POLYTECH SORBONNE

SORBONNE UNIVERSITÉ

Projet OS User
Sherlock 13

Sommaire:

Introduction

Dans le cadre de ce travail pratique, nous avons développé Sherlock 13, une application réseau destinée à faire jouer quatre participants à un jeu d'enquête. Inspiré de manière très similaire aux mécanismes du Cluedo, Sherlock 13 repose sur un principe stratégique : parmi treize cartes représentant des personnages, douze sont distribuées à parts égales (trois cartes par joueur), et une carte reste cachée, à découvrir.

Chaque carte est associée à deux ou trois caractéristiques (pipe, couronne, ampoule, etc.), partagées entre plusieurs personnages. En début de partie, chaque joueur compte le total de ces symboles sur ses propres cartes, puis utilise ces informations pour interroger ses adversaires. Deux types de questions sont possibles : d'une part "Qui a au moins un exemplaire de cette caractéristique ?", d'autre part "Combien de cartes de cette caractéristique possédez-vous ?". À partir des réponses collectées, les joueurs affinent leurs hypothèses jusqu'à formuler, lorsqu'ils sont prêts, une accusation contre le personnage qu'ils estiment coupable. Si l'accusation est correcte, le joueur remporte la partie.

Techniquement, Sherlock 13 s'appuie sur une architecture client-serveur classique. Côté serveur (`server.c`), la machine gère les connexions TCP, attribue un identifiant à chaque joueur, mélange et distribue les cartes, et orchestre le déroulement des tours, en traitant les requêtes de type connexion, question globale, question ciblée et accusation. Les clients (`sh13.c`), construits avec SDL2, SDL_image et SDL_ttf, offrent une interface graphique interactive : ils affichent les icônes des objets et des cartes, une grille d'hypothèses pour noter les déductions, et des boutons pour envoyer les différentes commandes au serveur, le tout en synchronisation temps réel.

Ce rapport se compose de trois parties. La première présente l'analyse et la conception de l'architecture client-serveur, ainsi que le protocole de communication mis en place. La seconde décrit l'implémentation technique en détail, autour des fichiers `server.c` et `sh13.c`, en expliquant les structures de données, la machine à états et le traitement des événements SDL. La troisième partie revient sur les résultats des tests, la robustesse et la réactivité du système, avant d'évoquer des pistes d'amélioration pour enrichir l'expérience de jeu.

I. Analyse et conception

Pour répondre aux exigences de communication réseau et d'interaction graphique, nous avons adopté une architecture client-serveur classique. Le serveur (`server.c`) joue le rôle de coordinateur : il écoute les connexions TCP des quatre clients, attribue un identifiant unique à chacun et diffuse la liste des participants. Une fois les quatre joueurs connectés, il mélange aléatoirement les 13 cartes, distribue trois cartes à chaque client et réserve la carte mystère. Il maintient ensuite une machine à états à deux phases : d'abord l'accueil des connexions, puis la gestion du déroulement du jeu. Le serveur reçoit et traite les messages de type O (question globale), S (question ciblée) et G (accusation), renvoie les réponses appropriées à tous les clients et fait tourner le tour du joueur courant.

Côté client (`sh13.c`), chaque instance SDL2 se connecte au serveur, récupère son identifiant et la liste des autres joueurs, puis attend la distribution initiale. L'interface est constituée d'une fenêtre SDL affichant les icônes des objets, les cartes en main et une grille d'hypothèses où le joueur note ses déductions. Les clics sur les objets ou les noms déclenchent la préparation de messages O, S ou G, envoyés au serveur via un thread TCP dédié. Les réponses reçues alimentent la mise à jour dynamique de la grille, garantissant une synchronisation rigoureuse de l'état du jeu sur toutes les interfaces.

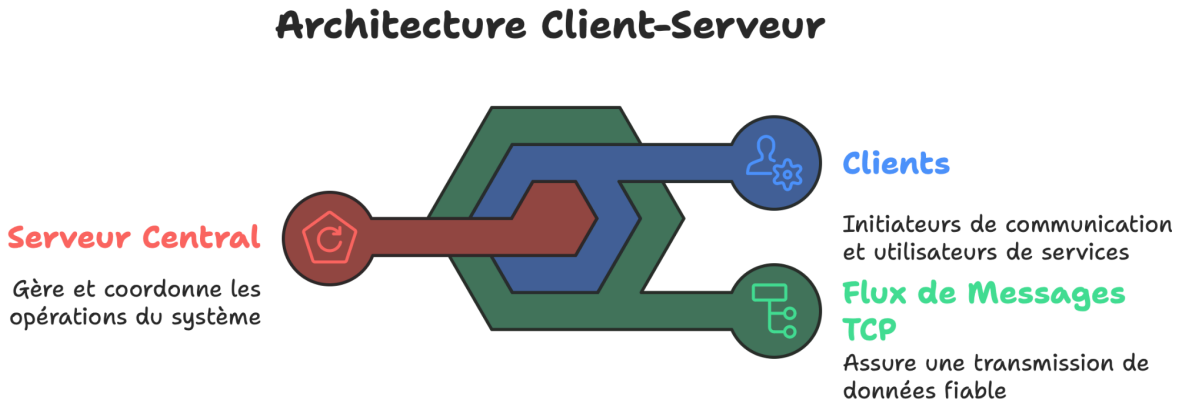

Figure 1. Schéma de l'architecture client-serveur

II. Implémentation technique

2.1 server.c

Le serveur est structuré autour de quelques variables globales : un tableau deck[13] contenant les identifiants de cartes, une matrice tableCartes[4][8] qui recense les caractéristiques de chaque joueur, un compteur joueurCourant et un indicateur fsmServer pour la machine à états.

La fonction melangerDeck() effectue un brassage simple du paquet, tandis que createTable() construit la matrice tableCartes en fonction des trois cartes attribuées à chaque joueur. La communication avec les clients repose sur sendMessageToClient() et broadcastMessage(), qui ouvrent à chaque envoi une connexion TCP, transmettent une chaîne formatée et ferment la socket.

Dans main(), après initialisation du socket serveur et passage en écoute, on entre dans la machine à états :

- État 0 : on collecte les messages ‘C ip port nom’ de chaque client, on envoie à chacun son ‘I id’ puis ‘L liste_noms’, et dès que les quatre joueurs sont présents, on diffuse la distribution ‘D c0 c1 c2’ suivie de ‘V v0…v7’ pour tableCartes, puis on passe à l’état 1.
- État 1 : on boucle en permanence pour traiter les messages ‘O’, ‘S’ et ‘G’. Les questions globales (O) entraînent pour chaque adversaire une réponse V avec valeur 100 ou 0 selon qu’il possède l’objet. Les questions ciblées (S) renvoient un V contenant le nombre exact de cartes objets. Enfin, les accusations (G) sont comparées à la carte mystère ; le serveur annonce “tu as faux” ou “<Nom> a gagné” et passe le tour au joueur suivant en broadcastant ‘M nouveau_id’.

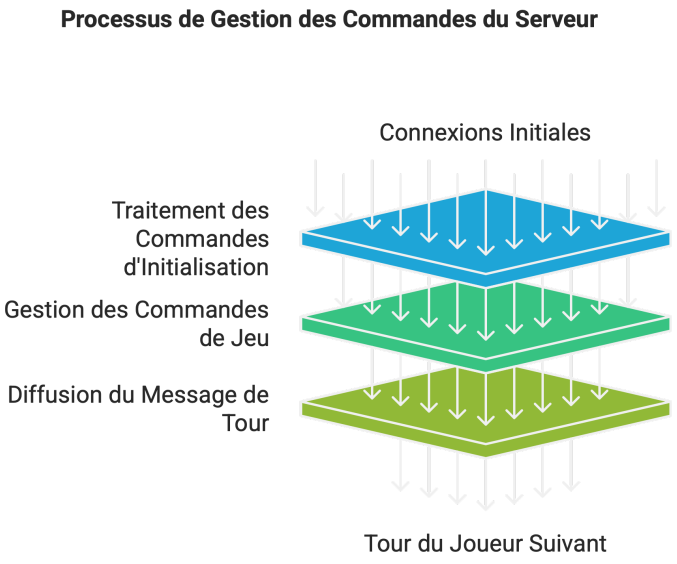

Figure 2. Diagramme de la machine à états du serveur.

2.2 sh13.c

Le client initialise SDL2, SDL_image et SDL_ttf pour créer une fenêtre 1024×768 px et un renderer. Les textures des 13 cartes et des 8 objets, ainsi que celles du bouton Go et du bouton Connect, sont chargées au démarrage. Un thread fn_serveur_tcp écoute en permanence les messages du serveur sur un socket dédié, stocke la chaîne reçue dans gbufferet déclenche le traitement des messages principaux.

La boucle principale utilise SDL_PollEvent : un clic en haut à gauche envoie le message ‘C ip port nom’ pour se connecter, les clics sur les zones d’objets et de noms mettent à jour respectivement objetSel, joueurSel et guiltSel, puis le bouton Go envoie O, S ou G selon l’état courant. Lorsque synchro passe à 1, le switch sur gbuffer[0] décode :

- I : on extrait l’identifiant dans gId
- L : on remplit gNames[0…3]
- D : on stocke les trois cartes dans b[0…2]
- M : on active goEnabled si c’est au tour de gId
- V : trois valeurs signifient une réponse ciblée, sinon on lit huit entiers pour la ligne tableCartes[gId].

L’affichage rafraîchit la grille des hypothèses (tableCartes et guiltGuess), les trois cartes en main, la liste des noms et le bouton Go, dont l’état (actif/inactif) reflète goEnabled.

Figure 3. Interface client après quelques tours, montrant la grille des objets, les cartes en main et le bouton Go activé pour le joueur courant.

III. Résultats

Les essais menés ont validé l'ensemble des fonctionnalités et confirmé la stabilité de l'architecture.

Connexion et initialisation
Dès le lancement du serveur, celui-ci affiche le deck avant et après mélange et la tableCartes initialisée à zéro. Chacun des quatre clients se connecte à son tour en cliquant sur **Connect**, reçoit un identifiant unique et la liste complète des participants. L'interface SDL reflète immédiatement ces mises à jour, sans délai perceptible.

Distribution des cartes
Au quatrième client, le serveur distribue trois cartes à chaque joueur et envoie la ligne de statistiques tableCartes. Les fenêtres des clients affichent instantanément leurs cartes en main et la grille d'indices passe de zéro à sa configuration initiale, prête pour les premières questions.

Phase de questions
Lorsque chaque joueur pose une question ciblée (S), l'interface envoie le bon message au serveur, qui renvoie à tous les clients les réponses au format V. À chaque tour, la grille d'hypothèses se met à jour automatiquement, permettant de suivre visuellement la répartition des caractéristiques.

Phase d'accusation
Les accusations (G) ont été testées en séquence : deux joueurs ont accusé à tort, déclenchant l'affichage "tu as faux", puis le troisième a désigné le bon suspect et a vu s'afficher "<Nom> a gagné". Le tour passe systématiquement au joueur suivant, même en cas d'erreur, ce qui garantit la continuité du jeu.

Robustesse et ergonomie
Tous les échanges TCP ont été traités sans perte ni blocage. Le thread de réception et le verrouillage minimal ont permis une synchronisation fiable des données partagées.
L'interface SDL2 reste fluide, même lors de rafraîchissements fréquents de la grille et des textures.

Ces résultats, illustrés dans les captures d'écran et les logs du terminal, confirment le bon fonctionnement de Sherlock 13, tant côté serveur que côté client.

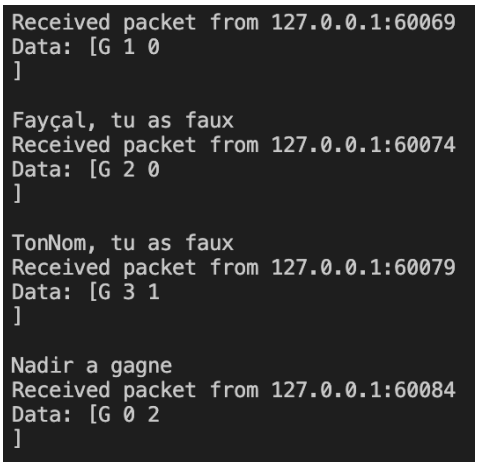

Figure 4. Extrait des logs du serveur montrant la séquence de connexions, questions S et accusations G, avec messages "tu as faux" puis "Nadir a gagné".

Conclusion

Objectifs atteints :

- Distribution aléatoire et équitable des cartes
- Gestion des tours de jeu et synchronisation client-serveur
- Interface graphique SDL2 simple et réactive
- Possibilité de poser des questions et d'accuser en temps réel

Perspectives d'amélioration :

- Ajout d'un chronomètre pour limiter le temps de chaque tour
- Mode spectateur pour observer la partie
- Feedbacks contextuels et aides en interface
- Rejouabilité automatique après fin de partie