

# SHERLOCK HOLMES 13



Projet réalisé par EA Steven et BIROLINI Nikolaï en  
EISE4

Corrigé par : Monsieur François Pécheux

## I. Introduction

L'objectif de ce projet est de programmer en C le jeu Sherlock Holmes 13 qui est un jeu de société se jouant à 4 personnes. Afin de bien définir les fonctionnalités à réaliser dans notre code, nous devons tout d'abord expliciter les règles du jeu.

Dans un premier temps, les 13 cartes sont distribuées aléatoirement aux 4 joueurs pour que chaque joueur débute le jeu avec 3 cartes. Par conséquent, la carte restante sera la carte à deviner pour remporter la partie.

Voici des exemples de carte :



On remarque que chaque carte représente un personnage qui est associé à plusieurs symboles.

Lorsque le jeu commence, les participants joueront tour à tour et à chaque tour, ils auront la possibilité de réaliser l'une de ces trois actions :

- Accuser un personnage
- Demander à tous les joueurs s'ils possèdent un symbole
- Demander un joueur en privée le nombre du symbole demandé possédé

Si le joueur accuse le bon personnage, il gagne la partie.

Lorsqu'il demande la possession d'un symbole à tous les joueurs, il recevra une réponse binaire de chaque joueur qui sera visible par tous.

Et enfin, si la demande est privée, la réponse le sera aussi et le nombre de symbole possédé par le joueur sera dévoilé à tous les joueurs.

Ces actions permettent d'affiner la réflexion des joueurs pour deviner la carte manquante.

Ainsi, pour concevoir le jeu, il faudra faire de la programmation réseaux pour connecter tous les joueurs, de la programmation graphique pour que les joueurs puissent interagir simplement avec le jeu et de la programmation de thread pour articuler le réseau et les processus du jeu avec l'interface graphique.

## II. Explications structurelles de la programmation du jeu

### a. Architecture réseau

Dans cette partie, il est nécessaire d'utiliser plusieurs diagrammes de séquence UML pour comprendre les différents processus à décrire dans le code.

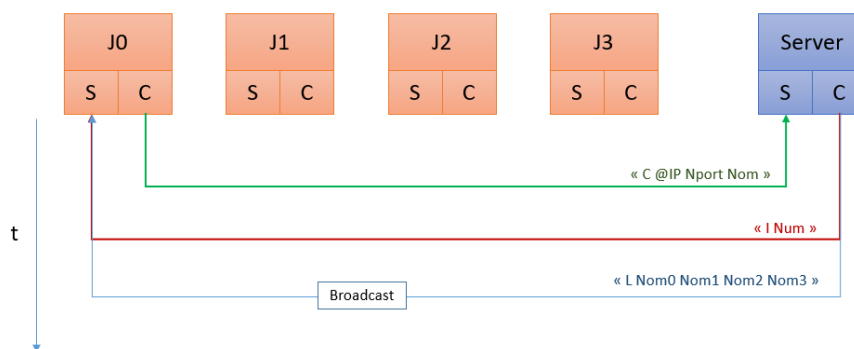
Cependant avant de décrire les protocoles de communication, il faut savoir que Sherlock Holmes 13 est un jeu se reposant sur la communication entre les joueurs, ainsi il faut établir un réseau informatique avec les différentes machines des joueurs. Dans ce réseau, il sera difficile de parler distinctement de client ou de serveur car les joueurs devront envoyer des requêtes à la machine maîtresse du jeu (demande de connexion par exemple), et cette machine devra elle aussi communiquer avec les joueurs (distribution des cartes par exemple). Ainsi chaque machine du réseau sera un client et un serveur.

De plus, nous distinguerons deux états de la machine principale dans notre code :

- L'état de connexion, le serveur principal (maître du jeu) attend la connexion des 4 joueurs.
- L'état de jeu, les 4 joueurs sont connectés, le serveur principal lance le jeu.

### b. Protocole de connexion

Afin de faciliter la compréhension des protocoles, nous avons morcelé les schémas UML.

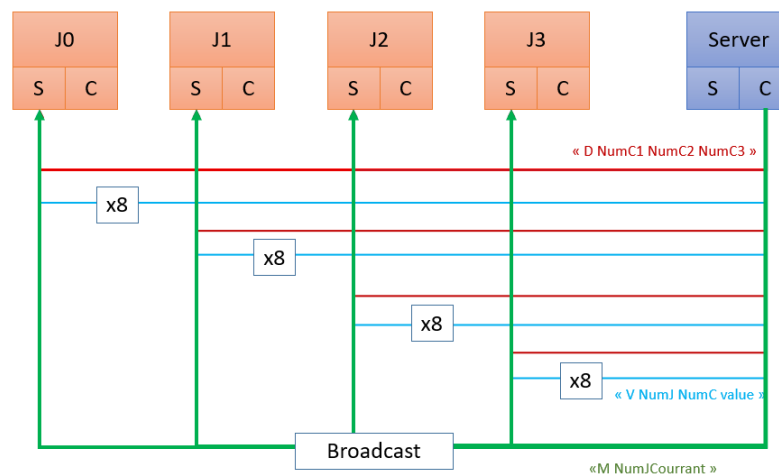


Dans cet exemple, le joueur 0 se connecte au serveur principal. La partie client du joueur0 envoie à la partie serveur du maître du jeu la commande « C @IP Nport Nom » qui est composée de l'IP du joueur0, le port de connexion du serveur et du pseudo du joueur0. Dès que la connexion est établie, le serveur envoie au joueur0 son numéro (ici 0) avec la commande « I Num ». Ensuite, le serveur envoie en broadcast à toutes les machines se trouvant sur son réseau, le nom de tous les joueurs connectés avec la commande « L Nom0 Nom1 Nom2 Nom3 ». Dans cette configuration seulement le joueur0 recevra les noms des joueurs connectés « Nom - - - » (« - », le nom des joueurs non connectés par défaut).

Durant son état de connexion, la machine principale répètera ce processus pour chaque joueur à chaque connexion.



### c. Protocole de début du jeu

Lorsque le jeu commence, le maître du jeu distribue 3 cartes aux 4 joueurs et complète pour chacun des joueurs, sa ligne de symbole lui correspondant.



La commande « D NumC1 NumC2 NumC3 » sert à distribuer les cartes à chaque joueur, le serveur transmet l'index des cartes dans le deck, ainsi le joueur0 aura les cartes Deck[0], Deck[1] et Deck[2], le joueur 1 Deck[3], Deck[4] et Deck[5] ... Ensuite, la commande « V NumJ NumC value » permet de remplir la ligne de symboles de chaque joueur. NumJ indique le numéro du joueur (ligne du tableau), NumC le numéro de la colonne du tableau et value le nombre de symbole devant être stocké dans la case du tableau. Ce nombre de symbole est déterminé grâce aux cartes possédées par le joueur.

Par exemple, si le joueur 0 est Alice, on obtiendra un tableau de cette forme :

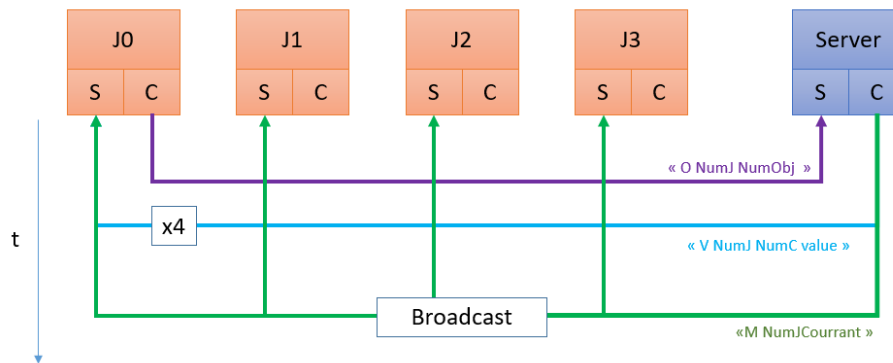
							
	5	5	5	5	4	3	3
alice	1	1	2	0	0	1	2
bob							
charlie							
denis							

On remarque que le tableau possède 8 objets différents, c'est pour cela que la requête V sera répétée 8 fois. Ce protocole est exécuté une nouvelle fois pour chaque joueur et enfin, il indiquera grâce à une requête broadcast « M numJ Courrant » le joueur ayant la main (le jeu commencera par défaut par le joueur 0 et ainsi de suite).

### d. Protocole de demande binaire à tous les joueurs

Cette demande peut être résumé par la question « Qui possède ce symbole ? ». Pour qu'un joueur active cette commande, il faut que durant son tour, il clique sur le symbole désiré dans le tableau.

Dans ce diagramme, le joueur0 réalise une demande de symbole au serveur principal.

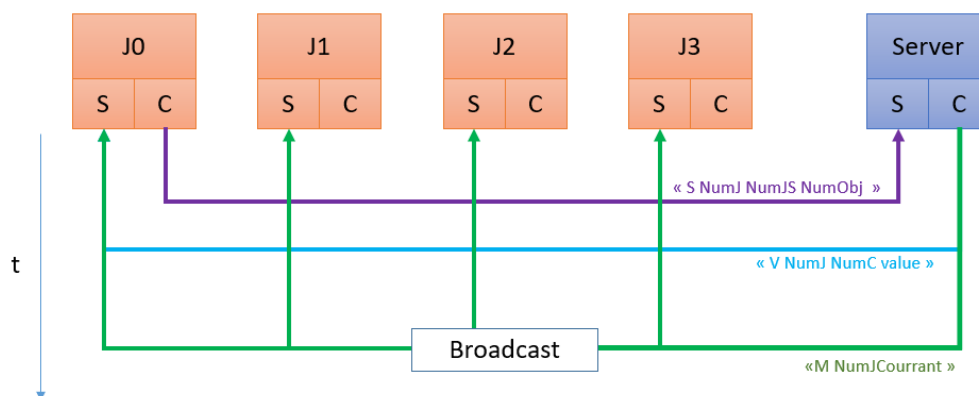


Le commande « O numJ numObj » demande au serveur maître de donner une réponse binaire si oui (« \* ») ou non (« 0 ») les joueurs possèdent le symbole demandé par le joueur0. Le numObj est le numéro de la colonne à laquelle le joueur0 a cliqué, cette colonne est associée à un objet et le numJ indique au serveur quel joueur a réalisé la requête (ici 0). Par conséquent, le programme affichera sur chaque ligne de la colonne un astérisque, si le joueur possède le symbole ou alors dans le cas contraire, un 0. Ces résultats seront visibles par tous les joueurs.

A la fin du tour du joueur0, le serveur donne la main au joueur suivant grâce à la commande « M » et diffuse un broadcast pour avertir tous les joueurs du nouveau tour.

### e. Protocole de demande privée des symboles

L'objectif de cette demande est qu'un joueur demande un autre joueur en privée de donner le nombre exact du symbole demandé. Pour réaliser cette action, il faut sélectionner l'objet voulu puis le joueur à questionner, ainsi le joueur sélectionné enverra une réponse à tous les joueurs.



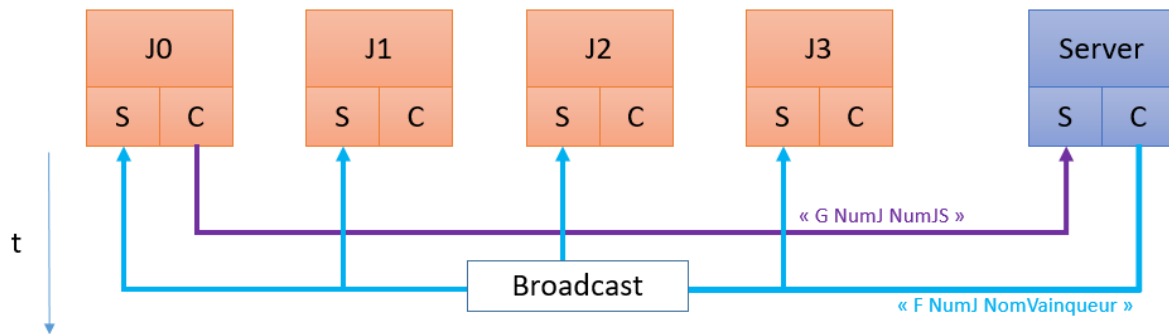
Pour faire cela, il faut faire la commande « S NumJ NumJS NumObj », NumJ indique le numéro du joueur qui fait la demande, NumJS le numéro du joueur sélectionné et NumObj le numéro de l'objet demandé. Ensuite grâce à la requête v, le serveur vous donne le nombre demandé et enfin, le joueur suivant commence son tour grâce à la commande « M ».

## f. Protocole de fin de partie

La dernière action que le joueur peut réaliser est de proposer un suspect au maître du jeu. Si le joueur trouve le coupable, la partie est finie et il gagne. Si le suspect proposé est le mauvais, le jeu continue et le tour suivant peut commencer.

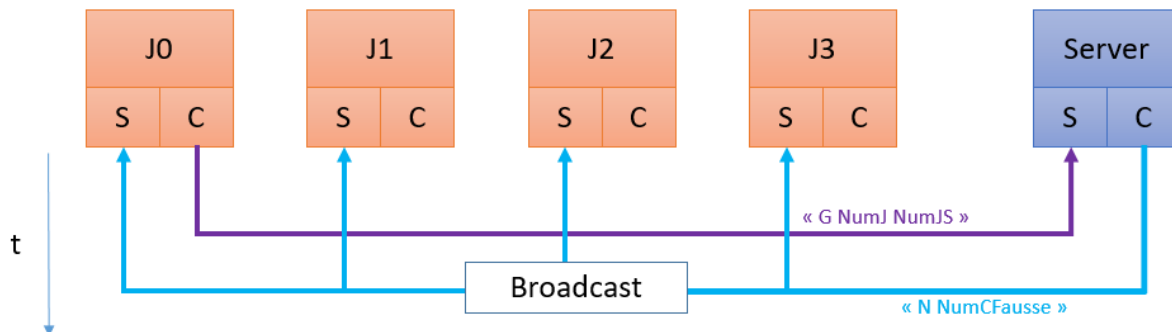
Pour effectuer cette demande, il faut cliquer sur le nom de la carte suspecté dans la liste des cartes.

Voici le Diagramme UML en cas de victoire :



L'action « G NumJ NumJS » est composé de NumJ, le numéro du joueur qui fait la demande et de NumJS qui est le numéro de la carte sélectionnée. Si la carte sélectionné est bien coupable, le jeu se termine et le serveur envoie un message broadcast qui affichera le nom du joueur vainqueur sur tous les écrans.

Voici le diagramme UML en cas de mauvais suspect :



Si le joueur se trompe, le serveur envoie la requête « N NumCFausse » qui transmet le numéro de la carte fausse à tous les joueurs. Lorsque les joueurs reçoivent ce numéro, la carte est barrée pour tout le monde.

## III. Analyse des programmes

### a. Le réseau

Dans la précédente partie, nous avons vu que les protocoles communiquent à travers un réseau TCP et que toutes les machines de ce réseau sont à la fois client et serveur.

Nous créons alors deux codes : le serveur principal qui est le garant des données du jeu ainsi que de ses structures et un fichier application pour les joueurs.

Tout d'abord la création du serveur dans les deux fichiers est identique, on crée une socket TCP, on rentre les informations du serveur dans le bind() et enfin, le programme écoute la socket avec la fonction listen() dans une boucle while(1).

Par conséquent, notre serveur principal a une IP et un numéro de port que les joueurs vont utiliser pour se connecter. Lorsque les joueurs se connectent, ils doivent informer le serveur principal à propos des identifiants de leur serveur afin que le maître du jeu puisse leur envoyer des requêtes.

Pour que le serveur principal prenne connaissance de cette information, lors de la commande de connexion d'un joueur, le joueur enverra ses informations de connexions (il enverra le numéro de port de son serveur et son IP).

## b. La partie graphique

La gestion de l'interface graphique du jeu est réalisée dans le fichier du joueur sh13.c, cet interface graphique est réalisé avec la librairie SDL2.0.

Tout d'abord, on fait les includes des interfaces des bibliothèques afin d'avoir accès aux fonctions puis dans le main, on crée notre fenêtre avec la fonction :

```
SDL_CreateWindow("SDL2 SH13", SDL_WINDOWPOS_UNDEFINED, SDL_WINDOWPOS_UNDEFINED, 1024, 768, 0);
```

Ensuite, on définit la variable « SDL\_Event event » car la programmation graphique est une programmation événementielle, c'est-à-dire que dans une boucle infini nous scruterons ces différents événements :

- SDL\_QUIT : Évènement « quitter le jeu »
- SDL\_MOUSEBUTTONDOWN: « Évènement cliquer », selon les coordonnées cliqués, le programme va reconnaître différentes actions (appuie sur un bouton ou non)
- SDL\_MOUSEMOTION: « Évènement déplacement de la souris », indique la position x et y de la souris

Si un événement est réalisé alors il faut mettre à jour la fenêtre graphique pour ensuite la redessiner.

Cette dernière étape fonctionne grâce à un système de double buffer, un buffer est utilisé pour afficher la fenêtre graphique et l'autre buffer stocke les mises à jour graphique de la fenêtre.

La fonction *SDL\_CreateRenderer(window, -1, 0)* sert à construire le renderer qui commutera ces deux buffers pour éviter le scintillement de l'écran.

Enfin, pour placer des images sur la fenêtre graphique, il faut les charger avec la fonction *IMG\_Load(« NomImage.png »)*, les mettre sur une surface avec *SDL\_Surface* pour enfin les mettre sur des textures avec *SDL\_CreateTextureFromSurface(renderer, surfaceDeImage)*.

Il existe de nombreuses fonctions stylistiques dans la SDL : *TTF\_Init()* (permet de modifier la police d'écriture) ou encore *SDL\_SetColorKey()* (permet de modifier la couleur de l'arrière-plan d'une image).

### c. Les Threads

Nous remarquons que la boucle réseau et la boucle des événements sont toutes les deux des boucles infinies et nous devons exécuter ces deux codes sur le fichier du joueur.

Cependant, il est impossible de fusionner ces deux boucles infinies car la fonction `accept()` dans le code réseau est bloquante et cela est problématique car la boucle événementielle ne doit jamais être interrompue. Il faut donc paralléliser les deux tâches à l'aide des threads.

Dans `sh13.c` (le fichier joueur), on déclare un thread réseau `pthread_t thread_serveur_tcp_id`; puis on l'exécute : `ret = pthread_create ( & thread_serveur_tcp_id, NULL, fn_serveur_tcp, NULL);`

Grâce à cette stratégie, nous pouvons faire tourner les deux fonctions, mais il faut que les événements réseaux influencent l'état graphique de la fenêtre. Par conséquent, il faut synchroniser les deux threads.

Pour cela, on va définir la variable synchro de type volatile int, déclarer une variable en volatile force le programme à regarder la valeur « réelle » de cette variable dans les registres (pas dans le cache).

Ainsi, si une information est transmise dans le thread réseau, cette variable sera mise à 1, ce qui provoquera dans le thread principal, après les événements graphiques, la détection d'un événement réseau. Bien sûr, après que l'événement réseau ait été pris en compte, on remet la variable synchro à 0.

## IV. Conclusion

Le jeu est terminé et il est fonctionnel, nous vous souhaitons une agréable expérience !!!