Sorbonne Université

Sherlock 13

Rapport projet

Amine Oumert
21/04/2025

Rapport de développement du programme multijoueur (client & serveur)

Introduction

Ce projet vise à compléter deux codes d'un serveur et client multijoueur en langage C implique la gestion de la communication entre 04 client et du serveur, ainsi que la gestion des cartes et des joueurs. Ce rapport présente les différentes étapes et méthodologies adoptées pour la construction du programme. Un serveur centralisé gère la communication avec plusieurs clients. Le serveur maintien des informations sur les cartes de jeu et les états des joueurs. La gestion des interactions entre les joueurs et la synchronisation de communication, ce qui nous amène à l'utilisation de sockets et de threads.

Méthodologie de Développement

Dans un premier temps, j'ai commencé par analyser en détail les règles du jeu afin de bien comprendre le jeu.

Ensuite, j'ai étudié les deux codes fournis (serveur et client). Pour comprendre la structure générale de chaque programme, et identifier les parties déjà implémentées et celles qui restaient à compléter. Certaines portions du code, en particulier celles relatives à la gestion réseau et à la synchronisation, n'étaient pas immédiatement claires. J'ai donc utilisé IA pour obtenir des explications sur certains segments de code, ce qui m'a permis d'approfondir ma compréhension.

Une fois la structure générale maîtrisée et les parties manquantes identifiées, j'ai pu compléter les deux codes ce qui est était utile pour compléter le code ces les portions des code tets qu'on a pu développer durant le cours et les TP. Pour le serveur, les fonctionnalités principales à implémenter comprenaient : la gestion des tours de jeu, la distribution des cartes, ainsi que la validation des actions des joueurs. Pour le client, les ajouts portaient principalement sur le traitement des messages reçus du serveur.

Enfin, j'ai procédé à une phase de tests fonctionnels en simulant les échanges de quatre clients (joueurs) connectés simultanément avec le serveur.

Architecture des Programmes Serveur et Client

Serveur

1. Gestion des connexions
    - Crée un socket TCP sur le port spécifié (par exemple 8080)
    - Utilise une boucle accept() pour gérer la connexion de 4 joueurs(client)
    - Stocke chaque client (IP, port, nom) dans le tableau tcpClients

2. Gestion du jeu

- Mélange les cartes fonction melangerDeck() (exemple de sortie : [7, 12, 3] (Sherlock, Moriarty, Gregson)
- Initialise le plateau avec createTable() (associe cartes/symboles) avec Sherlock, Moriarty, Gregson ça nous fait la combinaison des cartes suivante
    - **Sherlock Holmes :**

      tableCartes[0][0] ++ ; →Pipe

      tableCartes[0][1] ++ ; →Ampoule

      tableCartes[0][2] ++ ; → Poing
    - **James Moriarty :**

      tableCartes[0][7] ++ ; → Crâne

      tableCartes[0][1] ++ ; → Ampoule
    - **Inspecteur Gregson**

      tableCartes[0][3] ++ ; → Couronne

      tableCartes[0][2] ++ ; → Poing

      tableCartes[0][4] ++ ; → Carnet
- Alterne les tours avec joueurCourant
- Valide les accusations en comparant avec deck [12] (coupable)

3. Communication Client-Serveur

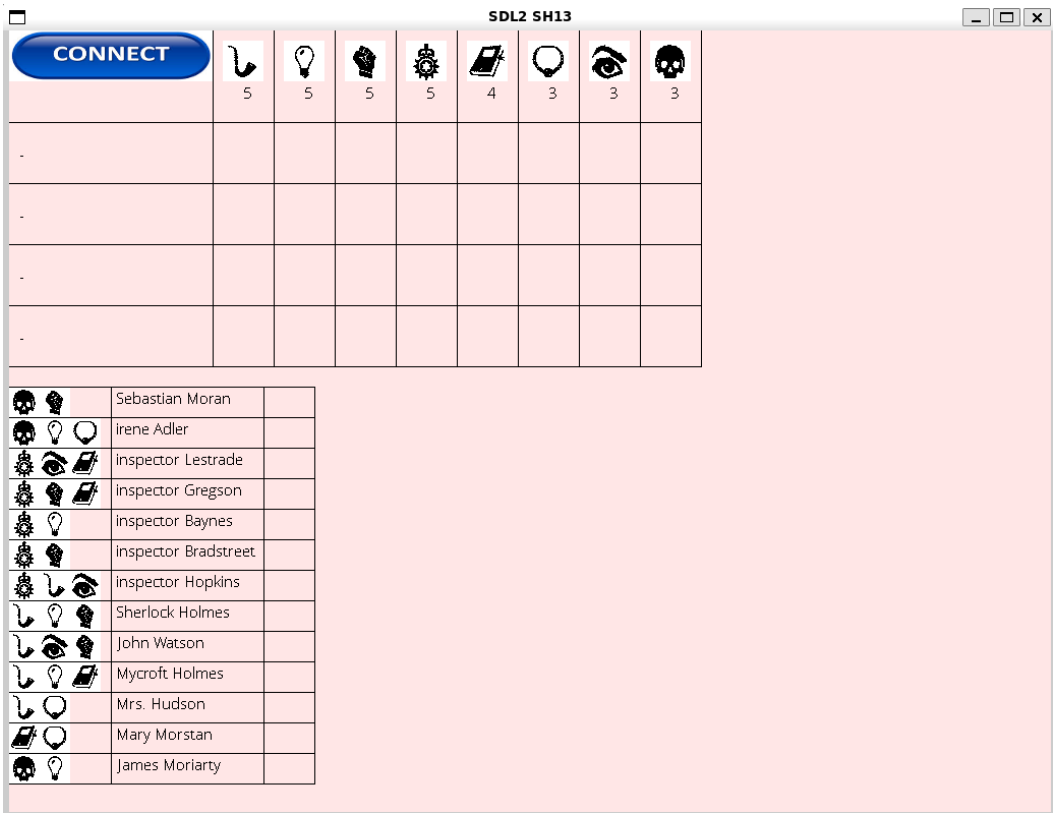
Commande	Origine	Format	Description
C	Client	C <IP> <Port> <Nom>	Connexion au serveur
I	Serveur	I <ID>	Attribution d'un ID au joueur
L	Serveur	L <Joueur1> ...	Liste des joueurs connectés
D	Serveur	D <Carte1> <Carte2>	Distribution des cartes au joueur
M	Serveur	M <ID>	Définit le joueur courant
G	Client	G <ID> <Suspect>	Accusation ex: G 2 12 = Joueur 2 accuse Moriarty
O	Client	O <ID> <Symbole>	Observation d'un symbole
S	Client	S <ID> <Cible> <Symbole>	Suggestion à un autre joueur

Client

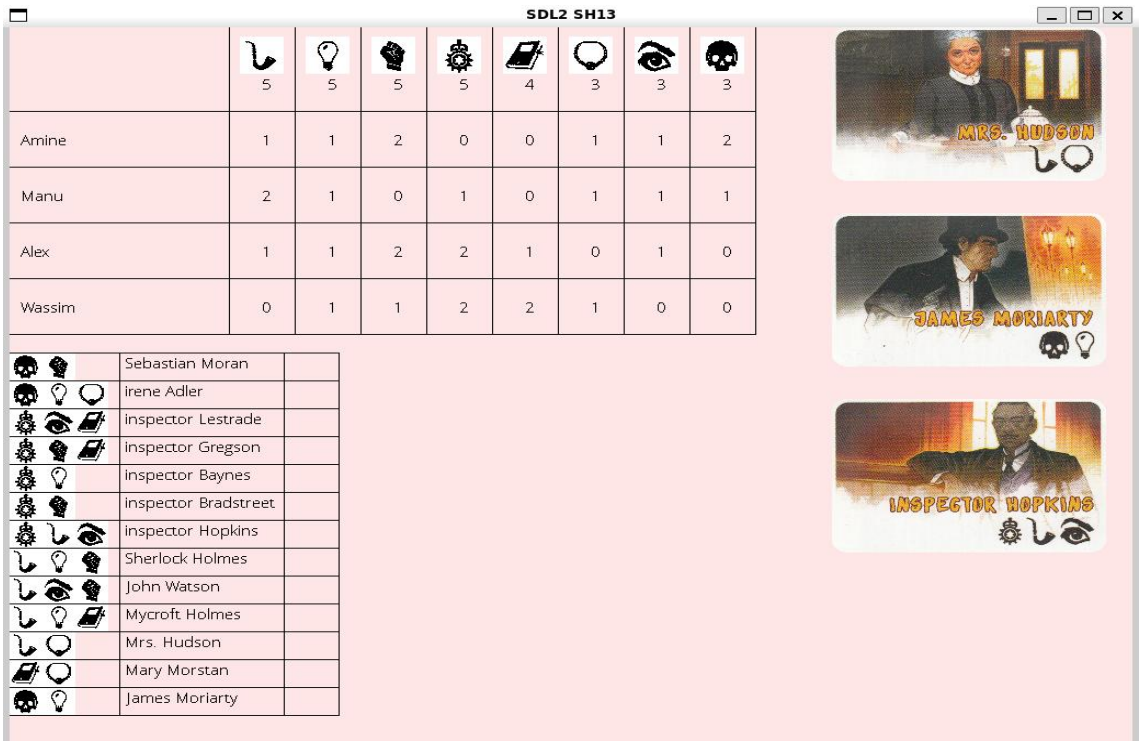
1. Interface graphique (SDL2) :
    - Affiche les cartes et la grille de jeu
    - Propose des boutons cliquables (*Connect*, *Go*)
    - Permet de sélectionner des éléments (joueurs, symboles, suspects)
    - L'interface SDL s'appuie sur le thread principal, tandis qu'un thread secondaire (fn_serveur_tcp) gère la réception des messages
2. Communication réseau :
    - Un thread reçoit en continu les messages du serveur « fn_serveur_tcp »
    - Met à jour l'état du jeu via des variables partagées
    - Envoie les actions du joueur (accuser, questionner)
3. Gestion de l'état :
    - Stockage des infos comme les cartes reçues
    - Synchronise les données entre le réseau et l'interface

Déroulement de jeu

Le jeu suit un cycle tour-par-tour géré par le serveur.



Lorsque 4 joueurs sont connectés, le serveur mélange les cartes et attribue à chaque joueur 3 personnages



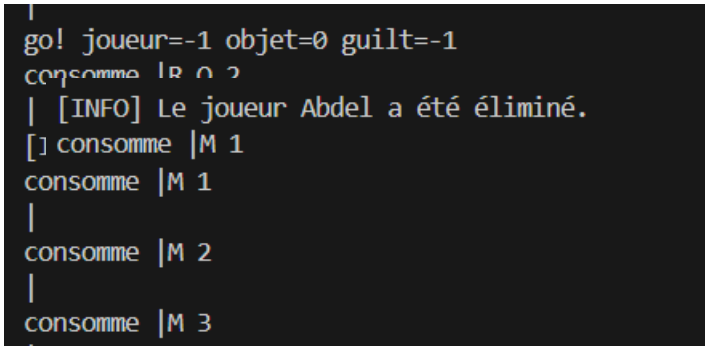
Pendant un tour le serveur sélection un joueur le quelle a le tour pour jouer et le bouton Go s'active

```
consomme |M 0
```

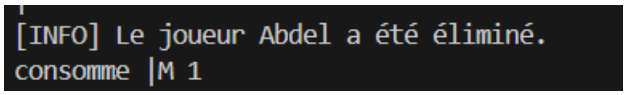
Le joueur actif peut :

1. Questionner :

- Sélectionner un symbole (ex: pipe)
- Cliquer sur "Go" pour envoyer le nombre de symbole de contient de chaque joueur



2. Accuser :
   - Sélectionner un suspect
   - Cliquer sur "Go" pour envoyer G <ID_joueur> <ID_suspect>
   - Le serveur vérifie si c'est effectivement le coupable (ID_suspect). Si c'est correct, le joueur remporte la partie et tous les clients reçoivent un message de fin ; sinon, le joueur est éliminé.



Conclusion

Ce projet nous a permis de comprendre une architecture de serveur et client multijoueur avec une communication via TCP, en exploitant les concepts étudiés dans le module tels que les sockets, les threads. Ces mécanismes ont permis de garantir une gestion efficace de la communication réseau, une synchronisation correcte des données, et une gestion concurrente des clients.