Projet de Système et Réseaux

Jouer à 6 qui prend en ligne

Présenté par:
Rharmaoui Rafik et **Kaidi Ahmed El-Aziz**

Licence 3 Informatique
Année universitaire: 2023/2024

1 Introduction

Le projet que nous présentons vise à concrétiser les concepts et mécanismes enseignés dans le cadre du cours de Systèmes Réseaux. Notre objectif est de développer une version informatique du jeu de cartes "6 qui prend", permettant aux utilisateurs de s'engager dans des parties en ligne via un terminal, en interagissant avec d'autres joueurs réels ou des adversaires automatisés.

"6 qui prend" est un jeu de type "défausse" créé par Wolfgang Kramer en 1994. Dans ce jeu stratégique, les joueurs doivent placer leurs cartes dans différentes rangées sans jamais dépasser la sixième carte. Les choix de cartes étant secrets et simultanés, une dose de calcul stratégique est nécessaire pour éviter d'accumuler un nombre excessif de "têtes de bœuf". Le principe du jeu réside dans la quête du joueur ayant le moins de "têtes de bœuf" à la fin de chaque partie.

Chaque manche commence avec quatre cartes tirées au hasard, formant ainsi quatre rangées. Chaque joueur reçoit ensuite dix cartes, et à chaque tour, ils choisissent secrètement une carte parmi celles-ci pour la placer dans l'une des rangées existantes. Lorsqu'une rangée atteint cinq cartes, le joueur qui pose la sixième carte prend la rangée, plaçant la sienne en première position. Si un joueur pose une carte plus petite que celles déjà présentes dans une rangée, il ramasse la rangée de son choix, généralement celle contenant le moins de "têtes de bœuf".

À la fin de chaque manche, les joueurs additionnent les "têtes de bœuf" accumulées. Le joueur atteignant 66 "têtes de bœuf" (ou un autre seuil défini) perd la partie.

Les fonctionnalités du système que nous développerons englobent quatre types de processus : le Gestionnaire du jeu, responsable de l'accueil des joueurs, de la distribution des cartes, de la gestion des rangées, et du suivi des scores ; le Joueur Humain, permettant aux utilisateurs de jouer en affichant le début et la fin des parties, et en interagissant avec le gestionnaire pour jouer leurs cartes ; le Joueur Robot, participant au jeu de manière automatisée en cherchant à minimiser les "têtes de bœuf" ; enfin, la Génération de stats, un processus en shell/awk interagissant avec le gestionnaire pour générer des statistiques, enregistrer les résultats dans des fichiers de log, et produire un document PDF illustrant le déroulement des parties.

Ce rapport détaillera la conception, l'implémentation et les résultats obtenus dans le développement de ce jeu de cartes en ligne, mettant en lumière l'application des concepts fondamentaux des Systèmes et Réseaux dans un contexte ludique et interactif.

2 Gestionnaire

La classe Gestionnaire constitue le pivot central du jeu 6 qui prend, orchestrant avec précision l'ensemble des mécanismes nécessaires à son bon déroulement. Responsable de l'initialisation du paquet de cartes, de la distribution aux joueurs, et du positionnement stratégique sur la table, elle incarne le cœur du jeu. Au-delà de ses fonctions logistiques, elle prend en charge la coordination des tours de jeu, gère les scores des joueurs en fonction des cartes jouées, et détermine le dénouement de la partie. Son architecture intègre la gestion des threads pour permettre des interactions simultanées avec plusieurs joueurs, tandis que ses méthodes de sérialisation et de désérialisation facilitent la communication entre le serveur et les clients. Grâce à cette classe, le jeu de cartes s'anime dans une structure robuste, guidée par les règles définies et prête à offrir une expérience interactive et immersive aux participants.

2.1 Structures

2.1.1 Carte

- **number**: Un entier représentant la valeur numérique associée à une carte.
- **tetesDeBoeuf**: Un entier indiquant le nombre de "têtes de bœuf" associé à une carte.
- **name**: Un tableau de caractères permettant de stocker le nom du joueur qui a joué cette carte.

2.1.2 Joueur

- **cartes**: Un tableau de structures de type Carte, représentant les cartes en possession du joueur.
- **points**: Un entier indiquant le nombre de points accumulés par le joueur pendant la partie.

- **socket**: Un entier représentant le socket associé au joueur pour la communication dans un contexte réseau.
- **name**: Un tableau de caractères contenant le nom du joueur.
- **carteJouet**: Une structure de type Carte représentant la carte actuellement jouée par le joueur.
- **nbCartesRestantes**: Un entier représentant le nombre de cartes restantes dans la main du joueur.
- **indice**: Un entier servant potentiellement à identifier le joueur.
- **gagné**: Un indicateur binaire ou un entier déterminant si le joueur a remporté la partie.

2.1.3 Paquet

- **NbCartes**: Un entier indiquant le nombre total de cartes dans le paquet.
- **cartes**: Un tableau de structures de type Carte représentant les cartes présentes dans le paquet.

2.1.4 Table

- **ranges**: Un tableau dynamique de pointeurs vers des structures de type Carte, représentant les différentes rangées de cartes sur la table.
- **nombresCartesRange**: Un tableau d'entiers représentant le nombre de cartes dans chaque rangée.
- **tetesDeBoeuf**: Un tableau d'entiers représentant le nombre de "têtes de bœuf" associées à chaque rangée.
- **manche**: Un entier indiquant le numéro de la manche en cours.
- **tour**: Un entier indiquant le numéro du tour actuel.

2.2 Fonctions

2.2.1 afficheCarte

Cette fonction permet d'afficher les informations d'une carte de manière colorée en fonction du nombre de têtes de bœuf qu'elle contient.

2.2.2 affichePaquet

Cette fonction permet d'afficher l'ensemble des cartes d'un joueur en utilisant la fonction afficheCarte pour chaque carte. Cela offre une manière de visualiser le contenu du paquet d'un joueur.

2.2.3 initialiserPaquetJoueur

Cette fonction initialise le paquet d'un joueur en lui attribuant 10 cartes retirées du paquet de jeu à l'aide de la fonction retirCarte.

2.2.4 retireCarte

La fonction retirCarte est conçue pour retirer la carte du dessus d'un paquet. Elle prend en paramètre un pointeur vers une structure de paquet (Paquet) et retourne une structure de carte (Carte). La fonction vérifie d'abord si le paquet contient au moins une carte en examinant le champ NbCartes. Si c'est le cas, elle extrait la carte du dessus du paquet, ajuste le nombre de cartes dans le paquet, et renvoie cette carte. Si le paquet est vide, la fonction crée une carte factice avec number et tetesDeBoeuf égaux à 0, symbolisant ainsi une carte vide, et la renvoie. En résumé, la fonction assure la manipulation sécurisée des paquets de cartes, évitant les erreurs liées à un paquet vide et fournissant une carte vide comme indicateur lorsque le paquet ne contient aucune carte.

2.2.5 placerDansRangee

La fonction placerDansRangee prend en paramètres un pointeur vers une structure de tableau (Table), un pointeur vers une carte (Carte), un tableau d'entiers représentant les sockets des joueurs (socketsJoueurs), et un pointeur vers la structure d'un joueur (Joueur). La fonction a pour but de placer une carte dans l'une des quatre rangées du tableau en respectant certaines conditions.

Elle commence par initialiser une variable differenceMinimale à une valeur maximale possible et rangeeCible à -1. Ensuite, elle parcourt les quatre rangées du tableau pour déterminer dans laquelle la carte peut être placée. La carte ne peut être placée que si son numéro est supérieur ou égal au numéro de la première carte de la rangée, et inférieur à tous les autres numéros de cartes de la même rangée.

Si la carte peut être placée dans une rangée, la fonction calcule la différence entre le numéro de la carte et le numéro de la dernière carte de la rangée. Si cette différence est plus petite que la differenceMinimale actuelle, la differenceMinimale est mise à jour et la rangeeCible est définie sur l'index de la rangée.

Enfin, la fonction vérifie si la carte peut être ajoutée à la rangée cible sans dépasser la limite de cinq cartes par rangée. Si c'est le cas, la carte est ajoutée à la rangée, le nombre de cartes dans la rangée est mis à jour, et les têtes de bœuf de la rangée sont ajustées en conséquence. Si la rangée est pleine, les points du joueur sont mis à jour en fonction des têtes de bœuf de chaque carte de la rangée, puis la rangée est vidée pour accueillir la nouvelle carte.

2.2.6 initialiserTable

La fonction initialiserTable prend en paramètre un pointeur vers une structure de paquet (Paquet *paquet) et retourne un pointeur vers une structure de table (Table *). La fonction a pour objectif d'initialiser une table de jeu en allouant la mémoire nécessaire pour les rangées, les nombres de cartes dans chaque rangée, et les têtes de bœuf de chaque rangée.

La fonction commence par allouer la mémoire pour la structure de la table (Table) et ses composants tels que les rangées, les nombres de cartes dans chaque rangée et les têtes de bœuf. Ensuite, elle utilise une boucle pour initialiser chaque rangée avec une carte retirée du paquet à l'aide de la fonction retirCarte. Le nom de la carte est également défini comme "Gestionnaire", et les nombres de cartes et les têtes de bœuf de chaque rangée sont mis à jour en conséquence.

Enfin, la fonction initialise les variables manche et tour de la table à 1, puis renvoie un pointeur vers la table nouvellement créée.

2.2.7 serializeJoueur

La fonction serializeJoueur prend en paramètre un pointeur vers une structure Joueur, un tableau de caractères buffer destiné à contenir la sérialisation, et la taille bufferSize de ce tableau. La fonction vérifie d'abord que les paramètres sont valides, puis elle copie les membres de la structure Joueur (à l'exception du tableau de cartes) dans le tableau buffer de manière séquentielle. Chaque membre est copié à l'aide de la fonction memcpy, et l'offset est mis à jour en conséquence. La sérialisation inclut les entiers points, socket, indice, gagné, nbCartesRestantes, le tableau de caractères name de taille 100, ainsi que la structure Carte appelée carteJouet. Enfin, les cartes individuelles du joueur sont ajoutées au tableau buffer une par une à la fin de la sérialisation. La fonction s'assure que la taille totale de la sérialisation n'excède pas bufferSize.

2.2.8 deserializeJoueur

La fonction deserializeJoueur prend en paramètre un tableau de caractères buffer contenant une sérialisation d'une structure Joueur, la taille bufferSize de ce tableau, et un pointeur vers une structure Joueur destinée à être remplie par la désérialisation. La fonction vérifie d'abord que les paramètres sont valides, puis elle copie séquentiellement les données du tableau buffer dans les membres correspondants de la structure Joueur. Chaque copie est réalisée à l'aide de la fonction memcpy, en mettant à jour l'offset en conséquence. La désérialisation inclut les entiers points, socket, indice, gagné, nbCartesRestantes, le tableau de caractères name de taille 100, ainsi que la structure Carte appelée carteJouet. Les cartes individuelles du joueur sont également extraites du tableau buffer une par une à la fin de la désérialisation. La fonction s'assure que la taille totale de la désérialisation n'excède pas bufferSize.

2.2.9 initialiserPaquet

La fonction initialiserPaquet crée dynamiquement une instance de la structure Paquet et initialise les cartes du paquet. Chaque carte a un numéro unique allant de 1 à 104, et le nombre de têtes de bœuf est déterminé en fonction de diverses conditions sur le numéro de la carte. Par exemple, la carte numéro 55 a 7 têtes de bœuf, les cartes dont le numéro se termine par 0 ont 3 têtes de bœuf, celles dont le numéro est un multiple de 11 ont 5 têtes de bœuf, etc.

Ensuite, la fonction effectue un mélange aléatoire des cartes en utilisant l'algorithme de Fisher-Yates. Cela garantit que les cartes sont mélangées de manière aléatoire dans le paquet. En fin de compte, la fonction renvoie un pointeur vers le paquet nouvellement créé

2.2.10 afficheTable

La fonction afficheTable prend un pointeur vers une instance de la structure Table en paramètre et affiche les détails de la table, y compris le numéro de la manche, le numéro du tour, le contenu de chaque rangée (y compris le nombre de têtes de bœuf dans chaque rangée) et les cartes présentes dans chaque rangée.

La fonction utilise des codes de couleur ANSI (notamment 033[1;31m et 033[0m) pour afficher le texte en rouge, ce qui peut rendre la sortie plus lisible ou esthétique sur des terminaux qui prennent en charge ces codes. La présentation est organisée de manière claire, avec des informations distinctes pour chaque rangée de la table.

2.2.11 trouverRangeeAvecMoinsTetesDeBoeuf

La fonction trouverRangeeAvecMoinsTetesDeBoeuf prend un pointeur vers une instance de la structure Table en paramètre et recherche la rangée qui a le moins de têtes de bœuf. Elle parcourt chaque rangée de la table, compare le nombre de têtes de bœuf de chaque rangée avec la valeur minimale trouvée jusqu'à présent, et met à jour l'indice de la rangée avec le moins de têtes de bœuf.

La fonction retourne l'indice de la rangée qui a le moins de têtes de bœuf. Si toutes les rangées ont le même nombre de têtes de bœuf, la fonction retournera l'indice de la première rangée rencontrée avec ce minimum. Si la table est vide, la fonction retournera -1 pour indiquer qu'aucune rangée n'a été trouvée.

2.2.12 testerCarteEtViderRangee

La fonction testerCarteEtViderRangee prend en paramètres un pointeur vers une instance de la structure Table (tableau), un pointeur vers une instance de la structure Carte (carte), un tableau d'entiers (socketsJoueurs), et un pointeur vers une instance de la structure Joueur (joueur). Cette fonction teste la carte par rapport à chaque rangée de la table et décide si elle doit être placée dans une rangée existante ou si elle doit vider une rangée pour minimiser l'impact sur le total des têtes de bœuf.

La fonction commence par initialiser rangeeAVider à une valeur qui n'est pas une rangée valide (7 dans cet exemple). Ensuite, elle parcourt chaque rangée de la table et ajuste rangeeAVider si la carte peut être placée dans une rangée existante. Si rangeeAVider reste à la valeur initiale (7), cela signifie que la carte ne peut pas être placée dans une rangée existante, et la fonction utilise la fonction trouverRangeeAvecMoinsTetesDeBoeuf pour déterminer dans quelle rangée vider une carte.

Enfin, la fonction appelle soit la fonction placerDansRangee pour placer la carte dans une rangée existante, soit elle vide la rangée avec le moins de têtes de bœuf et y place la nouvelle carte. Cette approche permet de minimiser les points attribués au joueur tout en gérant les différentes situations de jeu.

2.2.13 serializeTable

La fonction serializeTable est destinée à convertir les données d'une instance de la structure Table en une représentation sérialisée pour la transmission ou le stockage. Elle prend en paramètres un pointeur vers la table à sérialiser (table), un tableau de caractères destiné à contenir la sérialisation (buffer), et la taille de ce tableau (bufferSize). La fonction procède à la sérialisation des rangées de la table en appelant une fonction serializeCard pour chaque rangée, puis elle sérialise les nombres de cartes dans chaque rangée (nombresCartesRange) ainsi que les têtes de bœuf (tetesDeBoeuf) à l'aide d'une fonction

serializeIntPtr. Enfin, les entiers manche et tour de la table sont copiés directement dans le tableau de caractères. L'utilisation de pointeurs dans la manipulation des membres de la structure permet de compacter les données sérialisées de manière efficace. La fonction inclut également un débogage qui affiche le nombre de cartes dans chaque rangée côté serveur.

2.2.14 deserializeTable

La fonction deserializeTable est conçue pour récupérer les données sérialisées d'une table à partir d'un tableau de caractères (buffer). Les paramètres comprennent également la taille de ce tableau (bufferSize) et un pointeur vers la structure Table (table) qui va être remplie avec les données désérialisées.

La fonction commence par désérialiser les rangées de la table en appelant une fonction deserializeCard pour chaque rangée. Ensuite, elle désérialise les nombres de cartes dans chaque rangée (nombresCartesRange) et les têtes de bœuf (tetesDeBoeuf) en utilisant une fonction deserializeIntPtr. La fonction copie ensuite directement dans la structure les entiers manche et tour à partir du tableau de caractères. Un débogage est inclus pour afficher le nombre de cartes dans chaque rangée côté joueur.

2.2.15 sendTable

La fonction sendTable est destinée à envoyer les données d'une structure Table à un client via une socket. Elle prend en paramètre le descripteur de fichier (clientSocket) représentant la socket du client et un pointeur vers la structure Table (table) dont les données vont être envoyées.

La fonction commence par allouer dynamiquement de la mémoire pour un tableau de caractères (data) de taille calculée en fonction de la structure de la table. Ensuite, elle copie séquentiellement les données de la table dans ce tableau en utilisant memcpy. Les données comprennent les nombres de cartes dans chaque rangée (nombresCartesRange), les têtes de bœuf dans chaque rangée (tetesDeBoeuf), les cartes individuelles dans chaque rangée (ranges), ainsi que les entiers manche et tour.

Finalement, la fonction utilise la fonction send pour envoyer le tableau de caractères sur la socket du client, puis libère la mémoire allouée dynamiquement.

2.2.16 initialiserJoueurs

La fonction initialiserJoueurs a pour objectif d'initialiser un tableau de pointeurs vers des structures Joueur. Elle prend en paramètre un tableau de pointeurs joueurs[] qui pointe vers des structures Joueur et initialise chaque joueur individuellement.

La boucle externe parcourt chaque joueur dans le tableau joueurs[] en utilisant l'indice i. À l'intérieur de cette boucle, la fonction alloue dynamiquement de la mémoire pour chaque joueur à l'aide de malloc(sizeof(Joueur)). Ensuite, elle initialise les différents membres de la structure Joueur. Chaque joueur reçoit un identifiant (indice), initialisé à la valeur de l'indice de la boucle externe. Le champ gagné est initialisé à zéro, et le nombre de cartes restantes (nbCartesRestantes) est fixé à 10.

La boucle interne parcourt chaque carte dans le tableau cartes[] du joueur et attribue à chaque carte une valeur obtenue en appelant la fonction retirCarte(paquetJeu).

En résumé, cette fonction crée et initialise un tableau de joueurs, en veillant à ce que chaque joueur ait un identifiant unique, un nombre initial de cartes, et que chaque carte soit retirée d'un paquet de jeu.

2.2.17 compareJoueurs

La fonction compareJoueurs est une fonction de comparaison destinée à être utilisée avec la fonction de tri qsort de la bibliothèque standard en langage C. Cette fonction prend deux pointeurs vers des structures Joueur (a et b) et renvoie un entier représentant le résultat de la comparaison entre ces deux joueurs.

La fonction commence par convertir les pointeurs a et b en pointeurs de pointeurs vers Joueur, ce qui est nécessaire pour utiliser qsort avec un tableau de pointeurs. Ensuite, elle effectue la comparaison en se basant sur le numéro de la carte jouet (carteJouet.number). La différence entre les numéros des cartes jouets de deux joueurs est renvoyée comme résultat de la comparaison.

Si la valeur renvoyée est négative, cela signifie que le joueur A doit être placé avant le joueur B lors du tri. Si la valeur est positive, cela indique que le joueur B doit être placé avant le joueur A. Si la valeur est nulle, les joueurs sont considérés comme équivalents pour le tri.

En résumé, cette fonction de comparaison est utilisée pour trier un tableau de pointeurs vers des structures Joueur en fonction des numéros de leurs cartes jouets.

2.2.18 trouverScoreMin

La fonction trouverScoreMin prend un tableau de pointeurs vers des structures Joueur (joueurs[]) en paramètre et retourne le score minimum parmi les joueurs présents dans le tableau. La fonction initialise la variable minScore avec le score du premier joueur (joueurs[0]-¿points) et ensuite, elle parcourt le reste du tableau pour comparer les scores des autres joueurs.

À chaque itération, la fonction vérifie si le score du joueur actuel (joueurs[i]-¿points) est inférieur à la valeur actuelle de minScore. Si c'est le cas, elle met à jour la valeur de minScore avec le score du joueur actuel.

À la fin de la boucle, la fonction renvoie la valeur de minScore, représentant ainsi le score minimum parmi tous les joueurs du tableau.

2.2.19 trouverScoreMax

La fonction trouverScoreMax prend un tableau de pointeurs vers des structures Joueur (joueurs[]) en paramètre et retourne le score maximum parmi les joueurs présents dans le tableau. La fonction initialise la variable maxScore avec le score du premier joueur (joueurs[0]-¿points) et ensuite, elle parcourt le reste du tableau pour comparer les scores des autres joueurs.

À chaque itération, la fonction vérifie si le score du joueur actuel (joueurs[i]-¿points) est supérieur à la valeur actuelle de maxScore. Si c'est le cas, elle met à jour la valeur de maxScore avec le score du joueur actuel.

À la fin de la boucle, la fonction renvoie la valeur de maxScore, représentant ainsi le score maximum parmi tous les joueurs du tableau.

2.2.20 scoreJoueurs

La fonction scoresJoueurs prend un tableau de pointeurs vers des structures Joueur (joueurs[]) en paramètre et vérifie si l'un des joueurs a atteint ou dépassé un certain score (représenté par la constante SCORE). Si tel est le cas, la fonction appelle les fonctions trouverScoreMax et trouverScoreMin pour déterminer le score maximum et minimum parmi tous les joueurs.

Ensuite, la fonction renvoie 1 pour indiquer que le score a été atteint ou dépassé par au moins un joueur, et elle met à jour les variables globales max et min avec les valeurs retournées par les fonctions trouverScoreMax et trouverScoreMin.

Si aucun joueur n'a atteint ou dépassé le score spécifié, la fonction renvoie 0.

2.2.21 ecrire

La fonction ecrire a pour objectif d'exécuter un script shell nommé ecrire.sh en utilisant la fonction system de C. Cette fonction génère une commande en formatant une chaîne de caractères appelée command. La commande initiale inclut le numéro de la manche provenant de la structure tableJeu-¿manche. Ensuite, la fonction parcourt un tableau de joueurs (joueurs) et ajoute à la commande leur nom et leur score à l'aide de la fonction snprintf. La commande finale est exécutée à l'aide de la fonction system(command).

2.2.22 main

Le programme principal (main) met en œuvre la logique du jeu pour le nombre spécifié de joueurs. Il commence par initialiser le paquet de cartes, la table de jeu, et les joueurs. Ensuite, il crée un socket serveur, accepte les connexions des joueurs, et lance la partie.

Le jeu se déroule en boucle tant que la condition de fin de partie n'est pas remplie. À chaque tour, les informations sur les joueurs sont sérialisées et envoyées aux joueurs, qui répondent avec leurs cartes jouées. La table de jeu est mise à jour en fonction des actions des joueurs, et le score est affiché à la fin de chaque tour.

Lorsqu'une manche est terminée, le programme vérifie si la partie est terminée en fonction des scores des joueurs. Si c'est le cas, il affiche le résultat final et termine la partie. Sinon, il réinitialise le paquet et la table pour la prochaine manche.

Il est à noter que le programme utilise des sockets pour la communication entre le serveur et les joueurs, et des fonctions de sérialisation/désérialisation pour échanger les données. De plus, le programme utilise des scripts shell pour écrire les résultats dans un fichier externe.

3 Joueur

3.1 Fonctions

3.1.1 supprimerCarte

La fonction supprimerCarte prend en paramètres la position pos d'une carte dans le tableau de cartes d'un joueur de type Joueur. Elle est utilisée pour retirer une carte du paquet du joueur à la position spécifiée. La fonction effectue cette suppression en décalant toutes les cartes situées à une position supérieure d'un indice vers le bas, puis diminue le nombre total de cartes nbCartesRestantes du joueur de 1. Cette opération a pour effet de "supprimer" la carte du paquet du joueur, ajustant ainsi la taille effective de son paquet.

3.1.2 carteValide

La fonction carteValide prend en paramètres un entier nombre représentant le numéro d'une carte et un joueur de type Joueur. Elle vérifie si une carte avec le numéro spécifié est présente dans le paquet du joueur. La fonction parcourt le tableau de cartes du joueur (cartes) en comparant le numéro de chaque carte avec la valeur nombre. Si elle trouve une correspondance, la fonction renvoie 1, indiquant que la carte est valide. Dans le cas contraire, elle retourne 0, signalant que la carte n'est pas présente dans le paquet du joueur. Cette fonction est utilisée pour valider le choix d'une carte par le joueur avant de la jouer.

3.1.3 receiveTable

La fonction receiveTable est responsable de la réception des données de la table depuis le serveur. Elle prend en paramètres le socket du serveur (serverSocket) et un pointeur vers une structure Table (table). La taille des données attendues est calculée en fonction de la taille des membres de la structure Table.

Ensuite, un tampon (data) est alloué pour stocker les données reçues depuis le serveur. La fonction recv est utilisée pour recevoir ces données à partir du socket du serveur.

L'offset est initialisé à zéro, et la fonction copie ensuite les tableaux d'entiers, les données de structure Carte pour chaque rangée, ainsi que les nouveaux membres (manche et tour) dans la structure Table.

Finalement, la mémoire allouée pour le tampon data est libérée avec la fonction free.

3.1.4 rechercherCarte

La fonction rechercherCarte prend en paramètres un entier nombre représentant le numéro d'une carte et un joueur de type Joueur. Elle recherche la carte avec le numéro spécifié dans le paquet du joueur. Si elle trouve une correspondance, la fonction renvoie la carte trouvée, puis elle supprime cette carte du paquet du joueur en appelant la fonction supprimerCarte. Enfin, elle retourne la carte trouvée. Si aucune carte correspondante n'est trouvée, la fonction renvoie une carte vide.

3.1.5 main

Lors de son exécution, le joueur est invité à fournir l'adresse IP du serveur ainsi que son nom. Ensuite, le programme crée un socket client et tente de se connecter au serveur. Une fois connecté, le nom du joueur est envoyé au serveur, et les données de la table de jeu sont récupérées à l'aide de la fonction receiveTable, puis affichées avec afficheTable. Ensuite, une boucle principale du jeu démarre, où le joueur effectue des actions à chaque tour. Les informations du joueur, telles que son paquet de cartes

et son score, sont reçues depuis le serveur et affichées à chaque itération. Le joueur choisit une carte, cette information est envoyée au serveur, et la table est mise à jour. Le processus continue jusqu'à ce que la partie se termine, et le résultat, indiquant si le joueur a gagné ou perdu, est affiché à la fin. Enfin, la connexion au serveur est fermée avec la fonction close.

4 joueurRobot

4.1 fonctions

4.1.1 verifier

La méthode verifier a pour objectif de déterminer la meilleure rangée où le robot peut placer une carte dans le jeu. Initialement, elle initialise une variable de différence minimale à une valeur maximale et l'indice de la rangée cible à -1. En parcourant les quatre rangées du tableau de jeu, elle identifie la rangée vide comme rangée cible si elle existe. Ensuite, elle évalue la possibilité de placer la carte dans chaque rangée, en vérifiant si la carte est supérieure à la plus petite carte déjà présente. Si la carte peut être placée, la méthode met à jour la rangée cible si la différence entre la valeur de la carte et la plus petite carte dans la rangée est plus petite que la différence minimale actuelle. Finalement, elle retourne l'indice de la rangée cible, indiquant où le robot devrait jouer la carte.

4.1.2 placerCarteRobot

Quant à la méthode placerCarteRobot, elle parcourt les cartes du joueur et utilise la méthode verifier pour déterminer la meilleure rangée cible pour chaque carte. Si une rangée cible est trouvée et qu'il y a de l'espace dans cette rangée, la carte est retirée du paquet du joueur et retournée. Si la première boucle ne réussit pas, une deuxième tentative est effectuée avec des conditions moins strictes, permettant même de jouer dans une rangée pleine. Si cette tentative échoue également, la méthode choisit finalement la première carte du joueur. En résumé, la méthode cherche à jouer de manière stratégique en choisissant la meilleure rangée cible, mais elle s'adapte à des conditions moins idéales si nécessaire.

5 Génération de stats

5.1 ecrire

Le script Bash ecrire permet de générer un fichier texte contenant les résultats d'une manche de jeu. Il prend en compte plusieurs arguments, notamment le numéro de la manche, les noms des joueurs et le nombre de points de chaque joueur. Le fichier texte résultant, appelé resultats.txt, est créé ou mis à jour pour inclure les détails de la manche en cours. Il utilise également un fichier temporaire, score.txt, pour stocker temporairement les informations avant de les ajouter au fichier principal.

Le script démarre en vérifiant la présence d'un nombre suffisant d'arguments, puis définit plusieurs variables pour les noms de fichiers. Il extrait ensuite le numéro de la manche et ajoute un en-tête dans le fichier texte pour indiquer la manche en cours. En parcourant les arguments restants, il écrit les noms des joueurs et leurs scores associés dans le fichier texte principal.

Une particularité du script est son appel à un script externe, toPdf.sh, suggérant une fonctionnalité de conversion du fichier texte en un format PDF.

En résumé, ce script joue un rôle dans le processus global de gestion des résultats de jeu, avec un accent sur la création d'un fichier texte structuré et la possibilité de le convertir en un format PDF, offrant ainsi une manière organisée de documenter les scores de chaque manche.

5.2 toPdf

Le script Bash toPdf a pour objectif de convertir le fichier resultats.txt en un fichier PDF à l'aide de l'outil pdflatex.

Le script commence par vérifier le nombre d'arguments passés lors de son exécution. Si le nombre d'arguments n'est pas égal à 1, le script affiche un message d'utilisation indiquant le format correct et se termine avec le code de sortie 1.

Ensuite, le nom du fichier texte en entrée est assigné à la variable input_file, avec une vérification subséquente de l'existence du fichier. Si le fichier texte spécifié n'existe pas, le script affiche un message d'erreur et se termine également avec le code de sortie 1.

Le nom du fichier PDF de sortie est dérivé du nom du fichier texte en supprimant l'extension ".txt".

Cela est accompli en utilisant la construction `${input_file%.txt}`, qui renvoie le nom du fichier sans l'extension.

La conversion réelle du fichier texte en fichier PDF est réalisée en utilisant la commande pdflatex.

Le contenu LaTeX est généré dynamiquement en utilisant une syntaxe ¡¡EOF, ce qui permet d'inclure le contenu du fichier texte dans le document LaTeX. Les messages de sortie de pdflatex sont redirigés vers /dev/null pour éviter de polluer la sortie standard.

Enfin, les fichiers temporaires générés par pdflatex (les fichiers .aux et .log) sont supprimés pour maintenir la propreté du répertoire.

En résumé, ce script simplifie le processus de conversion d'un fichier texte en un fichier PDF en utilisant pdflatex, un outil couramment utilisé pour la création de documents au format PDF à partir de documents LaTeX.

6 Changements

Les changements apportés à la dernière version présentée en démonstration sont les suivants :

- La possibilité de définir le nombre de joueurs directement depuis le terminal.
- La possibilité de définir le score directement depuis le terminal.
- Génération d'un fichier PDF contenant les statistiques.
- L'ajout du fichier Makefile.

7 Conclusion

En conclusion, le projet de jeu 6 qui prend en langage C présente une structure bien organisée et des fonctionnalités de base solides. L'implémentation de la communication via des sockets offre une base robuste pour la connectivité entre le serveur et les clients, permettant une expérience de jeu interactive. Les algorithmes utilisés, tels que la logique de jeu du joueur robot, la vérification de la validité des cartes, et le tri des paquets, démontrent une approche réfléchie dans la conception du jeu.

L'utilisation de fonctions modulaires contribue à la lisibilité et à la maintenabilité du code.

L'interface utilisateur, bien que principalement en ligne de commande, fournit des informations nécessaires aux joueurs et affiche l'état actuel du jeu de manière claire.