Rapport

13/04/2025

OS User

Table des matières

I- Fonctionnement du programme

Architecture client-serveur

L'architecture se compose de deux entités principales :

Le serveur (server.c) :

- Gérer les connexions des joueurs.
- Mélanger et distribuer les cartes aux joueurs.
- Coordonner les tours de jeu et synchroniser les états entre les joueurs.

Le serveur agit comme un point central de communication. Il reçoit les actions des joueurs, traite les informations selon les règles du jeu, et diffuse les résultats à l'ensemble des clients.

Les clients (sh13.c) :

- Fournir une interface graphique pour que chaque joueur puisse interagir avec le jeu.
- Envoyer les actions au serveur.
- Afficher les réponses et les hypothèses mises à jour en temps réel.

Chaque client est lié au serveur par un socket TCP. Il affiche l'état du jeu à partir des messages reçus et permet au joueur d'effectuer des actions à l'aide de clics sur les différents éléments de l'interface.

Logique de communication et synchronisation

Le jeu repose sur une communication réseau fluide et une synchronisation rigoureuse entre les clients et le serveur. Chaque action effectuée par un joueur est :

1. Envoyée au serveur.
2. Traitée par le serveur selon les règles du jeu.
3. Diffusée à tous les clients pour garantir un état cohérent et synchronisé.

Cette architecture permet de réaliser un jeu multijoueur fonctionnel où les interactions des joueurs sont gérées efficacement par un serveur central.

II- Construction du programme

server.c

Le fichier server.c est responsable de la gestion du serveur multijoueur pour le jeu Sherlock Holmes Consulting Detective. Il gère les connexions des joueurs, la distribution des cartes, ainsi que la logique principale du jeu en réseau.

Structures et variables principales

● Structure _client : Cette structure représente les informations sur un client connecté au serveur.
- ipAddress : Adresse IP du client.
- port : Port utilisé par le client.
- name : Nom du joueur.
- Tableau tcpClients[4] : Stocke les informations des quatre joueurs connectés.

● Variables globales :
- deck[13] : Contient les identifiants des cartes, indexées de 0 à 12.
- tableCartes[4][8] : Représente les informations sur les objets possédés par chaque joueur. Chaque ligne correspond à un joueur et chaque colonne à un objet.
- joueurCourant : Pour savoir c'est à quel joueur de jouer.
- fsmServer : Deux états principaux sont utilisés :
    - 0 : En attente de connexions des joueurs.
    - 1 : En phase de jeu.

Fonctions principales

a) Fonctions utilitaires

- void error(const char *msg) : Affiche un message d'erreur et interrompt l'exécution.
- void melangerDeck() : Mélange le deck de cartes de manière aléatoire.

b) Création et gestion des cartes

void createTable() :
- Distribue 3 cartes du *deck* à chaque joueur.
- Associe les indices des cartes à des caractéristiques dans *tableCartes* (par exemple, les inspecteurs, Sherlock Holmes, ou Moriarty).
- La dernière carte (indice 12) représente le coupable, et n'est pas attribuée aux joueurs. C'est elle que les joueurs vont devoir trouver.

c) Communication avec les clients

*void sendMessageToClient(char *clientip, int clientport, char *mess)* :
- Établit une connexion avec un joueur en utilisant son adresse IP et son port.
- Envoie un message au joueur via un socket TCP.

*void broadcastMessage(char *mess) :*
- Envoie un message à tous les joueurs.

d) Logique serveur

*int main(int argc, char *argv[])* :
- Initialise le socket du serveur sur le port spécifié en argument.
- Utilise une boucle infinie pour accepter les connexions entrantes des clients et traiter les messages reçus.
- Gère les états de la machine à états (*fsmServer*) :
    - **État 0** : Attente des connexions des 4 joueurs. Chaque joueur se connecte en envoyant un message de type 'C'. Une fois tous les joueurs connectés, les cartes sont distribuées, et la partie commence.
    - **État 1** : Traitement des messages de jeu :
    - 'G' : Gestion des accusations des joueurs (vérifie si le coupable est correctement identifié).
    - 'O' : Permet à un joueur de demander des informations sur un objet auprès des autres joueurs.
    - 'S' : Permet à un joueur de poser une question ciblée à un autre joueur.
- Passe le tour au joueur suivant une fois qu'une action est terminée.

Traitement des messages reçus

Chaque message reçu par le serveur est interprété en fonction de son premier caractère (*buffer[0]*) :
- 'C' (Connexion) : - Enregistre les informations du joueur (adresse IP, port et nom).
- Envoie un message personnel au joueur pour lui communiquer son ID.
- Diffuse la liste des joueurs connectés à tous les clients.
- Si 4 joueurs sont connectés, démarre la partie en distribuant les cartes et les informations d'objets.

- 'G' (Accusation) : - Vérifie si le joueur a accusé correctement le coupable.
- Informe tous les clients du résultat et passe le tour au joueur suivant.

- 'O' (Objet demandé) : - Un joueur demande si un objet spécifique est possédé par les autres joueurs.
- Les réponses sont diffusées à tous les clients, et le tour passe au joueur suivant.

- 'S' (Question ciblée) : - Permet à un joueur de poser une question ciblée à un autre joueur sur un objet particulier.
- La réponse est diffusée à tous les clients.

sh13.c

Le fichier *sh13.c* implémente le client d'un jeu multijoueur inspiré de l'univers de Sherlock Holmes. Le programme permet à un joueur de se connecter à un serveur, de participer à des interactions basées sur des cartes et des objets, et de visualiser l'état du jeu via une interface graphique créée avec SDL.

Principales Fonctions :

Connexion et Communication

*sendMessageToServer(char *ipAddress, int portno, char *mess)* Cette fonction établit une connexion TCP avec le serveur, envoie un message (comme des actions ou des mises à jour), et ferme la connexion. Elle est utilisée pour transmettre des informations importantes telles que les actions des joueurs ou leurs sélections.

Gestion des Événements SDL

La boucle principale du programme repose sur *SDL_PollEvent*, qui capture et traite les événements utilisateur (clics de souris, fermeture de fenêtre, etc.).
Exemple d'événement géré : un clic sur un bouton "connect" envoie un message au serveur avec les informations du joueur via *sendMessageToServer*.

Affichage Graphique

Initialisation : Les fenêtres et éléments graphiques (textures, boutons, cartes, etc.) sont initialisés via SDL, SDL_Image et SDL_TTF. Exemple : *SDL_CreateWindow* et *SDL_CreateRenderer* créent respectivement la fenêtre et le moteur de rendu.
Rendu des objets : Les éléments du jeu (cartes, objets, boutons) sont affichés avec des fonctions comme *SDL_RenderCopy*. Chaque objet ou carte est positionné à l'écran avec des coordonnées précises.

Synchronisation et Mise à Jour

*fn_serveur_tcp(void *arg)* Un thread dédié à la communication avec le serveur écoute les messages entrants (comme les ID des joueurs, les cartes distribuées ou les mises à jour de la table). Ces messages déclenchent des actions locales, comme l'affichage d'informations à l'écran ou l'activation de boutons.

Traitement des Messages

Les messages reçus du serveur sont traités selon leur type (indiqué par la première lettre du message) :

- I : Le joueur reçoit son ID et l'enregistre dans la variable *gId*.
- L : La liste des joueurs connectés est mise à jour.
- D : Les cartes du joueur sont stockées dans le tableau *b*.
- M : Indique quel joueur est actif pour jouer, et active le bouton "go" si c'est au tour de ce joueur.

Interactions Utilisateur

Les clics de souris permettent au joueur de sélectionner des cartes, des objets ou d'effectuer des actions comme accuser un personnage ou poser une question. Les choix sont traduits en messages envoyés au serveur.

Données du Jeu

Les tableaux *tableCartes* et *guiltGuess* stockent respectivement les informations sur les objets des autres joueurs et les suppositions du joueur sur le coupable.