PROJET SHERLOCK 13

Antoine HEBERT
Professeur : Thibault Hilaire

Socket

Le programme utilise des sockets TCP pour communiquer entre le serveur et les clients. Le serveur attend les connexions, reçoit les messages des joueurs (connexion, actions de jeu), et leur répond en envoyant des données (cartes, tour de jeu, résultats). Chaque message est échangé via un socket temporaire ouvert, utilisé, puis fermé. Côté client, un thread écoute les messages du serveur et met à jour l'interface en conséquence.

Threads

Le programme client utilise un thread pour écouter en permanence les messages du serveur. Ce thread TCP tourne en parallèle de l'interface graphique, ce qui permet au joueur de continuer à interagir avec le jeu pendant que les données réseau sont reçues en arrière-plan. Cela évite que l'interface ne se bloque en attendant une réponse.

Mutex

Le mutex agit comme une sécurité, avant de lire ou écrire une variable partagée, un thread le verrouille, puis le libère après modification. Cela évite les comportements imprévisibles causés par l'accès concurrent à la même donnée.

SERVEUR :

Introduction au code serveur

Le programme commence par une phase d'initialisation : le serveur ouvre un socket TCP sur le port spécifié, mélange les cartes grâce à melangerDeck, puis initialise la table des objets avec createTable. Ensuite, les joueurs se connectent un par un en envoyant une commande C. Le serveur enregistre leur IP, port et nom, leur attribue un identifiant (I), puis diffuse la liste complète des joueurs (L). Lorsque les 4 joueurs sont connectés, le jeu démarre : chacun reçoit ses 3 cartes (D) et sa ligne d'objets (V), puis le serveur indique qui commence avec le message J.

Chaque joueur peut effectuer une action en envoyant un message spécifique. Le message G permet de faire une accusation : si elle est correcte, le joueur gagne (W), sinon le tour passe (M). Le message O permet de poser une question générale à tous : "Quelqu'un possède-t-il l'objet x ?", et le serveur répond avec un message V. Enfin, le message S cible un joueur pour savoir s'il possède un certain objet, et là encore, le serveur répond via V. Après chaque action, le serveur passe automatiquement au joueur suivant (M).

Toutes les communications se font en TCP, soit individuellement via sendMessageToClient, soit collectivement via broadcastMessage.

Le serveur attend que 4 joueurs se connectent. Une cela fait, il mélange les cartes avec la fonction melangerDeck, et distribue 3 cartes à chaque et calcule les objets associés avec la fonction createTable.

```
// On envoie ses cartes au joueur 0, ainsi que la ligne qui lui correspond dans tableCartes

sprintf(reply, "D %d %d %d", deck[0], deck[1], deck[2]);
sendMessageToClient(tcpClients[0].ipAddress, tcpClients[0].port, reply);
for (int j = 0; j < 8; j++)
{
    sprintf(reply, "V %d %d %d", 0, j, tableCartes[0][j]);
    sendMessageToClient(tcpClients[0].ipAddress, tcpClients[0].port, reply);
}
```

sprintf(reply, "D %d %d %d", deck[0], deck[1], deck[2]);
Le serveur crée un message avec les 3 carte et les envois au client avec la ligne suivante :
sendMessageToClient(tcpClients[0].ipAddress, tcpClients[0].port, reply);

La boucle va envoyer 8 messages pour chaque type d'objet.
Ensuite, pour chaque joueur, il envoie :

- ses 3 cartes
- les objets qu'il possède (selon ses cartes)
- puis informe tous les joueurs de qui commence le tour

Les clients reçoivent les informations et les utilisent pour afficher :

- leurs cartes
- leurs objets

Dans le code on retrouvera 4 fois le même type de code pour chaque joueur(client).
Il est possible de faire une boucle pour ne pas à avoir à répéter le code 4 fois en changeant le numéro du joueur.

```
// On envoie ses cartes au joueur 1, ainsi que la ligne qui lui correspond dans tableCartes

sprintf(reply, "D %d %d %d", deck[3], deck[4], deck[5]);
sendMessageToClient(tcpClients[1].ipAddress, tcpClients[1].port, reply);
for (int j = 0; j < 8; j++)
{
    sprintf(reply, "V %d %d %d", 1, j, tableCartes[1][j]);
    sendMessageToClient(tcpClients[1].ipAddress, tcpClients[1].port, reply);
}
```

```
// On envoie ses cartes au joueur 2, ainsi que la ligne qui lui correspond dans tableCartes

sprintf(reply, "D %d %d %d", deck[6], deck[7], deck[8]);
sendMessageToClient(tcpClients[2].ipAddress, tcpClients[2].port, reply);

for (int j = 0; j < 8; j++)
{
    sprintf(reply, "V %d %d %d", 2, j, tableCartes[2][j]);
    sendMessageToClient(tcpClients[2].ipAddress, tcpClients[2].port, reply);
}
```

```
// On envoie ses cartes au joueur 3, ainsi que la ligne qui lui correspond dans tableCartes

sprintf(reply, "D %d %d %d", deck[9], deck[10], deck[11]);
sendMessageToClient(tcpClients[3].ipAddress, tcpClients[3].port, reply);

for (int j = 0; j < 8; j++)
{
    sprintf(reply, "V %d %d %d", 3, j, tableCartes[3][j]);
    sendMessageToClient(tcpClients[3].ipAddress, tcpClients[3].port, reply);
}
```

Chaque message reçu par le serveur correspond à une action d'un joueur :

- deviner le coupable (G)
- poser une question générale (O)
- interroger un joueur spécifique (S)

Le serveur traite chaque cas et répond à tous les joueurs.

Pour deviner le coupable (G) le joueur propose une carte comme coupable.
Le serveur vérifie si c'est bien la carte du coupable
Si c'est correct le joueur gagne.
Si c'est faux :
Le tour passe au joueur suivant et le serveur envoie à tous le message M x pour indiquer quel joueur doit jouer maintenant.

```
case 'G':
{
    int guess;
    sscanf(buffer, "G %d", &guess);
    if (guess == deck[12])
    {
        sprintf(reply, "W %d", joueurCourant);
        broadcastMessage(reply);
        fsmServer = 0;
    }
    else
    {
        joueurCourant = (joueurCourant + 1) % 4;
        sprintf(reply, "M %d", joueurCourant);
        broadcastMessage(reply);
    }
    break;
}
```

Pour poser une question générale (O) le joueur demande si au moins un autre joueur possède l'objet x.
Le serveur récupère l'indice de l'objet sur lequel porte la question, puis il va parcourir les 4 joueurs sauf le joueur qui demande.
Pour chaque autre joueur, on vérifie s'il possède cet objet et tout le monde reçoit la réponse.

```
case 'O':
{
    int symbole;
    sscanf(buffer, "O %d", &symbole);
    for (int i = 0; i < 4; i++)
    {
        if (i != joueurCourant)
        {
            sprintf(reply, "V %d %d %d", i, symbole, tableCartes[i][symbole] > 0 ? 100 : 0);
            broadcastMessage(reply);
        }
    }
    joueurCourant = (joueurCourant + 1) % 4;
    sprintf(reply, "M %d", joueurCourant);
    broadcastMessage(reply);
    break;
}
```

Pour interroger un joueur spécifique (S) un joueur pose une question ciblée à un autre joueur.

Le serveur lit le joueur à qui on pose la question et l'objet choisis, et le serveur va vérifier dans la table des objets combien le joueur cible possédé cet objet et il envoie la réponse à tous les joueurs.

```
case 'S':
{
    int cible, symbole;
    sscanf(buffer, "S %d %d", &cible, &symbole);
    sprintf(reply, "V %d %d %d", cible, symbole, tableCartes[cible][symbole]);
    broadcastMessage(reply);
    joueurCourant = (joueurCourant + 1) % 4;
    sprintf(reply, "M %d", joueurCourant);
    broadcastMessage(reply);
    break;
}
```

CLIENT :

Introduction au code client

Ce programme est un client graphique pour le jeu “Sherlock 13”, utilisant SDL2. Il se connecte à un serveur TCP, reçoit les données de jeu, et permet au joueur d’interagir via une interface visuelle. À l’exécution, le programme initialise la connexion avec les paramètres donnés (IP/port serveur, IP/port client, nom du joueur), charge les images des cartes et objets, puis crée une fenêtre SDL pour afficher l’interface du jeu.

Un thread TCP est lancé en parallèle pour écouter les messages entrants du serveur (comme l’ID du joueur, la liste des joueurs, ses cartes ou les mises à jour de la table). L’utilisateur interagit avec la souris pour sélectionner un joueur, un objet ou faire une accusation. Les actions sont traduites en messages (G, O, S) envoyés au serveur. Le programme affiche dynamiquement les cartes, objets, et les suppositions à l’écran, tout en mettant à jour la table selon les réponses reçues (V).
Le client permet à un joueur de participer à une partie en réseau avec une interface interactive et une communication constante avec le serveur central.

```
else if ((mx>=500) && (mx<700) && (my>=350) && (my<450) && (goEnabled==1))
{
    printf("go! joueur=%d objet=%d guilt=%d\n",joueurSel, objetSel, guiltSel);
    if (guiltSel!=-1)
    {
        sprintf(sendBuffer,"G %d %d",gId, guiltSel);

        sendMessageToServer(gServerIpAddress, gServerPort, sendBuffer);
    }
    else if ((objetSel!=-1) && (joueurSel==-1))
    {
        sprintf(sendBuffer,"O %d %d",gId, objetSel);

        sendMessageToServer(gServerIpAddress, gServerPort, sendBuffer);

    }
    else if ((objetSel!=-1) && (joueurSel!=-1))
    {
        sprintf(sendBuffer,"S %d %d %d",gId, joueurSel,objetSel);

        sendMessageToServer(gServerIpAddress, gServerPort, sendBuffer);

    }
}
```

Quand un joueur clique sur le bouton “Go”, il vérifie ce que le joueur a sélectionné, et in renvoie l’un des 3 choix en fonction de ce que le joueur à choisi :

- deviner le coupable (G)
  Si un personnage est sélectionné il est peut-être le coupable.
  Il envoie un message de type G au serveur avec l’identifiant du personnage suspecté.
  Et le serveur dira si c’est vrai ou faux.
- poser une question générale (O)
  Si un objet est sélectionné et aucun joueur n’est ciblé, le joueur pose une question à tous si ils ont l’objet.
  Et le serveur renverra qui possède l’objet.
- interroger un joueur spécifique (S)

Si un objet et un joueur sont tous deux sélectionnés, le joueur pose une question en visant un joueur spécifique.
Il envoie un message de type S avec les identifiants du joueur et de l'objet.
Et le serveur renverra si le joueur sélectionné possède l'objet.

Chaque fois que le client reçoit un message du serveur, il le traite selon son type (I, L, D, M, V).

case 'I'
Ce code permet au joueur de recevoir et d'enregistrer son identifiant quand il se connect e au serveur.
On extrait l'ID du message "I x" et on le stocke dans la variable gId et affiche cet ID dans la console pour que le joueur le voie.

```
    // Message 'I' : le joueur recoit son Id
    case 'I':
        sscanf(gbuffer, "I %d", &gId);
        printf("Mon ID est : %d\n", gId);

        break;
```

case 'L'
On lit dans le message reçu les 4 noms des joueurs et on les stocke dans le tableau gNames et ensuite on affiche la liste des joueurs dans la console.
Ce code permet au client de recevoir et d'affiché les noms des 4 joueurs connectés.

```
// Message 'L' : le joueur recoit la liste des joueurs
case 'L':
    sscanf(gbuffer, "L %s %s %s %s", gNames[0], gNames[1], gNames[2], gNames[3]);
    printf("Liste des joueurs : %s, %s, %s, %s\n", gNames[0], gNames[1], gNames[2], gNames[3]);

    break;
```

case 'D'
Ce code permet au client de recevoir ses 3 cartes de jeu et de les affichés.
Le message contient 3 numéros de carte qui sont stockés dans le tableau b, et on affiche les 3 cartes dans la console pour le joueur.

```
// Message 'D' : le joueur recoit ses trois cartes
case 'D':
    sscanf(gbuffer, "D %d %d %d", &b[0], &b[1], &b[2]);
    printf("Mes cartes : %d, %d, %d\n", b[0], b[1], b[2]);

    break;
```

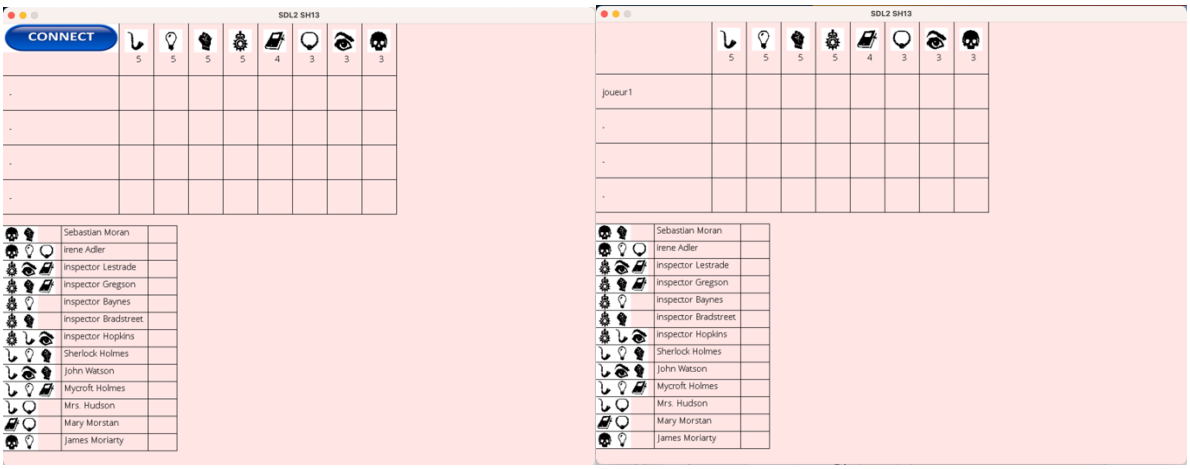
case 'M'
Le code permet au client de dire si c'est à lui de jouer.
On lit le numéro du joueur qui doit jouer, et on affiche le joueur dans la console.

```
case 'M':
{
    int joueurActif;
    sscanf(gbuffer, "M %d", &joueurActif);
    printf("C'est au joueur %d de jouer\n", joueurActif);
    goEnabled = (joueurActif == gId) ? 1 : 0;
    break;
}
```

case 'V'
Le code permet au joueur de recevoir des informations et de mettre a jour la grille tableCartes.
Le message reçu contient une position (i et j) dans le tableau et une valeur, et on met à jour cette case du tableau avec la valeur reçue.

```
// Message 'V' : le joueur recoit une valeur de tableCartes
case 'V':
    int i, j, val;
    sscanf(gbuffer, "V %d %d %d", &i, &j, &val);
    tableCartes[i][j] = val;
    break;
```

On peut voir sur la capture d'écran, en bas à gauche, que la console du joueur 1 a bien été lancée. Ensuite, lorsqu'on clique sur le bouton "connect", le joueur se connecte au serveur, ce qui est confirmé dans la fenêtre en bas à droite : le joueur 1 apparaît comme connecté.



Une fois les 4 joueurs connectés au serveur, les cartes sont distribuées automatiquement et le bouton Go apparaît uniquement pour le joueur dont c'est le tour de jouer.

Une fois que le bouton Go est cliqué, le tour passe automatiquement au joueur suivant.
Conclusion :

Ce projet m'a permis de mieux comprendre comment fonctionne la communication entre un serveur et plusieurs clients en réseau. Grâce aux sockets, les messages passent bien entre les joueurs et le serveur, et avec les threads, le jeu reste fluide sans bloquer l'interface. Une fois que les 4 joueurs sont connectés, tout se lance automatiquement : les cartes sont distribuées, le bouton "Go" s'affiche pour le joueur dont c'est le tour, et chaque action déclenche une réponse du serveur. L'interface avec SDL permet de bien visualiser ce qu'il se passe. En résumé, le jeu fonctionne correctement et respecte les règles de Sherlock 13.