Jean-Loup Berkens
Lucien Coudert
Dimitri Vercaigne

Jeu de Tarot Africain

Introduction:

Ce document a pour objectif d'expliquer le code derrière ce jeu de Tarot Africain utilisant une architecture client-serveur. Il a pour objectif de permettre, au besoin, de déboguer et/ou améliorer ce code. Nous avons mis en place ce jeu sur deux protocoles réseaux: TCP et UDP. Les codes sources étant quasiment les mêmes, nous expliquerons la logique derrière ce jeu sur la version TCP puis nous détaillerons les changements faits pour la version UDP.
Enfin nous présenterons avec des aides de capture d'écran les résultats attendus. Mais avant tout cela nous présenterons la manière dont nous avons choisis de communiquer les informations au sein de notre système.
Pour mettre en place ces protocoles réseaux nous nous sommes basés sur le travail présenté ici: Les sockets en C.

Protocole de communication:

Afin de pouvoir communiquer efficacement entre le serveur et le client nous avons décidé de mettre en place un protocole spécifique. C'est-à-dire que le premier caractère du buffer permet à celui qui le reçoit de déterminer de ce qu'il s'agit. Cela fonctionne suivant la table de correspondance suivante:

Code(buffer[0])	Client	Serveur
1	reçoit des cartes	reçoit un pseudo
2	reçoit toutes les annonces	reçoit une annonce faite par un joueur
3	reçoit tous les pseudos	
4	reçoit l'état de la table	reçoit une carte jouée par un joueur
5	reçoit les scores	

Ensuite afin d'éviter de transmettre des 0 qui est le caractère de fin de chaîne nous avons transmis les scores et les annonces avec des +1 et les cartes avec des +2.

TCP:

Nous commencerons par détailler le fonctionnement du serveur. La partie serveur est découpée en trois programmes avec chacun un fichier header associé contenant toute les déclarations utiles: server.c, jeuCoteServer, cartes.c.
Pour **cartes.c**:

Ce programme gère une liste chaînée représentant un paquet de cartes. Chaque carte est identifiée par un numéro unique (`numéroCarte`). Les fonctionnalités incluent la création, l'ajout, la suppression, l'affichage des cartes du paquet, ainsi que le tirage aléatoire d'une carte. La structure de données utilisée est la suivante:

```
typedef struct carte {
    int numeroCarte; // Numéro unique de la carte
    struct carte* suivant; // Pointeur vers la carte suivante dans la liste
} carte;
```

Cette structure représente une carte individuelle dans une liste chaînée. Les fonctionnalités de code sont les suivantes:

1. Création d'une nouvelle carte: carte* newCarte(int n);

Description

- Crée une nouvelle carte avec un numéro donné (n).
- Alloue dynamiquement de la mémoire pour la carte.
- Retourne un pointeur vers cette nouvelle carte.

Gestion des Erreurs

- Si l'allocation de mémoire échoue, le programme affiche un message d'erreur et termine.

2. Ajout d'une carte dans le paquet: carte* insererCarte(carte* paquet, int n);

Description

- Ajoute une carte au paquet, en l'insérant à la fin de la liste chaînée.
- Si le paquet est vide (`NULL`), la carte ajoutée devient la première du paquet.

3. Initialisation d'un paquet: carte* initPaquet();

Description

- Initialise un paquet contenant 22 cartes, numérotées de 0 à 21.
- Utilise la fonction `insererCarte` pour ajouter chaque carte au paquet.

4. Suppression d'une carte du paquet: carte* supprimerCarte(carte* paquet, int n);

Description

- Supprime la carte portant le numéro n du paquet.
- Si la carte supprimée est la première de la liste, le pointeur du paquet est mis à jour pour pointer sur la carte suivante.
- Si la carte n'existe pas, le paquet reste inchangé.

5. Tirer une carte au hasard:int tirerUneCarte(carte** paquet)

Description

- Sélectionne une carte aléatoirement dans le paquet, grâce à la fonction `rand`.
- Supprime la carte tirée du paquet via la fonction `supprimerCarte`.
- Retourne le numéro de la carte tirée.
- Retourne `-1` si le paquet est vide.

Remarque

- Cette fonction nécessite un pointeur sur le paquet pour pouvoir le modifier directement.

6. Affichage du paquet: void afficherPaquet(carte* paquet);

Description

- Parcourt la liste chaînée et affiche les numéros de toutes les cartes du paquet, séparés par des espaces.
- Termine l'affichage par un retour à la ligne.

7. Libération de la mémoire: void freePaquet(carte* paquet);

Description

- Libère la mémoire allouée pour toutes les cartes du paquet.
- Parcourt la liste chaînée en supprimant chaque élément un par un.

Pour **server.c:**

Les variables globales sont:

1. `clients[MAX_CLIENTS]` : Un tableau contenant les informations des clients connectés. Chaque élément de ce tableau est une structure `Client` qui contient :
    - La socket client (`sock`).
    - Le pseudo associé à ce client (`name`).
2. `actual` : Nombre actuel de clients connectés.

Les fonctionnalités de `server.c`:

Fonction principale : void app(void);

Description : La fonction `app` représente la boucle principale du serveur. Elle gère :

- Les nouvelles connexions des clients.
- La réception et l'envoi de messages.
- La déconnexion des clients.

Étapes principales **:**

1. Initialisation de la connexion avec `init_connection()` pour mettre en place une socket serveur.
2. Surveillance des événements grâce à `select()` :
    - Nouvelle connexion client sur la socket serveur.
    - Messages ou déconnexions des clients existants.
    - Commande serveur (entrées clavier via `STDIN`).
3. Traitement des différents événements détectés :
    - Pour une nouvelle connexion :
        - Acceptation de la socket.
        - Lecture du pseudo du client.
        - Ajout du client à la liste globale `clients`.
    - Pour un message :
        - Lecture du message reçu.
        - Si un client se déconnecte, suppression de ses informations et diffusion aux autres clients
        - Sinon appelle `recoitUnMessage(buffer)`

Points clés :

- `fd_set` permet de surveiller plusieurs descripteurs de fichiers simultanément.
- La gestion des erreurs (comme les déconnexions) est intégrée.

Initialisation de la connexion : int init_connection(void);

Description :

- Initialise et configure la socket du serveur pour accepter les connexions entrantes.
- Définit l'adresse et le port d'écoute.

Étapes :

1. Création de la socket avec `socket()`.
2. Association de l'adresse (bind) avec `bind()`.
3. Mise en mode écoute avec `listen()` pour attendre les connexions.

Gestion des erreurs:

- Le programme affiche un message d'erreur et termine si une étape échoue.

Gestion des clients : `remove_client` et `clear_clients`

1. Suppression d'un client : void remove_client(int to_remove, int *actual);
   Description **:**
   - Supprime un client spécifique du tableau `clients`.
   - Réorganise la liste pour combler l'espace libéré.
2. Fermeture de toutes les connexions client : void clear_clients(int actual);
   Description **:**
   - Ferme proprement toutes les sockets ouvertes pour les clients encore connectés.

Lecture et écriture des messages:

1. Lecture d'un message : int read_client(SOCKET sock, char *buffer);
   Description :
   - Reçoit les données d'un client sur une socket donnée.
   - Retourne la taille du message ou 0 en cas d'erreur ou de déconnexion.
2. Envoi d'un message : void write_client(int indiceSocket, const char *buffer);
   Description :
   - Envoie un message à un client donné, identifié par son indice dans le tableau `clients`.

Diffusion des messages : void send_message_to_all_clients(Client sender, int actual, const char *buffer, char from_server);

Description**:**

- Diffuse un message à tous les clients sauf à l'expéditeur.
- Ajoute le pseudo de l'expéditeur si le message ne provient pas du serveur.

Pour **jeuCoteServer.c:**

Variables globales

- `Partie* partie` : Une structure globale représentant l'état actuel de la partie, incluant :
  - Les joueurs connectés et leurs attributs.
  - L'état du paquet de cartes.
  - Les informations spécifiques à chaque tour (distribution, scores, etc.).

Détails des fonctionnalités:

Traitement des messages reçus: int recoitUnMessage(char* buffer);

Description

Cette fonction décode un message reçu et déclenche des actions en fonction du type de message (indiqué par bufffer[0). Les types incluent :

- 1 : Un nouveau joueur rejoint la partie.
- 2 : Annonce d'un joueur concernant ses plis prévus.
- 4 : Communication de la carte jouée par un joueur.

Comportement

- Le message est analysé, et une fonction correspondante est appelée pour traiter l'information.

Gestion des nouveaux joueurs : void recoitUnNouveauJoueur(char pseudoDuNouveauJoueur[TAILLE_MAX_PSEUDO]);

Description

- Initialise un nouveau joueur et l'ajoute à la partie si elle n'est pas encore lancée.
- Informe tous les clients de la liste actuelle des joueurs connectés.

Points clés

- Limite les joueurs à un maximum de trois.
- Lance automatiquement le tour de jeu une fois le troisième joueur ajouté via `lancerUnTourDeJeu`.

Gestion des annonces : void recoitAnnonce(int indiceJoueur, int nombreDePlis);

Description

- Gère les annonces des joueurs concernant leurs prédictions de plis.
- Assure la cohérence des annonces en contrôlant qu'elles respectent les règles du jeu.
- Passe le tour à l'annonceur suivant ou déclenche la phase de jeu si toutes les annonces sont terminées.

Gestion des tours de jeu : void lancerUnTourDeJeu();

Description

- Initialise un nouveau tour si la partie n'est pas encore en cours.
- Distribue les cartes, met à jour les scores et gère les transitions entre les phases du jeu.

Distribution des cartes : void distriberLesCartes();

Description

- Distribue les cartes du paquet initialisé à chaque joueur.
- S'assure que chaque joueur reçoit un nombre de cartes conforme à la phase de jeu actuelle.

Envoi des Cartes Jouées: void envoyerCartesJouees(int signal);

- Informe tous les joueurs des cartes jouées à chaque tour et du joueur gagnant.

Envoi des Scores: void envoyerLesScores();

- Transmet le tableau des scores aux joueurs après un tour complet.

Envoi des Annonces: void envoyerMessageTourDAnnonces(int dernierMessTourDAnnonces);

- Envoie les annonces des joueurs actuelles et désigne qui doit annoncer ensuite.

Nous allons ensuite détailler le fonctionnement du côté client, celui-ci se compose de deux programmes: client.c et jeuCoteClient.c.

Pour **client.c:**

Les fonctionnalités clés sont :

1. Connexion au Serveur :
    - Initialise une connexion réseau avec une adresse serveur donnée.
    - Établit une communication bidirectionnelle via une socket TCP.
2. Interaction Temps Réel :
    - Surveille les messages reçus du serveur.
    - Envoie les messages générés localement.
3. Gestion des Ressources :
    - Libération des ressources (affichage, socket) à la fermeture de l'application.

Description des fonctions:

Fonction principale : void app(const char *address, const char *name);

Description

- Établit une connexion au serveur via `init_connection`.
- Gère une boucle principale surveillant :
    1. Les messages entrants du serveur.
    2. Les messages générés localement.

Étapes clés

1. Établir la connexion :
    - Envoie un message d'identification (pseudo).
    - Initialise les structures d'affichage locales via `init_Affichage`.
2. Écoute active :
    - Surveille les événements réseau via `select`.
    - Reçoit des messages avec `read_server` et les traite via `recoitMessage`.
3. Émission conditionnelle :

    - Envoie les messages générés par le client avec `write_server` si le buffer n’est pas vide.

Gestion des ressources

- Termine proprement la connexion avec `end_connection`.
- Libère la mémoire liée aux structures locales avec `liberer_Affichage`.

Initialisation de la connexion : int init_connection(const char *address);

Description

- Crée une socket pour se connecter à une adresse serveur.
- Traduit l'adresse en format réseau grâce à `gethostbyname`.
- Établit une connexion avec le serveur via `connect`.

Gestion des erreurs

- Retourne un message d'erreur si la création de socket ou la connexion échoue.

Lecture et écriture des messages

1. Lecture : `int` read_server(SOCKET sock, char *buffer);
   Description :
    - Reçoit des données du serveur.
    - Ajoute un caractère nul (`\0`) à la fin du message reçu pour le rendre lisible.
2. Écriture : void write_server(SOCKET sock, const char *buffer);
   Description :
    - Envoie des données au serveur.
    - Assure l'intégrité des transmissions en gérant les erreurs.

Fin de connexion : void end_connection(int sock);

Description

- Ferme la socket réseau pour mettre fin à la connexion.
- Garantit la libération propre des ressources système.

Pour **jeuCoteClient.c:**

Les fonctionnalités clés sont:

1. Gestion des ressources d'affichage :
    - Initialisation et libération des structures nécessaires à la représentation graphique de la partie.
2. Interaction joueur :
    - Prise en charge des annonces et des choix de cartes par l'utilisateur.
3. Affichage dynamique :

    - Représentation en temps réel de l'état de la partie (scores, cartes jouées, annonces).

Description des fonctions:

Initialisation de l'affichage: Affichage* init_Affichage();

Description :

- Alloue de la mémoire pour la structure `Affichage`.
- Initialise les tableaux pour les scores, les annonces, les plis effectués, et les cartes en jeu à leurs valeurs par défaut.
- Retourne un pointeur sur la structure initialisée.

Libération de la mémoire pour l'affichage : void liberer_Affichage(Affichage* affichage);

Description :

- Libère la mémoire associée à la structure `Affichage`.

Gestion des annonces : int demanderAnnonce(int maxAnnonce, int totalAnnoncee, char *touteLesAnnonce);

Description :

- Permet au joueur de saisir son annonce en vérifiant qu'elle respecte les règles (valeur comprise entre 0 et un maximum autorisé).
- Si le joueur est le dernier à faire son annonce, la valeur totale des annonces est contrôlée pour éviter une annonce non réalisable.

Détection du dernier Joueur à parler : int estLeDernierAParler(char *touteLesAnnonce);

Description :

- Retourne 1 si le joueur courant est le dernier à faire une annonce, sinon `0`.

Choix de la carte à jouer : int choisirCarteAJouer(int cartes[]);

Description :

- Permet au joueur de choisir une carte de sa main en vérifiant qu'elle est valide (présente dans ses cartes disponibles).
- Gère le cas spécial de l'excuse (choix entre la meilleure ou la pire carte).

Extraction des pseudos : int extrairePseudos(char *message, char pseudos[3][TAILLE_MAX_PSEUDO], char *monPseudo);

Description :

- Analyse le message contenant les pseudos des joueurs et les stocke dans un tableau.
- Identifie l'indice du joueur local dans ce tableau.

Gestion des messages reçus : void recoitMessage(char *buffer, char *name, Affichage* affichage);

Description :

- Analyse le contenu d'un message et met à jour l'état de la structure `Affichage`.
- Réagit aux différents types de messages (annonces, cartes jouées, scores).

Détermination du gagnant d'un pli : int quiAGagnePli(char *buffer);

Description :

- Analyse un message pour déterminer quel joueur a remporté le pli courant.

Affichage des résultats : void afficherClassement(int score[3], char pseudos[3][TAILLE_MAX_PSEUDO]);

Description :

- Affiche le classement final des joueurs, trié par score.

Interface dynamique : void afficher(Affichage *affichage);

Description :

- Représente graphiquement l'état actuel de la partie.
- Affiche les scores, les annonces, les cartes jouées et les plis pour chaque joueur.
- Inclut la liste des cartes disponibles pour le joueur local.

UDP

Le jeu avec une connexion UDP fonctionnant avec la même logique nous présenterons uniquement les changements avec le protocole TCP. La différence principale se trouve dans le protocole en lui-même, en effet en UDP il n’y a pas de connexion avec le serveur à proprement parler. Ici les clients enregistre l’adresse du serveur est lui envoie directement les informations, et inversement. Ainsi le prototype de write_client devient:void write_client(SOCKET sock, SOCKADDR_IN *sin, const char *buffer). Ainsi cela oblige dans

toutes les fonctions qui envoie un message au client d'ajouter dans le prototype le tableau des clients avec leur adresse ainsi que la socket.

Déroulement d'une partie:

L'écran lors de la compilation

```
gcc -Wall -c -o client.o client.c
client.c: In function ‘app’:
client.c:39:32: warning: passing argument 2 of ‘recoitMessage’ discards ‘const’ qualifier from pointer target type [-Wdiscarded-qualifiers]
   39 |          recoitMessage(buffer, name , affichage);
      |                                ^~~~
In file included from client.c:6:
client.h:65:40: note: expected ‘char *’ but argument is of type ‘const char *’
   65 | void recoitMessage(char *buffer, char *name, Affichage *affichage);
      |                                  ~~~~~~^~~~
gcc -Wall -c -o jeuCoteClient.o jeuCoteClient.c
gcc -Wall -o client client.o jeuCoteClient.o
gcc -Wall -c -o jeuCoteServer.o jeuCoteServer.c
jeuCoteServer.c: In function ‘recoitUnMessage’:
jeuCoteServer.c:33:48: warning: operation on ‘i’ may be undefined [-Wsequence-point]
   33 |                         pseudoDuNouveauJoueur[i++] = buffer[i+1];
      |                                               ~^~
jeuCoteServer.c: In function ‘fusionnerDeuxStrings’:
jeuCoteServer.c:54:23: warning: operation on ‘i’ may be undefined [-Wsequence-point]
   54 |                 str1[i++] = str2[i - indicePremierCharEcrase];
      |                       ~^~
jeuCoteServer.c: In function ‘recoitUnNouveauJoueur’:
jeuCoteServer.c:67:106: warning: operation on ‘i’ may be undefined [-Wsequence-point]
   67 |                 while (pseudoDuNouveauJoueur[i] != '\0' && i < TAILLE_MAX_PSEUDO) nouveauJoueur->pseudo[i++] = pseudoDuNouveauJoueur[i
];
      |                                                                                                          ~^~
gcc -Wall -c -o server.o server.c
gcc -Wall -c -o cartes.o cartes.c
gcc -Wall -o server jeuCoteServer.o server.o cartes.o
```

Lancement du serveur

```
(base) lucien-coudert@la-machine:~/Documents/école/aveclescopains/ProjetInfo/testTCP$  ./server
serveur ouvert
```

Lorsque un joueur se connecte

```
                Partie de Tarot Africain
-------------------------------------------------------------------------------------------------------
            JOUEUR             |   SCORE   |   NOMBRE PLI ANNONCES   |   NOMBRE PLI FAIT   |   CARTE JOUEE
-------------------------------------------------------------------------------------------------------
JL                             |     0     |                         |          0          |
-------------------------------------------------------------------------------------------------------
                               |     0     |                         |          0          |
-------------------------------------------------------------------------------------------------------
                               |     0     |                         |          0          |
-------------------------------------------------------------------------------------------------------
```

Lorsque les trois joueurs sont connectés:

```
                Partie de Tarot Africain
-------------------------------------------------------------------------------------------------------
            JOUEUR             |   SCORE   |   NOMBRE PLI ANNONCES   |   NOMBRE PLI FAIT   |   CARTE JOUEE
-------------------------------------------------------------------------------------------------------
JL                             |     0     |                         |          0          |
-------------------------------------------------------------------------------------------------------
Dimitri                        |     0     |                         |          0          |
-------------------------------------------------------------------------------------------------------
Lucien                         |     0     |                         |          0          |
-------------------------------------------------------------------------------------------------------
 Cartes disponibles : 8     17     16     14     6
```

Pendant la partie:

```
                Partie de Tarot Africain
-------------------------------------------------------------------------------------------------------------
           JOUEUR            |   SCORE   |   NOMBRE PLI ANNONCES   |   NOMBRE PLI FAIT   |   CARTE JOUEE
-------------------------------------------------------------------------------------------------------------
JL                           |     0     |           2             |          0          |        21
-------------------------------------------------------------------------------------------------------------
Dimitri                      |     0     |           2             |          0          |       7
-------------------------------------------------------------------------------------------------------------
Lucien                       |     0     |           2             |          0          |
-------------------------------------------------------------------------------------------------------------
 Cartes disponibles : 12     19     7     20     9
```

Fin de la partie:

```
La partie est terminée !

Classement final :
1. Dimitri avec 4 points
2. Lucien avec 5 points
3. JL avec 8 points
```