

Projet OS USER

Implémentation du jeu Sherlock 13

Table des matières

Table des matières.....	2
Introduction	3
I. Architecture Client-Serveur.....	3
II. Protocole de communication	3
Principaux types de messages :	3
III. Fonctionnement du Serveur (server.c).....	4
a) Accueil des clients	4
b) Distribution des cartes	4
c) Gestion du tour	5
d) Traitement des actions des joueurs	5
e) Diffusion des messages.....	5
IV. Fonctionnement du Client (sh13.c).....	6
a) Interface SDL.....	6
b) Serveur TCP local (fn_serveur_tcp)	6
c) Envoi des messages	7
d) Affichage dynamique.....	7
V. Lancement d'une partie.....	7

Introduction

Sherlock 13 (SH13) est un jeu de déduction multijoueur inspiré de l'univers de Sherlock Holmes. Dans ce projet nous implémentons une version réseau client-serveur du jeu avec une interface graphique réalisée en SDL2. Le but est de retrouver le personnage caché parmi 13 suspects, en posant des questions stratégiques sur les objets liés aux cartes.

I. Architecture Client-Serveur

Le projet repose sur une architecture centralisée :

- **Serveur TCP** → responsable de la gestion du jeu, des connexions et de la synchronisation des états.
- **Clients SDL** → chaque client représente un joueur, avec interface graphique et communication réseau.

Chaque client crée également un thread serveur TCP en local, pour recevoir les messages push du serveur principal.

II. Protocole de communication

Les messages sont envoyés sous forme de chaînes de caractères TCP, selon un protocole très simple (texte brut).

Principaux types de messages :

- C <ip> <port> <nom> → Connexion d'un joueur
- I <id> → Attribution d'ID joueur
- L <nom1> <nom2> <nom3> <nom4> → Liste des joueurs
- D <carte1> <carte2> <carte3> → Distribution des cartes au joueur
- M <id> → Indique quel joueur joue
- G <id> <suspect> → Accusation (Guess)
- O <id> <objet> → Question ouverte
- S <id> <cible> <objet> → Question spécifique
- V <joueur> <objet> <valeur> → Mise à jour de la grille (Visible)
- W <gagnant> → Win

- E <perdant> → Elimination

Le client analyse les messages reçus dans un thread dédié (fn_serveur_tcp) et met à jour les variables globales + affichage SDL.

III. Fonctionnement du Serveur (server.c)

Le serveur central de Sherlock 13 repose sur une architecture mono-thread TCP. Il gère l'intégralité de la logique du jeu, notamment la gestion des connexions clients, la distribution des cartes, l'organisation des tours, le traitement des actions et la diffusion des réponses à l'ensemble des joueurs.

a) Accueil des clients

Le serveur initialise une socket avec socket() et attend les connexions entrantes via accept(). Lorsqu'un client se connecte, il lui envoie un message de type :

C <ip_client> <port_client> <nom_joueur>

Le serveur traite ce message en :

Attribuant un identifiant unique (de 0 à 3) au joueur (**gld** côté client),

Enregistrant les informations reçues (IP, port, nom) dans des structures de type **gNames[]**,

Répondant au client avec un message I contenant son identifiant, suivi d'un message L listant tous les noms des joueurs connectés.

b) Distribution des cartes

Une fois les 4 joueurs connectés, le serveur génère une permutation aléatoire du paquet de 13 cartes personnages. Une carte est retirée pour représenter le coupable. Les 12 autres cartes sont distribuées équitablement entre les joueurs. Chaque joueur reçoit un message de type :

D <carte1> <carte2> <carte3>

Ce message est ensuite utilisé par le client pour remplir son tableau b[3], qui stocke ses cartes personnelles.

c) Gestion du tour

Le serveur conserve une variable **joueurCourant** indiquant l'identifiant du joueur dont c'est le tour. Il informe tous les clients via un message M <id>.

Côté client, la variable **goEnabled** est activée si l'identifiant reçu dans M correspond à gld. Cela permet de rendre interactif le bouton GO uniquement pour le joueur autorisé à jouer.

d) Traitement des actions des joueurs

Le serveur lit les commandes envoyées par les clients à l'aide de read() sur les sockets TCP :

- Commande O : question ouverte à tous les joueurs sur un objet donné. Le serveur compte combien de joueurs possèdent l'objet et envoie un message :

V <id_joueur> <id_objet> <valeur> mis à jour dans **tableCartes[][]**.

- Commande S : question ciblée à un joueur. Le serveur renvoie un message similaire de type V.
- Commande G : accusation. Si le personnage accusé correspond au coupable, le serveur envoie :

W <id_gagnant>

Sinon, le joueur est éliminé :

E <id_perdant>

Ces messages sont envoyés en TCP aux serveurs locaux des clients.

e) Diffusion des messages

Le serveur envoie ses réponses aux adresses **gClientIpAddress** et **gClientPort** des joueurs. Ces informations ont été fournies au moment de la connexion. Il contacte le mini-serveur TCP (fn_serveur_tcp) en cours d'exécution sur chaque client pour transmettre les messages.

IV. Fonctionnement du Client (sh13.c)

Chaque client SDL représente un joueur connecté à la partie. Il combine une interface graphique interactive (basée sur SDL2) et une communication réseau bidirectionnelle pour envoyer et recevoir des messages du serveur.

a) Interface SDL

L'interface est construite avec **SDL_CreateWindow**, **SDL_CreateRenderer** et **SDL_Texture**. Elle affiche :

- Les 13 cartes personnages à l'aide du tableau `deck[13]`,
- Les 8 objets via `objet[8]`,
- Les boutons GO et CONNECT (`gobutton`, `connectbutton`),
- La grille de déduction `tableCartes[4][8]`,
- Le tableau des suspects `guiltGuess[13]`.

L'utilisateur interagit avec la fenêtre via des clics de souris. Les coordonnées (`mx`, `my`) déterminent s'il clique sur un joueur, un objet, un suspect ou un bouton. Selon les sélections, les variables `joueurSel`, `objetSel` et `guiltSel` sont mises à jour.

b) Serveur TCP local (`fn_serveur_tcp`)

Chaque client lance un thread (`pthread_create`) qui initialise une socket TCP locale avec `bind()` sur le port fourni en argument (ex. 5001). Ce thread appelle `accept()` en boucle pour recevoir les messages envoyés par le serveur principal.

Les messages reçus sont stockés dans `gbuffer` puis analysés pour mettre à jour :

- `gId` (identifiant joueur)
- `gNames[4]` (noms des joueurs)
- `b[3]` (cartes personnelles)
- `tableCartes[4][8]` (informations objets/joueurs)
- `guiltGuess[13]` (hypothèses locales du joueur)

La synchronisation est gérée via la variable `synchro` et un `pthread_mutex_t` mutex.

c) Envoi des messages

Les actions du joueur sont traduites en chaînes de caractères et envoyées via des connexions TCP éphémères. Chaque appel à **sendMessageToServer()** ouvre une socket, construit le message (ex. C, S, O, G), l'envoie au serveur, puis ferme la connexion.

d) Affichage dynamique

Le rendu graphique est mis à jour à chaque tour via `SDL_RenderPresent()`. Les couleurs changent dynamiquement selon les variables d'état (`goEnabled`, `connectEnabled`, etc.) pour guider l'interaction du joueur. Les textes sont générés avec `TTF_RenderText_Solid` et les textures d'objets/cartes sont manipulées avec `SDL_RenderCopy`.

V. Lancement d'une partie

- Compilation du projet

Prérequis:

Installer les bibliothèques SDL2 et extensions :

sudo apt update

sudo apt install libsdl2-dev libsdl2-image-dev libsdl2-ttf-dev

Commande de compilation

Pour compiler le client SDL :

gcc -o sh13 sh13.c `sdl2-config --cflags --libs` -lSDL2_image -lSDL2_ttf

Pour compiler le serveur :

gcc -o server server.c

- Lancer une partie

1. Lancer le serveur

./server 2000

```
0005813.4025000 main -> main
papa@papa-HP-Laptop-15s-eq2xxx:~/Downloads/sh13_etu/Projet-Sherlock-13$ ./server 2000
0 Sebastian Moran
1 irene Adler
2 inspector Lestrade
3 inspector Gregson
4 inspector Baynes
5 inspector Bradstreet
6 inspector Hopkins
7 Sherlock Holmes
8 John Watson
9 Mycroft Holmes
10 Mrs. Hudson
11 Mary Morstan
12 James Moriarty
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
8 John Watson
0 Sebastian Moran
1 irene Adler
10 Mrs. Hudson
12 James Moriarty
6 inspector Hopkins
2 inspector Lestrade
5 inspector Bradstreet
7 Sherlock Holmes
3 inspector Gregson
11 Mary Morstan
4 inspector Baynes
9 Mycroft Holmes
01 01 02 00 00 01 01 02
02 01 00 01 00 01 01 01
01 01 02 02 01 00 01 00
00 01 01 02 02 01 00 00
Received packet from 127.0.0.1:45774
Data: [C 127.0.0.1 3000 Talla
]

COM=C ipAddress=127.0.0.1 port=3000 name=Talla
0: 127.0.0.1 03000 Talla
id=0
[]
```

2. Lancer les clients

Lancer 4 clients dans des terminaux séparés :

./sh13 127.0.0.1 2000 127.0.0.1 5001 Player1

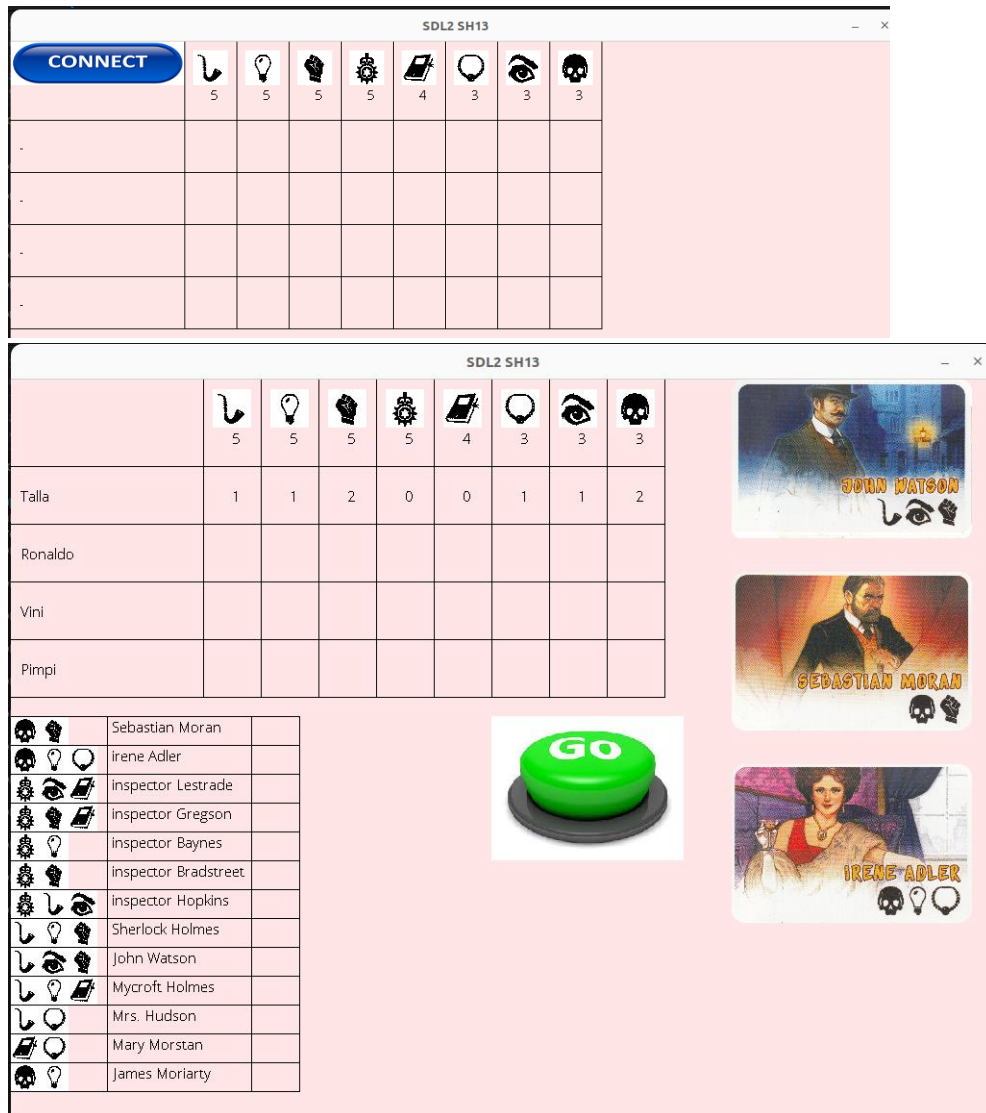
./sh13 127.0.0.1 2000 127.0.0.1 5002 Player2

./sh13 127.0.0.1 2000 127.0.0.1 5003 Player3

./sh13 127.0.0.1 2000 127.0.0.1 5004 Player4

```
papa@papa-HP-Laptop-15s-eq2xxx:~/Downloads/sh13_etu/Projet-Sherlock-13$ ./sh13 127.0.0.1 2000 127.0.0.1 3000 Talla
Sans=0x5da817d407f0
Creation du thread serveur tcp !
consomme |I 0
|
consomme |L Talla - - -
[]
```


Une fois connectés, cliquez sur le bouton "**Connect**" dans l'interface SDL pour rejoindre officiellement la partie.



Règles du jeu

Il y a 13 cartes personnage, 12 sont distribuées (3 par joueur), 1 est le coupable à deviner.

Chaque personnage possède 2 ou 3 objets caractéristiques.

Le but est de deviner qui est le coupable en posant des questions aux autres joueurs.

Actions possibles :

Poser une question à tous : "Qui a au moins un objet [X] ?" (commande O)

→ Sélectionner l'objet en question puis appuyer sur Go sur la fenetre SDL

Poser une question ciblée : "Combien de [X] as-tu ?" (commande S)

→ Sélectionner l'objet + la personne puis sur GO

Accuser un personnage si vous pensez avoir trouvé le coupable (commande G)

Fin du jeu :

Si un joueur accuse correctement → il gagne la partie (message W reçu)

S'il se trompe → il est éliminé (message E)

Ps: L'Élimination n'est pas encore implémentée donc le jeu continue jusqu'à qu'il y ait un gagnant.

PROBLEMS 1

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

```

]

Talla, tu as Faux
Received packet from 127.0.0.1:55726
Data: [G 1 1
]

Ronaldo, tu as Faux
Received packet from 127.0.0.1:40924
Data: [G 2 2
]

Vini, tu as Faux
Received packet from 127.0.0.1:53186
Data: [G 3 3
]

Pimpi, tu as Faux
Received packet from 127.0.0.1:34254
Data: [G 0 5
]

Talla, tu as Faux
Received packet from 127.0.0.1:60456
Data: [G 1 6
]

Ronaldo, tu as Faux
Received packet from 127.0.0.1:48520
Data: [G 2 7
]









Vini, tu as Faux
Received packet from 127.0.0.1:40508
Data: [G 3 8
]




































Pimpi, tu as Faux
Received packet from 127.0.0.1:41628
Data: [G 0 9
]

Talla a gagne
[

```

SDL2 SH13

								
	5	5	5	5	4	3	3	3
Talla	1	1	2	0	0	1	1	2
Ronaldo	*	*						1
Vini	*	*	2	2		0		
Pimpi		*						

 	Sebastian Moran	
 	Irene Adler	
 	Inspector Lestrade	
 	Inspector Gregson	
 	Inspector Baynes	
 	Inspector Bradstreet	
 	Inspector Hopkins	
 	Sherlock Holmes	
 	John Watson	
 	Mycroft Holmes	
 	Mrs. Hudson	
 	Mary Morstan	
 	James Moriarty	