car ils ne sont pas encore connectés.
- On met les variables qui permettre d'effectuer des actions à -1 (joueur qu'on cible, objet qu'on a sélectionné et coupable).
- On initialise aussi les cartes des joueurs avec des valeurs de base -1.
- On initialise la supposition sur le coupable à -1

- Et on définit la connaissance que les joueurs ont sur les cartes des autres à -1.

On termine par désactiver le bouton go car tous les participants ne sont pas connectés et on active le bouton de connexion.

Ensuite nous avons une partie sur la création des textures, cela permettra de rendre les différents boutons présent (le bouton connect, le bouton go ...) interactifs.

Nous constatons la création d'un thread. Ce dernier, effectuera le code présent dans la fonction fn\_serveur\_tcp, que je vous ai expliqué au départ.

Cela permettra donc de capter les messages transmis par le serveur. Je rappelle que dans la fonction les messages sont stockés dans un buffer et que dès qu'un message est arrivé la variable synchro passe à 1.

d) Boucle infinie tant que la fenêtre est ouverte :

A partir de maintenant, nous entrons dans une boucle infinie qui ne peut pas se terminer tant que l'utilisateur ne ferme pas la fenêtre du jeu. En cas de fermeture on stop le programme.

Si un événement utilisateur est détecté, un clic sur la souris ou sur le clavier, alors on cherche à savoir sur quoi l'utilisateur à cliquer, pour savoir quelle action on doit effectuer.

Pour déterminer cela, on récupère la position de la souris et on définit différents états pour les différentes variables en fonctions de la localisation de la souris :

- Mx < 200 et my < 50 et que le bouton connecte passe à 1, cela signifie qu'un des joueurs veut se connecter, on charge alors un message contenant les informations du client, ce message commencera par un C pour indiquer la connexion du joueur. Nous terminons par envoyer le message au serveur grâce à la fonction sendMessageToServer.
- Si la souris se situe entre 0 et 200 pour mx et 90 et 330 pour my, on calcul sur quelle personne le joueur à cliquer en faisant  $\frac{my-90}{60}$ .
- Si la souris se situe entre 200 et 680 pour mx et 0 et 90 pour my, on calcul sur quelle objet le joueur à cliquer en faisant  $\frac{mx-200}{60}$ .
- Si la souris se situe entre 100 et 250 pour mx et 350 et 740 pour my, on calcul sur quelle coupable le joueur à cliquer en faisant  $\frac{my-350}{30}$ .
- Si la souris se situe entre 250 et 300 pour mx et 350 et 740 pour my, le joueur clic sur une colonne et on inverse l'état de la croix
- Dans le dernier cas, si la souris se situe entre 500 et 700 pour mx et 350 et 450 pour my et que le bouton go passe à 1, cela signifie que le joueur à

fais son choix d'action. On affiche donc dans le terminal les informations qu'a sélectionné le joueur (les informations qui ont été calculé avant, joueur sélectionné, objet sélectionné et suspect sélectionné). En fonction de l'état des variables (objetSel, joueurSel, guiltSel, qui ont été calculé avant) qui définissent ce que le joueur a sélectionné, on prépare un message avec les informations de ces variables et une lettre spécifique au départ du message qui définira l'action à effectuer :

- Message commençant par un G, cela signifie que le joueur accuse un autre joueur.
- Message commençant par un O, cela signifie que le joueur interroge les autres joueurs sur leurs jeux.
- Message commençant par un S, cela signifie que le joueur interroge un joueur particulier sur son jeu.

Il nous reste juste à envoyer le message au serveur qui le décodera.

Dans la suite du code on vérifie si la variable synchro est à 1 (si un message a été envoyé par le serveur). Si c'est le cas, on identifie la première lettre du buffeur :

- Si c'est un I on reçoit l'id du joueur, on associe donc cette valeur à la variable qui contiendra l'id.
- Si c'est un L on reçoit la liste des joueurs, on change alors les noms de base qu'on leurs avaient donnés (« - »), par leur vrai nom.
- Si c'est un D on reçoit les 3 cartes du joueur, on les ajoute donc à notre deck.
- Si c'est un M le joueur reçoit l'id du joueur actuel. Si l'id est le sien alors le bouton go apparaît pour que le joueur puisse faire et valider son choix sinon rien ne se passe.
- Et si c'est un V, le joueur reçoit une valeur de carte, on met donc à jour le tableau en fonction de la valeur reçue.

e) Affichage :

Dans le reste du code, nous avons toute la partie pour afficher le jeu.

### **3. Résultat :**

#### **A. Lancement d'une partie :**

Commençons par lancer une partie.

Pour le lancement toutes les explications sont dans le README dans le dossier du projet.

Commençons par le serveur, pour lancer le serveur nous devons donné le port de communication du serveur :

```
(base) quentincesard@macbookair Projet_OSUSER_Quentin_CESARD % make leserveur
./serveur 6000
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
4 inspector Baynes
1 Irene Adler
10 Mrs. Hudson
3 inspector Gregson
11 Mary Morstan
5 inspector Bradstreet
9 Mycroft Holmes
7 Sherlock Holmes
8 John Watson
6 inspector Hopkins
0 Sebastian Moran
12 James Moriarty
2 inspector Lestrade
01 02 00 01 00 02 00 01
00 00 02 02 02 01 00 00
03 02 02 00 01 00 01 00
01 01 01 01 00 00 01 02
```

On constate dans un premier temps que les cartes ont été mélanger lors du lancement du serveur.

Maintenant faisons-en sort de rajouter des joueurs.

Pour lancer un joueur nous devons données des arguments lors de l'exécution :

- L'adresse IP du serveur.
- Le port de communication du serveur.
- L'adresse IP du client.
- Le port de communication du client.
- Et enfin le nom du joueur.

```
./client 127.0.0.1 6000 127.0.0.1 3000 Joueur1 & \
        ./client 127.0.0.1 6000 127.0.0.1 3001 Joueur2 & \
        ./client 127.0.0.1 6000 127.0.0.1 3002 Joueur3 & \
        ./client 127.0.0.1 6000 127.0.0.1 3003 Joueur4
Sans=0x1310a3400
Sans=0x14a034a00
Sans=0x12805ca00
Sans=0x12f0d3000
Creation du thread serveur tcp !
```

On constate sur le screen ci-dessous que 4 joueurs ont été créés, 4 fenêtres ont aussi été lancé comme celle-ci :



Sur ces fenêtres nous avons bien le boutons connect qui va permettre au client de se connecter au serveur, cliquons sur l'un d'entre eux.

```
Received packet from 127.0.0.1:49306
Data: [C 127.0.0.1 3000 Joueur1
]

COM=C ipAddress=127.0.0.1 port=3000 name=Joueur1
0: 127.0.0.1 03000 Joueur1
id=0
```

Du coté serveur nous avons eu du mouvement. On constate que le premier joueur c'est connecté, on nous retourne donc son adresse IP son port et son nom. On lui assigne donc un identifiant unique, étant le premier à se connecter celui-ci sera 0.

Du côté client il y a aussi eu du mouvement, suite à la connexion du joueur 3 sont identifiant dans la partie à bien été reçu (0) encadré en jaune sur le screen ci-dessous. On voit aussi que le serveur a renvoyé la liste des joueurs mise à jour en rouge ci-dessous.

```
consomme | I 0
|
consomme | L Joueur1 - - -
```

Ajoutons le second joueur.

Nous constatons que le serveur reçoit la demande et attribue l'id 1 au second joueur.

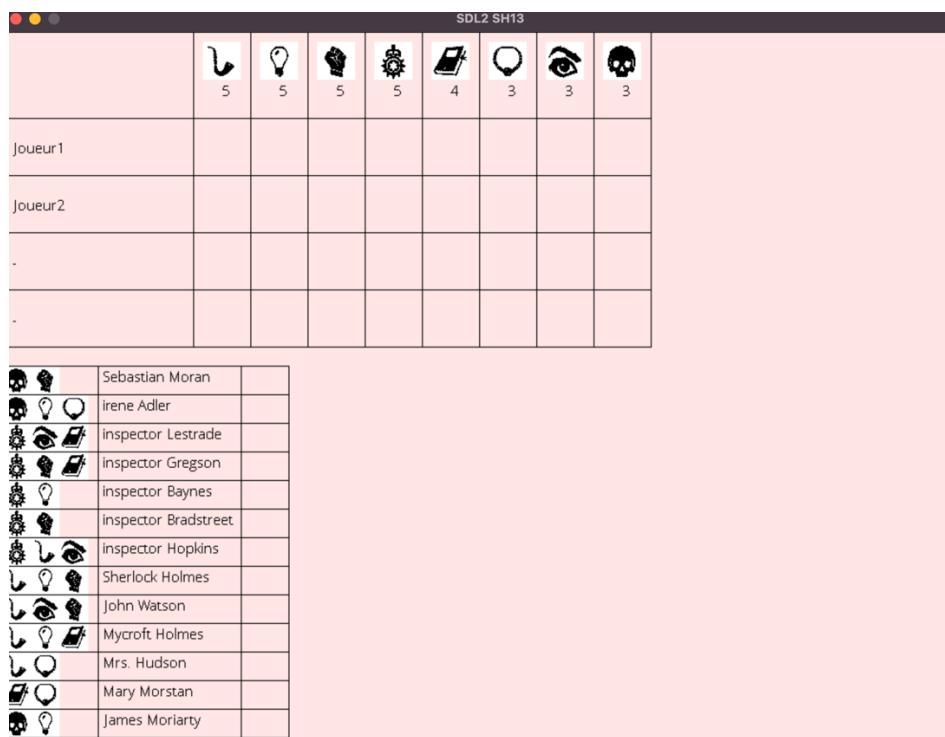
```
Received packet from 127.0.0.1:49309
Data: [C 127.0.0.1 3001 Joueur2
]

COM=C ipAddress=127.0.0.1 port=3001 name=Joueur2
0: 127.0.0.1 03000 Joueur1
1: 127.0.0.1 03001 Joueur2
id=1
```

Du coté des clients, on voit que le joueur 2 à bien reçu sont id et que la liste des joueurs à bien été mise à jour et envoyé au 2 joueur connecté :

The screenshot shows two terminal windows. The top window is titled "Projet\_OSUSER\_Quentin\_CESARD — client 127.0.0.1 6000 127.0.0.1 3000 Joueur1 — 121x32". It displays log messages for player 1's connection, including thread creation and memory consumption details. The bottom window is titled "Projet\_OSUSER\_Quentin\_CESARD — client 127.0.0.1 6000 127.0.0.1 3001 Joueur2 — 1". It shows similar log messages for player 2's connection, with specific memory addresses highlighted in red boxes around "consomme |I 0", "consomme |L Joueur1 Joueur2 --", and "consomme |L Joueur1 Joueur2 --".

Dans la partie graphique c'est le même constat le tableau des joueurs se mets à jour :



Une fois que tous les joueurs se sont connectés, les cartes sont distribuées à tous.

Par exemple le joueur 4 reçoit ses cartes :

```

consomme |I 3
|
consomme |L Joueur1 Joueur2 Joueur3 Joueur4
|
consomme |D 6 0 12
|
consomme |V 0 0 1
|
consomme |V 0 1 1
|
consomme |V 0 2 1
|
consomme |V 0 3 1
|
consomme |V 0 4 0
|
consomme |V 0 5 0
|
consomme |V 0 6 1
|

```

On constate que le joueur 4 reçoit l'ID 3 car c'est le dernier à cette connecté, il reçoit la liste des joueurs dans l'ordre d'arrivé (lui est bien en dernier), puis enfin il reçoit ses cartes, on constate qu'il a eu les carte 6, 0 et 12. Pour connaître les cartes et donc les personnages associé qu'il a eu, on a dans le code un tableau avec le nom des cartes :

```

char *nomcartes[]=
{"Sebastian Moran", "irene Adler", "inspector Lestrade",
 "inspector Gregson", "inspector Baynes", "inspector Bradstreet",
 "inspector Hopkins", "Sherlock Holmes", "John Watson", "Mycroft Holmes",
 "Mrs. Hudson", "Mary Morstan", "James Moriarty"};

```

Si on suit ce tableau, le joueur 4 devrait avoir Sebastian MORGAN, Inspecteur HOPKINS et James MORIATY, ce qui est bien le cas :

joueur1	5	5	5	5	4	3	3	3
joueur2	1	1	1	1	0	0	1	2
joueur3								
joueur4								

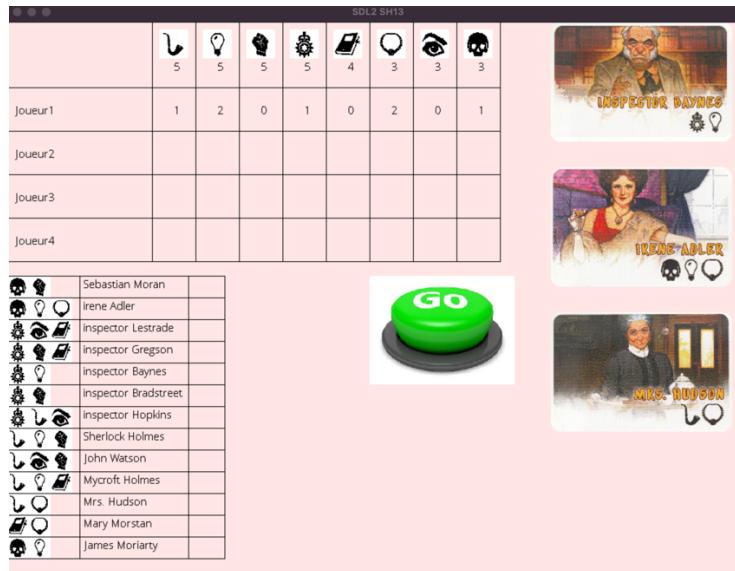
	Sebastian Moran	
	irene Adler	
	Inspector Lestrade	
	Inspector Gregson	
	Inspector Baynes	
	inspector Bradstreet	
	inspector Hopkins	
	Sherlock Holmes	
	John Watson	
	Mycroft Holmes	
	Mrs. Hudson	
	Mary Morstan	
	James Moriarty	

La suite avec les V est la ligne correspondante dans le tableau de carte que le serveur envoi.

## B. Le jeu :

Maintenant que tous les joueurs sont connectés, nous allons faire jouer ces joueurs.

Commençons par le joueur 1 qui est le premier à jouer, nous constatons qu'il a bien le bouton GO qui lui permet de valider son choix :



Pour cette la première action, le joueur 1 va demander à tous les joueurs s'ils ont l'objet pipe.

Lors de l'appui sur le bouton go, on constate sur le terminal du joueur 1 que nous avons bien une demande avec un objet, car la variable objet est à 0, par contre, le joueur n'a pas désigné de joueur spécifique (variable à -1) et que. Le joueur n'a accusé personne (variable à -1) :

```
go! joueur=-1 objet=0 guilt=-1
```

Du coté serveur la demande a bien été reçu, c'est une demande de type O :

```
Received packet from 127.0.0.1:49391
Data: [0 0 0] ← Objet demandé
] Joueur qui demande
```

La partie graphique des joueurs a bien été mise à jour, on constate que les joueurs 3 et 4 ont cet objet (\*) mais pas le joueur 2 (qui a un 0), ce qui est après vérification bien le cas :

	5	5	5	5	4	3	3	3
Joueur1	1	2	0	1	0	2	0	1
Joueur2	0							
Joueur3	*							
Joueur4	*							

	Sebastian Moran	
	Irene Adler	
	Inspector Lestrade	
	Inspector Gregson	
	Inspector Baynes	
	Inspector Bradstreet	
	Inspector Hopkins	
	Sherlock Holmes	
	John Watson	
	Mycroft Holmes	
	Mrs. Hudson	
	Mary Morstan	
	James Moriarty	





C'est au second joueur de jouer, il décide de demander au joueur 4 s'il a un crane.

Coté serveur on constate la requête du joueur 2 :

```
go! joueur=3 objet=7 guilt=-1
```

Coté serveur, nous avons bien la requête de type S avec les différentes informations :

```
Received packet from 127.0.0.1:49410
Data: [S 1 3 7] ← Objet
] ← Demande pour
    ↑           ↑
    Demande     venant de
```

On voit sur la partie graphique que la valeur 2 est apparue dans la case en du joueur 4 section crane, ce qui signifie que le joueur 4 a 2 crane, ce qui est, après vérification, bien le cas :

	5	5	5	5	4	3	3	3
Joueur1	0	0	2	2	2	1	0	0
Joueur2	0							
Joueur3	*							
Joueur4	*						2	

	Sebastian Moran	
	Irene Adler	
	Inspector Lestrade	
	Inspector Gregson	
	Inspector Baynes	
	Inspector Bradstreet	
	Inspector Hopkins	
	Sherlock Holmes	
	John Watson	
	Mycroft Holmes	
	Mrs. Hudson	
	Mary Morstan	
	James Moriarty	





Au tour du joueur 3, il décide de dénoncer Inspecteur GREGSON.

Dans le terminal du joueur 3, on constate que cette fois les objets et les joueurs sont vide (variables à -1) mais que cette fois la partie dénonciation est avec 3 ce qui correspond bien à la position de l'Inspecteur GREGSON dans le tableau que je vous ai montré avant :

```
go! joueur=-1 objet=-1 guilt=3
```

Côté serveur on constate que la demande de type G à bien été reçu, mais on voie que le joueur 3 à donné une mauvaise réponse. Effectivement la carte de l'Inspecteur GREGSON est détenu par le joueur 2 :

```
Received packet from 127.0.0.1:49432
Data: [G 2 3
]
Joueur qui demande      Position du nom
dans le tableau
Joueur3 a donné une mauvaise réponse !

```

Pour notre dernier test le joueur 4 va accuser l'Inspecteur Lestrade.

Du côté du joueur 4 on voit que la demande à bien été effectué :

```
go! joueur=-1 objet=-1 guilt=2
```

Côté serveur on constate que la demande de type G à bien été reçu, mais cette fois ci le joueur 4 a trouvé le coupable :

```
Received packet from 127.0.0.1:49472
Data: [G 3 2
]
Joueur4 a trouvé le coupable !

```

Et effectivement la carte de l'Inspecteur Lestrade n'était dans aucun des jeux des joueurs donc c'était bien lui le coupable.

Du coté client un message à été indiqué que le coupable a été trouvé :

```
consomme | Joueur4 a trouvé le coupable !
```

Le jeu ne continuera pas car nous ne changeons plus de joueur.

Nous avons donc un jeu Sherlock Holmes qui fonctionne. Et vous pouvez le relancer et le coupable ne sera pas le même.

Bonne partie !!