Rapport de Projet - OS USER - SH13

Etudiant : Ugo LUCCHI **Module** : OS USER **Projet** : SH13 - Interface graphique et réseau du jeu Sherlock Holmes 13 Indices

---

1. Introduction

Ce projet avait pour objectif de compléter une base de code fournie afin de créer une version multijoueur en réseau local du jeu "Sherlock Holmes: 13 Indices", avec une interface graphique réalisée en SDL2.

Deux fichiers étaient fournis partiellement complétés : `server.c` (serveur de jeu) et `sh13.c` (client SDL). Le but était de permettre l'interaction de 4 joueurs, chacun contrôlant un client, communiquant avec le serveur via TCP.

Le projet m'a permis de mettre en application concrètement les notions de :

- Sockets TCP
- Threads
- Mutex
- Architecture client/serveur

---

2. Règles du jeu - Sherlock 13

- 13 cartes représentent des suspects (Sherlock Holmes, Moriarty, etc.)
- Chaque carte a des symboles : pipe, oeil, carnet, couronne...
- Au début : chaque joueur reçoit 3 cartes, 1 est mise à part (le coupable)
- Objectif : deviner le coupable en posant des questions et en accusant

Pendant son tour, un joueur peut :

1. Poser une question sur un symbole à tout le monde
2. Poser une question ciblée à un joueur
3. Accuser un personnage (si erreur = éliminé)

---

3. Architecture du projet

Serveur (`server.c`)

- Attend la connexion de 4 clients
- Gère la répartition aléatoire des cartes (fonction `melangerDeck()`)
- Calcule la table des symboles pour chaque joueur (`createTable()`)
- Envoie les cartes et infos aux joueurs
- Gère le tour de jeu, les questions (types S et O), les accusations (G), et la fin de partie

Client (`sh13.c`)

- Interface SDL2 : rend cartes, suspects, objets, boutons (Connect/Go)
- Chaque client reçoit ses cartes, les noms des joueurs, et les indices
- Un thread TCP reçoit les messages du serveur pour ne pas bloquer SDL
- Envoie les actions au serveur (connexion, question, accusation)
- Utilise un mutex pour protéger l'accès au buffer `gbuffer`

---

4. Travaux réalisés

Coté serveur :

- **Connection des joueurs** : gestion du message `C` pour enregistrer IP/port/nom
- **Distribution des cartes** : à chaque joueur, via `D ...` et `V ...`
- **Tour par tour** : `M id` pour indiquer qui peut jouer
- **Gestion des actions** :
  - `O` : question ouverte (broadcast avec réponse étoile ou 0)
  - `S` : question ciblée (unicast)
  - `G` : accusation, avec gestion de la fin si correcte
  - `P` : joueur passif (après erreur)

Coté client :

- **Thread TCP** : fonction `fn_serveur_tcp()` pour lire les messages serveur
- **Protection buffer** : remplacement de `volatile synchro` par `mutex + flag` non bloquant
- **Affichage SDL** : rendu des cartes, suspects, grille d'indices, boutons
- **Gestion des clics** : interaction via `SDL_MOUSEBUTTONDOWN`
- **Traitement messages serveur** : `I`, `L`, `D`, `M`, `V`, `A`, `E`
- **Actions joueurs** : envoi de `C`, `S`, `O`, `G` selon clics

---

5. Compilation et exécution

Compilation (Linux)

```
sudo apt install libsdl2-dev libsdl2-image-dev libsdl2-ttf-dev
./scripts/compile.sh
```

Compilation (macOS)

```
brew install sdl2 sdl2_image sdl2_ttf
./scripts/compile_macos.sh
```

Exécution

1. Lancer le serveur : `./exec/server 1234`
2. Lancer les 4 clients dans 4 terminaux :

```
./scripts/run_sherlock0.sh
./scripts/run_sherlock1.sh
./scripts/run_sherlock2.sh
./scripts/run_sherlock3.sh
```

---

6. Difficultés rencontrées

- Synchronisation entre thread TCP et boucle SDL : j'ai remplacé `synchro` par un mutex + flag pour éviter le polling bloquant
- Comprendre la structure du jeu et le décodage des messages
- Affichage conditionnel dans SDL : marquage de cartes, indices, gagnant, etc.
- Débogage des communications réseau : erreurs de socket, erreurs de format de message

---

7. Notions OS USER mises en œuvre

Notion	Utilisation
Sockets TCP	Communication client/serveur bidirectionnelle
Threads	Un thread TCP par client, en parallèle de la boucle SDL
Mutex	Synchronisation entre thread réseau et rendu SDL
Synchronisation non bloquante	Flag + mutex pour signaler un message sans bloquer

---

8. Conclusion

Ce projet a été très formateur : il m'a permis de comprendre concrètement les mécanismes de communication réseau, de thread, de protection concurrente, et de développement SDL.

J'ai trouvé cela stimulant de devoir compléter un code partiellement préécrit, et de le faire fonctionner de bout en bout jusqu'à obtenir un jeu jouable à 4.

Enfin, j'ai gagné en compétence sur la gestion de projets C de taille intermédiaire, et en compréhension des interactions entre processus, sockets, threads et UI.