

Projet : Sherlock Holmes 13

Table des matières

<i>Introduction</i>	2
I. Règles du jeu	2
Distribution initiale.....	2
Objectif	2
Actions possibles à son tour	2
Fin de la partie	3
II. Architecture générale	3
1. Serveur	3
2. Clients	3
3. Communication réseau	4
III. Protocole de communication.....	4
1. Messages envoyés par les clients	5
2. Messages envoyés par le serveur	5
3. État synchrone	5
IV. Intégration système et gestion des communications	5
V. Test	7
Conclusion	10

Introduction

Ce projet est une implémentation en réseau du jeu **Sherlock Holmes 13**, un jeu de déduction logique dans lequel les joueurs doivent identifier un suspect parmi une liste de treize. Chaque joueur reçoit des cartes représentant des personnages associés à des objets, et en posant des questions aux autres joueurs, ils essaient d'exclure les suspects un à un jusqu'à trouver le coupable.

Ce projet a été développé en **langage C** en utilisant :

- **SDL2** pour l'interface graphique des clients,
- **Sockets TCP/IP** pour la communication réseau,
- **pthreads** pour la gestion des échanges côté client.

L'objectif était de simuler une partie du jeu SH13 à travers une architecture client-serveur, avec un serveur centralisé et quatre clients autonomes qui communiquent avec ce dernier.

I. Règles du jeu

Le jeu Sherlock Holmes 13 repose sur un principe de déduction : parmi 13 suspects, un seul est le coupable. Chacun de ces suspects est défini par un ensemble d'objets (pipe, carnet, collier, etc.).

Distribution initiale

- Chaque joueur reçoit 3 cartes suspect (personnages), soit 12 cartes réparties entre les 4 joueurs.
- La 13^e carte, non distribuée, représente le coupable.
- Chaque personnage est associé à certains objets, ce qui permet d'établir une table d'indices (tableCartes) propre à chaque joueur.

Objectif

Déduire l'identité du coupable en interrogeant les autres joueurs sur les objets associés aux suspects et en croisant les informations obtenues avec ses propres cartes et indices.

Actions possibles à son tour

Le joueur courant peut effectuer une seule action parmi :

Projet_OSUSER_Tom-Dema_GNOKPOWOU-TAZZOU

- G <id> <suspect> : proposer une accusation directe (guess). Si elle est correcte, le joueur gagne ; sinon, il perd son tour.
- O <id> <objet> : demander à tous les autres joueurs s'ils possèdent cet objet. Chacun répond de manière binaire (oui/non).
- S <id> <cible> <objet> : demander à un joueur spécifique combien de fois un objet apparaît dans ses cartes.

Fin de la partie

La partie se termine dès qu'un joueur identifie correctement le coupable. Un message est alors envoyé à tous les clients pour signaler la fin de la partie.

II. Architecture générale

Le projet SH13 repose sur une **architecture client-serveur** répartie comme suit :

1. Serveur

Le serveur centralise :

- La gestion des connexions des clients (jusqu'à 4),
- La distribution des cartes à chaque joueur,
- La gestion du tour de jeu (joueur courant),
- La réception des actions des joueurs (questions, accusations),
- La diffusion des réponses et de l'état du jeu à tous les clients.

Le serveur est implémenté en C avec les sockets TCP/IP et n'a pas d'interface graphique. Il est démarré avec un port de réception des connexions clients en argument.

2. Clients

Chaque client représente un joueur et dispose :

- D'une interface graphique réalisée avec SDL2 pour interagir avec le jeu,
- D'un thread serveur TCP local pour recevoir les messages du serveur,
- D'une logique de traitement des messages entrants,
- D'un envoi d'actions (via socket TCP) selon les clics réalisés sur l'interface.

Chaque client se lance avec 5 paramètres :

`./client <ipServeur> <portServeur> <ipClient> <portClient> <nom>`

3. Communication réseau

- Le serveur écoute les messages des clients sur un seul port.
- Chaque client écoute sur son propre port TCP pour recevoir les messages du serveur.
- La communication est asynchrone : un thread par client gère la réception.

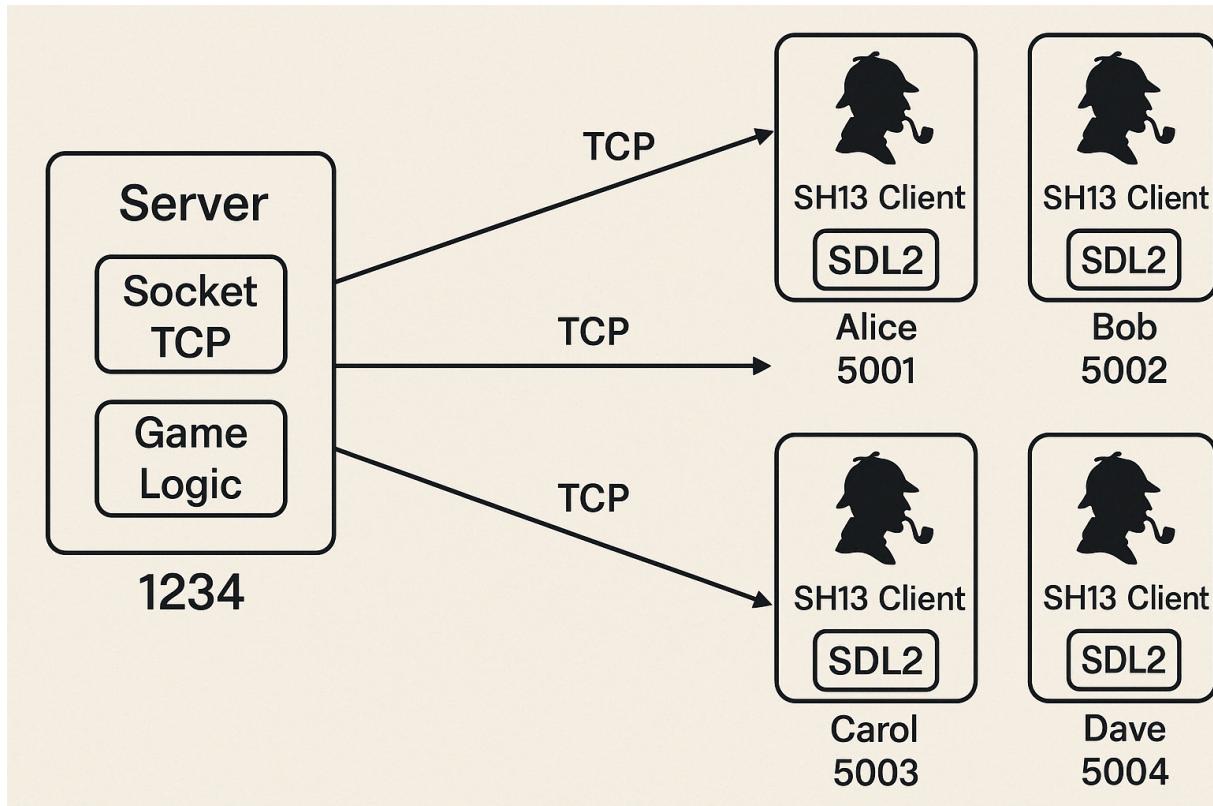


Figure 1-Schéma de l'architecture client-serveur du jeu SH13 utilisant des sockets TCP

III. Protocole de communication

La communication entre les clients et le serveur repose sur un protocole texte simple, utilisant des messages encodés sous forme de chaînes de caractères. Chaque message commence par une lettre-clé qui indique l'action ou le type de message, suivie des paramètres nécessaires.

Projet_OSUSER_Tom-Dema_GNOKPOWOU-TAZZOU

1. Messages envoyés par les clients

C <ipClient> <portClient> <nom>	Utilisé pour s'enregistrer auprès du serveur à la connexion.
G <id> <suspect>	Le joueur avec l'id id accuse le suspect suspect.
O <id> <objet>	Le joueur avec l'id id interroge tous les autres sur la présence de l'objet objet
S <id> <cible> <objet>	Le joueur avec l'id id interroge un joueur spécifique (cible) sur l'objet objet

2. Messages envoyés par le serveur

I <id>	Envoie au joueur son identifiant unique
L <nom0> <nom1> <nom2> <nom3>	Envoie à tous les clients la liste des joueurs connectés
D <carte1> <carte2> <carte3>	Envoie au joueur ses 3 cartes personnelles
V <joueur> <objet> <valeur>	Mise à jour de la table d'indices : indique qu'un joueur possède (ou non) un objet ¹
M <joueurCourant>	Définit quel joueur a la main et peut jouer
F <message>	Message de fin de partie envoyé à tous (victoire ou erreur)

3. État synchrone

Pour la réception côté client, une variable synchro est utilisée pour gérer la consommation séquentielle des messages dans le thread de réception. Ce qui permet de traiter un message à la fois, dans l'ordre d'arrivée (Respect de la logique du jeu)

IV. Intégration système et gestion des communications

Le projet a été développé pour être entièrement compatible avec macOS et Linux, avec une attention particulière portée à la configuration des bibliothèques SDL2, SDL_image et SDL_ttf. Le Makefile a été adapté pour détecter automatiquement l'environnement et compiler correctement les fichiers source selon les bibliothèques installées. Le protocole de communication TCP entre les clients et le serveur repose sur des messages structurés (C, I, L, D, V, M, etc.) permettant d'organiser les connexions, distribuer les cartes, transmettre les actions de jeu et mettre à jour les états. Graphiquement, l'interface client affiche toutes les informations nécessaires : la **grille d'indices** (4

¹ Valeur = 0 : l'objet est absent

Valeur > 0 : l'objet est présent valeur fois

Valeur = 100 : information confirmée (réponse indirecte)

Projet_OSUSER_Tom-Dema_GNOKPOWOU-TAZZOU

joueurs × 8 objets) se remplit dynamiquement à la réception des messages V, les **cartes du joueur** sont visibles dans une zone dédiée à droite, et la **sélection d'un joueur ou d'un objet** modifie la couleur des cases correspondantes pour indiquer les choix en cours. Le bouton "**GO**", activé uniquement lorsque c'est le tour du joueur (M id reçu), permet de lancer une action selon la sélection en cours : supposition du coupable (G), question globale (O), ou question ciblée (S). Les suppositions du joueur sur les suspects sont cochées dans une colonne dédiée, avec des croix dessinées à la main sur la grille. Enfin, les noms des joueurs sont affichés à gauche de l'écran, se mettant à jour automatiquement à la réception du message L. Cette intégration graphique assure une interaction intuitive avec les mécaniques du jeu, tout en maintenant la synchronisation réseau entre tous les participants.

Par exemple : Lorsqu'un joueur se connecte (*Alice*), le client envoie un message **C 127.0.0.1 5001 Tom-Dema** au serveur pour signaler son adresse IP, son port d'écoute et son nom. Le serveur enregistre ces informations, puis retourne à ce joueur son identifiant via un message **I 0**. Ensuite, une diffusion (**L Tom-Dema Eyadema Eya-Tom Wiyao**) est faite à tous les joueurs pour leur indiquer la liste complète des participants. Une fois les 4 joueurs connectés, le serveur distribue à chacun ses cartes (**D 3 5 7**) et les indices (**V 0 0 1**, etc.), puis annonce le joueur qui commence (**M 0**). Ce protocole d'échange structuré garantit que chaque action est bien synchronisée, grâce à la variable synchro utilisée côté client pour séquencer la réception et l'affichage des informations dans l'interface graphique.

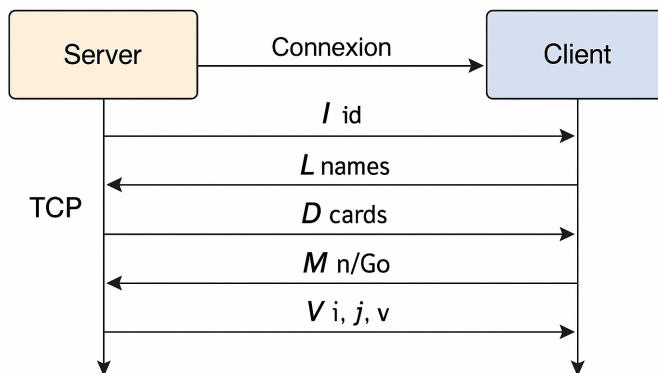


Figure 2 – Échanges de messages client - serveur

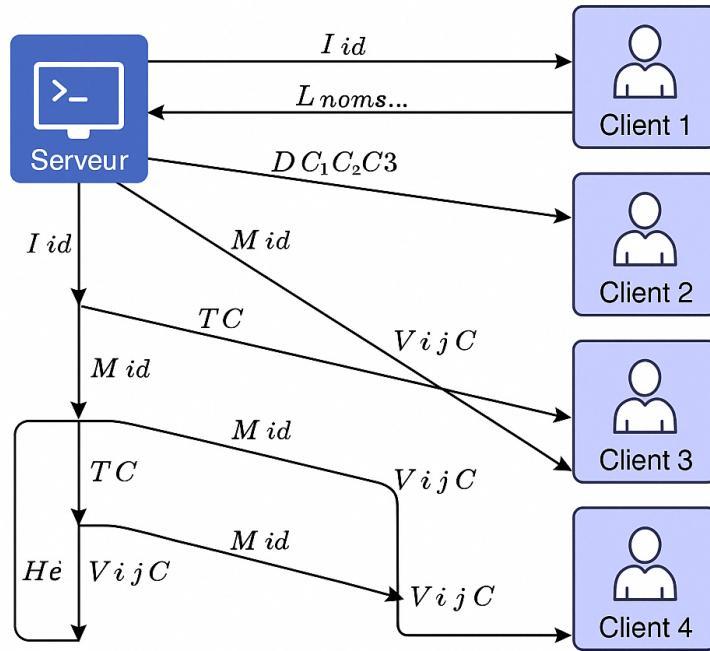
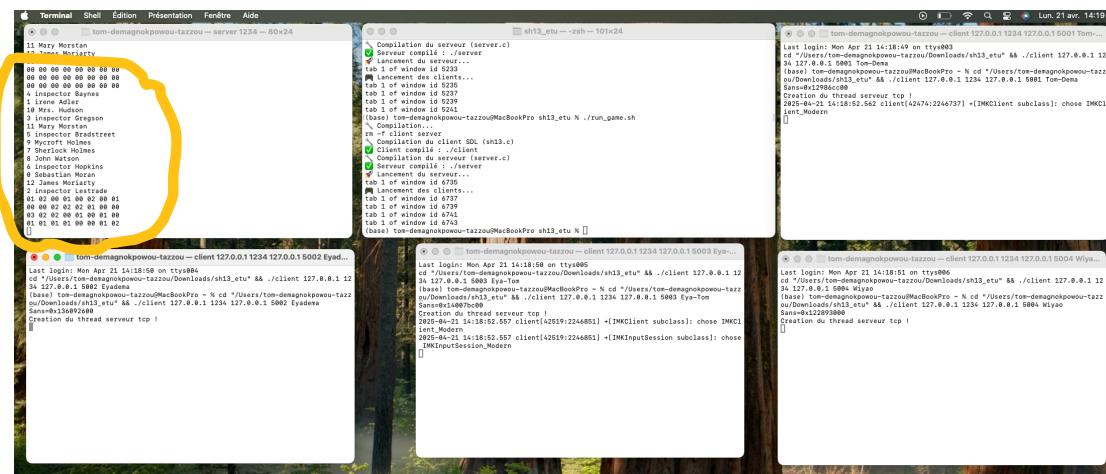


Figure 3 – Échanges de messages joueurs – serveur dans le jeu

V. Test

Une partie a été joué à l'arrache pour tester le jeu.

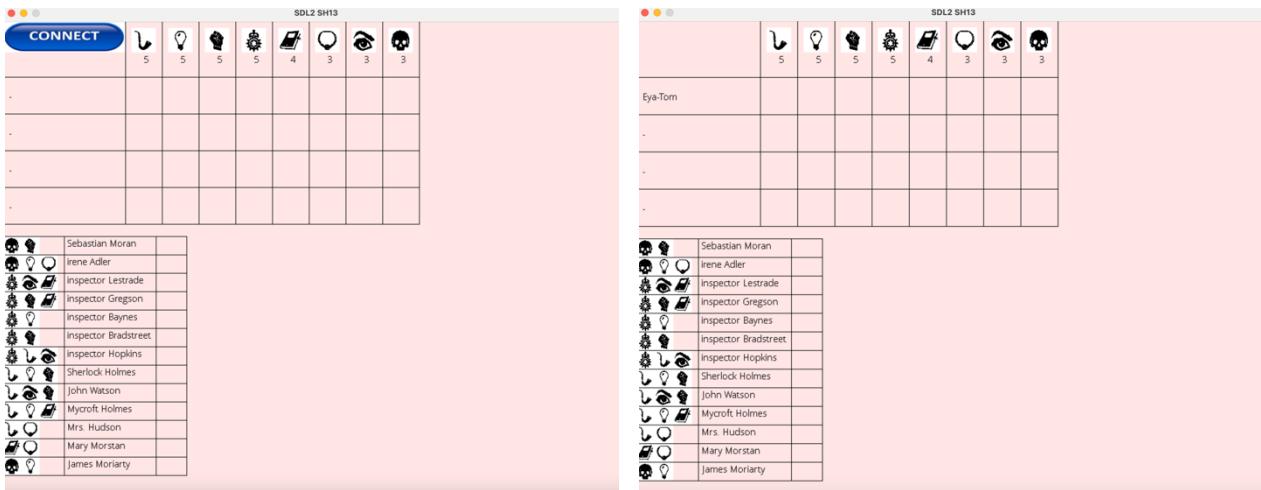
Un script run_game.sh est fourni pour démarrer automatiquement le serveur et les quatre clients.



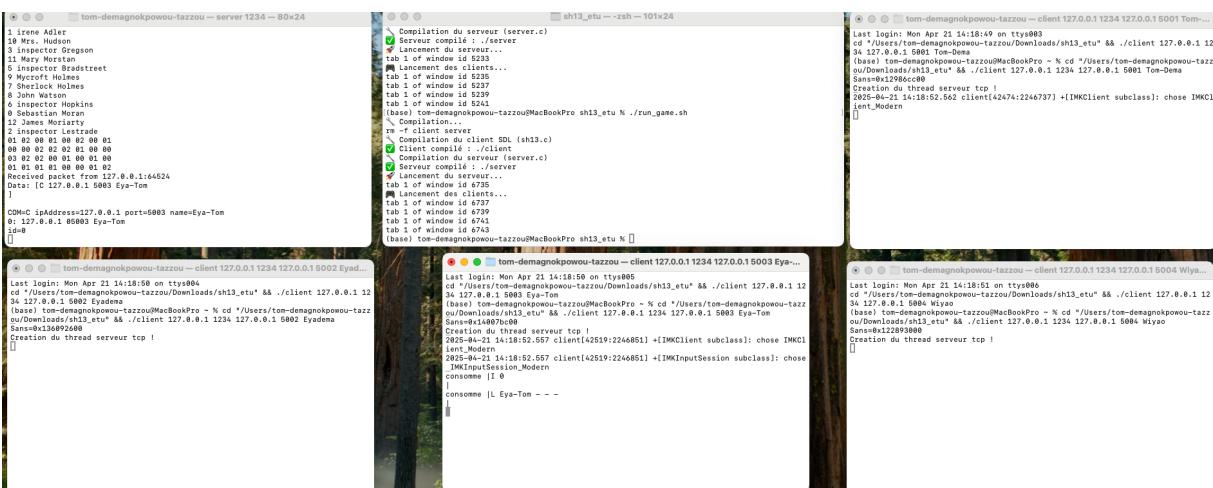
Cartes mélangées.

Projet_OSUSER_Tom-Dema_GNOKPOWOU-TAZZOU

On voit bien qu'un server est lancé ainsi que 4 autres terminaux pour les joueurs. Le server attend que les joueurs se connectent.

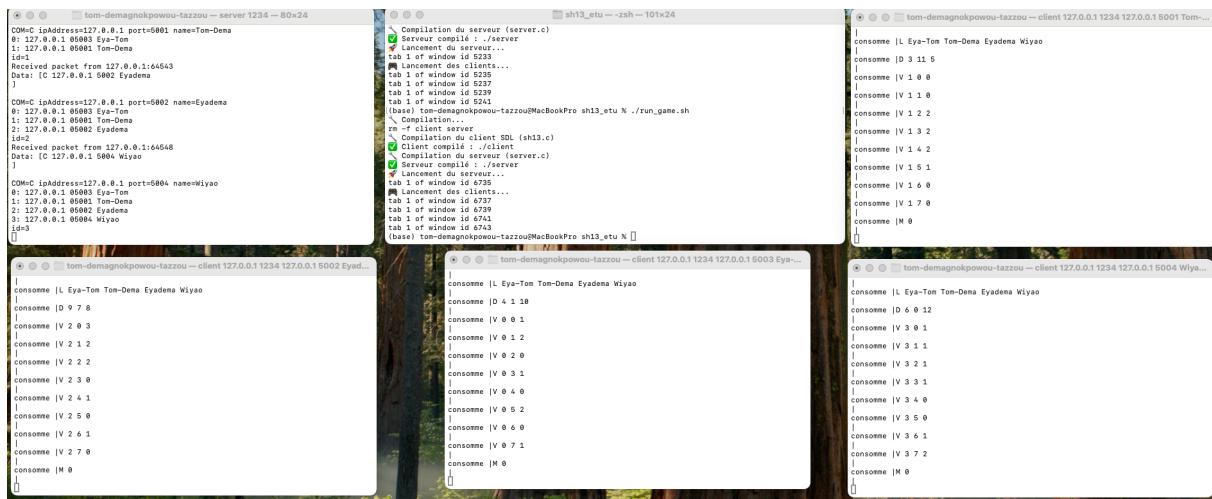


On voit bien que le simple fait que Eya-Tom soit connecté, ne lance pas le jeu. Le server attend que tous les joueurs soient connectés pour distribuer les cartes au préalable mélangées.



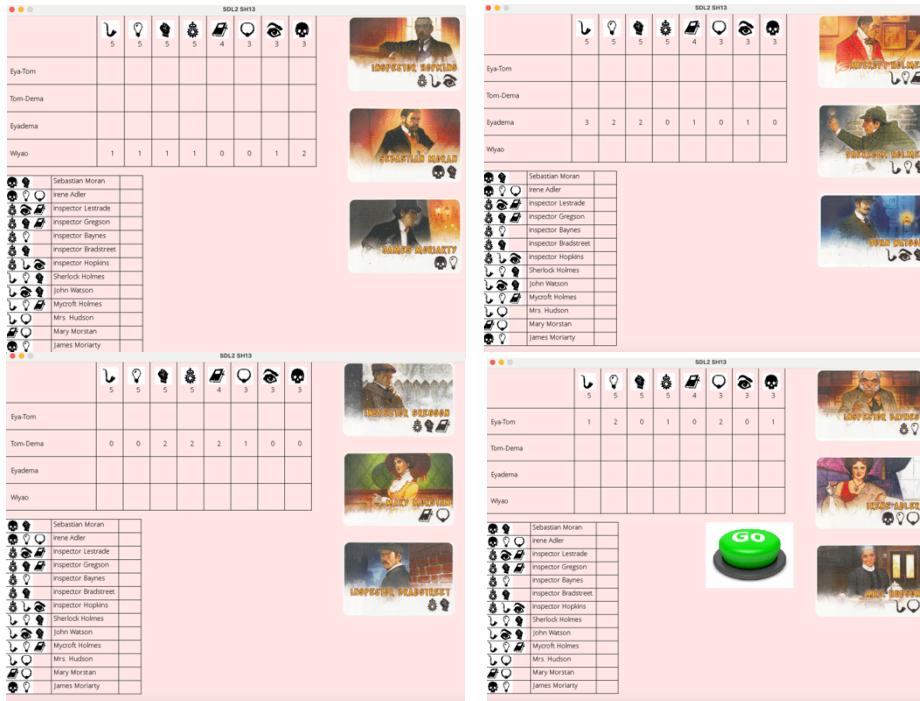
Eya-Tom seul connecté

Projet_OSUSER_Tom-Dema_GNOKPOWOU-TAZZOU



Tous connectés.

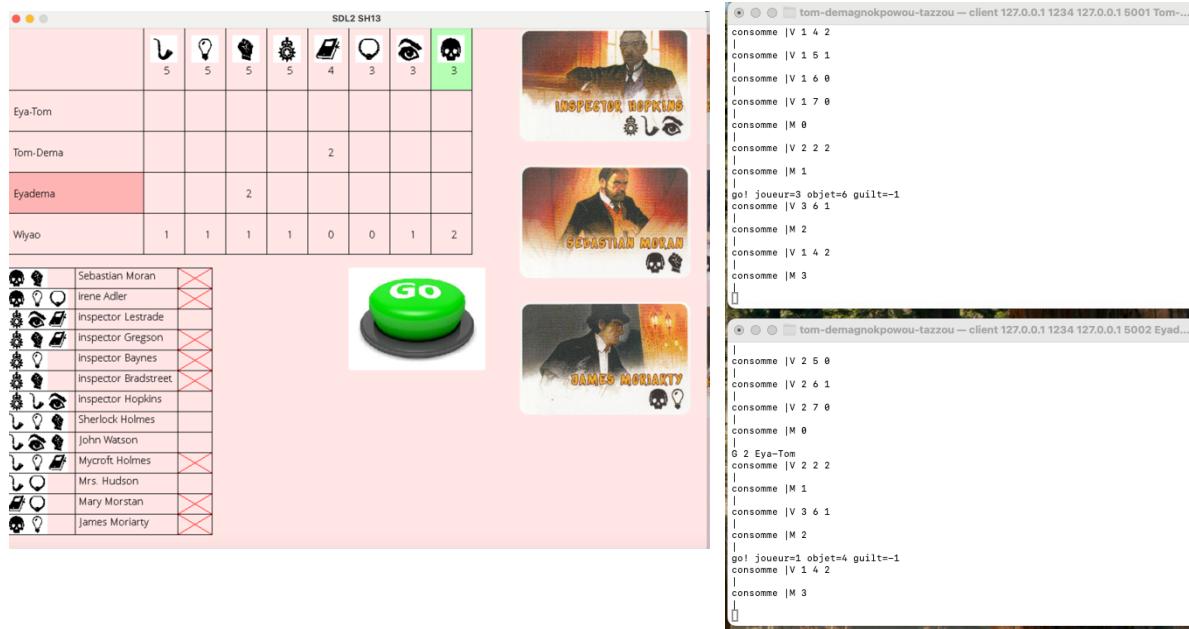
Une fois tous les joueurs connectés les cartes sont distribués.



Chaque joueur ayant obtenu ses cartes, la partie peut commencer !

Projet_OSUSER_Tom-Dema_GNOKPOWOU-TAZZOU

Après que Eya-Tom (joueur 1) fait une action, le bouton de go se déplace vers un autre joueur et ainsi de suite jusqu'à ce que la partie ne se termine.



Conclusion

Ce projet, réalisé de manière individuelle, a été une expérience enrichissante tant sur le plan technique que conceptuel. Il m'a permis d'approfondir mes connaissances sur les systèmes d'exploitation (cours OSUSER), notamment à travers la gestion des processus, la communication réseau via les sockets TCP, et l'utilisation des threads pour le parallélisme. Le thread serveur implémenté côté client permet de gérer la réception asynchrone des messages tout en laissant l'interface graphique réactive et fonctionnelle, illustrant parfaitement le principe de concurrence. De plus, la synchronisation via une variable partagée (synchro) m'a permis de comprendre les défis liés aux accès concurrents en environnement multi-thread. La conception du protocole de messages entre serveur et clients, et sa gestion en temps réel, m'a aussi sensibilisé à l'importance de la synchronisation des états dans des applications distribuées. Enfin, le projet offre une perspective concrète sur l'interaction entre le noyau système, les interfaces réseau et la gestion graphique, démontrant comment différents modules logiciels s'articulent autour des fonctionnalités fondamentales d'un OS.