Année 2022 - 2023

COMPTE RENDU – SYSTEMES ET RESEAUX

6 qui prend

BARBIER Tom – PERE Brandon

Table des matières

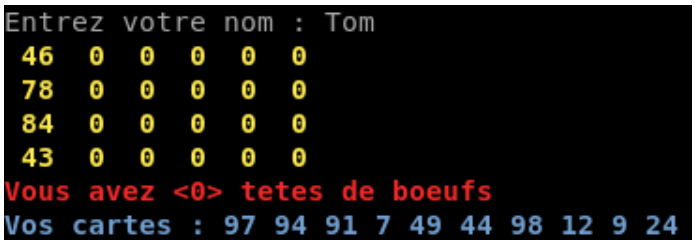
JEU

Pour lancer une partie il suffit de lancer le serveur en indiquant le nombre de joueurs et de bots qui y participent puis de lancer les clients en indiquant le nom du joueur. Ceux-ci vont ensuite se connecter au serveur et une partie débutera.

Une partie est composé de plusieurs rounds jusqu'à un total de 10 rounds dont chacun est composé de 10 manches.

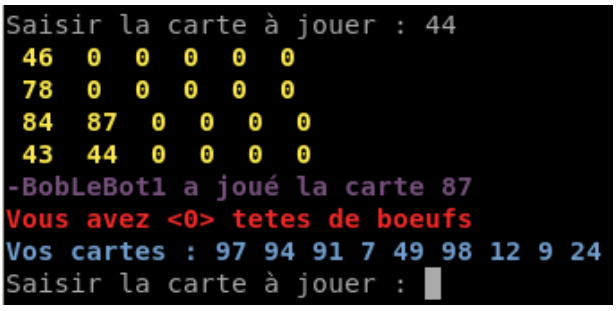
Boucle Jeu :

(Initialisation) Le serveur distribue les cartes restantes aux différents participants et il pose la première carte des quatre piles centrales à construire. Les joueurs ont connaissances des piles centrales, de leurs cartes, de leurs têtes de bœuf ainsi que les cartes jouées par les participants.



Boucle Round :

Chaque joueur va devoir entrée une des cartes qu'il possède. Le serveur va attendre que tous les joueurs aient joués pour actualiser les piles et les nombres de têtes pour chaque joueur. Le serveur va aussi réafficher les informations propres aux joueurs ainsi que les cartes ayant été jouées.

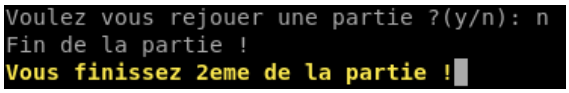


Fin boucle Round :

Lorsque les joueurs n'ont plus de cartes à jouer la manche est donc terminée et on demande à chacun s'ils veulent ou non continuer la partie.

Fin boucle Jeu :

Si le joueur décide de ne pas continuer la partie ou si la manche 10 est terminée alors c'est la fin du jeu et chaque joueur voit sa position dans le classement.



A la fin d'une partie un fichier .pdf nommé « History.pdf » est créé. Ce fichier retrace l'évolution de la partie soit toutes les actions réalisées par les joueurs ainsi que les évolutions des piles. Le fichier se termine par le résultat des joueurs c'est-à-dire un classement avec à coté de chaque nom de joueur le nombre de tête correspondant.

9 35 0 0 0 0
4 12 20 24 0 0
3 19 0 0 0 0
43 44 49 91 0 0
-Tom a joue la carte 91
-BobLeBot1 a joue la carte 19
---
9 35 0 0 0 0
4 12 20 24 27 0
3 19 0 0 0 0
43 44 49 91 94 0
-Tom a joue la carte 94
-BobLeBot1 a joue la carte 27
---
9 35 59 0 0 0
4 12 20 24 27 0
3 19 0 0 0 0
97 0 0 0 0 0
-Tom a joue la carte 97
-BobLeBot1 a joue la carte 59

Resultats :
2. Tom | 14
1. BobLeBot1 | 3

EXPLICATIONS TECHNIQUES

1. LE LANGAGE : C

Comme prévu dans le pré-rapport, nous avons utilisé le langage C pour réaliser l'entièreté du projet. Nous pensions que des scripts shell auraient pu nous simplifier certaines tâches, mais cela n'a pas été le cas. En effet, le langage C dispose de très nombreuses bibliothèques très utiles que nous avons choisi d'utiliser.

2. L'ARCHITECTURE SERVEUR-CLIENT

Afin de répondre à l'amélioration du jeu multijoueurs en réseau nous avons établi une architecture serveur-client. Pour cela, nous avons utilisé les bibliothèques<sys/socket.h> et <arpa/inet.h> afin de faire des connexions TCP-IP via socket. Le protocole TCP est un protocole dit connecté. Il contrôle si le paquet est arrivé à destination si ce n'est pas le cas il le renvoie.

Le client :

Le client est un programme classique de connexion via socket, il a besoin de l'adresse ip du serveur et du port pour se connecter à celui-ci. Lors de la réalisation de ce projet, nous avons effectué l'ensemble des tests en local, de ce fait l'ip du serveur (localhost) et le port sont inclus dans le code et non-modifiable. Mais si nous voulions améliorer cela, il suffirait de permettre à l'utilisateur lors du lancement de l'application client de pouvoir saisir l'adresse ip du serveur et le port.
Le comportement du processus client est très simple, il reçoit les messages envoyés par le serveur et les affichent au client. Il permet également, quand le serveur le demande, une entrée utilisateur. Nous avons fait le choix d'un client simple afin que si le client le souhaite, puisse utiliser un client normalisé comme « netcat ».

Le serveur :

Le programme serveur créé un socket permettant d'écouter sur toutes les interfaces sur un port hardcodé dans un fichier header. Au lancement du serveur, il faut spécifier le nombre de clients et le nombre de bots. Après cela, le serveur attend la connexion de tous les clients (nous verrons cela plus en détails après). Le serveur gère l'ensemble du jeu, comme vu précédemment le client ne sert qu'à afficher les informations du serveur et envoyer des informations quand cela est demandé par le serveur. Contrairement à ce que nous avions prévu, les bots ne sont pas des processus à part entière, le serveur s'occupe donc de les gérer également.

3. LA GESTIONS DES JOUEURS

La création des joueurs :

Les joueurs sont définis par une structure joueur, ils possèdent différents attributs :
- name : le nom du joueur stocké dans un tableau de char
- cartes : adresse dans le tas du tableau contenant les cartes de la main du joueur, cette adresse change au fil des cartes que le joueur possède.

```
typedef struct{
    char name[SIZE_NAME];
    int *cartes;
    FILE *file_ptr;
    size_t nbCarte;
    size_t teteBoeuf;
    bool isBot;
}joueur;
```

-file_ptr : adresse du fichier permettant la communication avec les clients via un flux de caractère
-nbCarte : le nombre de cartes dans la main du joueur
-teteNoeuf : le nombre de têtes de bœuf possédées par le joueur
-isBot : booléen permettant de savoir si le joueur est un bot, cela permet au serveur de faire une action différente si le joueur est un bot (nous verrons cela plus en détail après).

Les joueurs sont stockés dans le tas via une structure joueurArray possédant également différents attributs :
-lst : l'adresse dans le tas du tableau contenant les joueurs
-size : le nombre de joueurs (clients+bots)
-nb_client : le nombre de clients

```
typedef struct{
    joueur *lst;
    size_t size;
    size_t nb_client;
} joueurArray;
```

La structure joueurArray est initialisé dans le tas au lancement du serveur.

La connexion des clients :

Après la création de la structure joueurArray, le serveur va attendre la connexion des clients.
Lorsque qu'un client rejoint le serveur, un thread est créé afin que celui-ci puisse choisir son pseudo et permet d'initialiser les attributs du joueur. L'utilisation des threads permet aux clients de choisir leur pseudo et d'être initialisés en même temps et donc de ne pas devoir attendre le client précédent. Cela fluidifie la connexion au serveur.

Les bots :

Comme dit précédemment, les bots ne sont pas des processus à part. De ce fait, lorsque tous les clients ont rejoint, le reste des joueurs (correspondant au bot) sont initialisés avec un pseudo prédéfini et n'ont pas de file_ptr assigné.

4. LE DEROULEMENT DE LA PARTIE

Au début d'un round, le serveur envoie les informations relatives à chaque client via la fonction fprintf(file_ptr, message) permettant d'envoyer un flux de caractère à afficher au client via le file_ptr.

Les clients :

Après avoir reçu les informations, des threads sont utilisés afin de demander les cartes à jouer à tous les joueurs. L'utilisation des threads dans ce cas permet de reproduire le déroulement réel d'une manche, donc que tous les joueurs choisissent leur carte en même temps et non pas chacun leur tour.
Afin de demander une carte, la fonction fgets(carte, SIZE_INPUT, file_ptr) est utilisée, celle-ci permet de récupérer l'entrée utilisateur saisi par le client via un flux de caractère défini par file_ptr.
Après cela, la carte de chaque joueur est jouée via différentes fonctions et les attributs des joueurs sont mis à jour.
Dans certains cas, le serveur peut demander au client de choisir la pile à enlever, cela est fait simplement avec la fonction fgets.

Les bots :

Concernant les bots, le serveur ne demande pas la carte à jouer et sélectionne automatiquement la première carte de leur main.

Amélioration « bots plus intelligent » :

Afin de répondre à l'amélioration « bot plus intelligent » nous avons décidé que lorsqu'un bot doit choisir une pile à retirer, c'est la pile contenant le moins de têtes de bœufs qui est retirée, et non pas une pile choisie aléatoirement.

Amélioration « têtes de bœuf variables » :

Afin de faire l'amélioration tête de bœufs variable, nous avons créé une fonction qui, selon les cartes à retirer, retourne le nombre de têtes de bœuf correspondant.

5. FIN DE MANCHE

À la fin d'une manche, quand toutes les cartes des joueurs ont étées jouées, une fonction utilisant des threads et la fonction fgets demandent à tous les clients s'ils veulent continuer à jouer. Si un seul des clients ne souhaite pas rejouer la partie s'arrête.
Les bots n'ont pas d'impact sur le choix d'une nouvelle manche.

6. Fin de partie

Amélioration « classement » :

Le nombre de têtes de bœuf de chaque joueur est recuperé puis trié afin d'obtenir le classement de la partie avant d'être renvoyé au client avec la fonction fprintf().

Amélioration « génération du pdf » :

Pour la génération du pdf nous avons utilisé la bibliothèque « libHaru ».

Nous l'avons utilisé de manière ce qu'elle retrace toutes les informations du serveur.

Nous l'avons d'abord initialisé :

```
/* CREATION PDF */
const char* page_title = "6quiprend";
HPDF_Doc pdf = HPDF_New(error_handler, NULL);
HPDF_Font font = HPDF_GetFont(pdf, "Helvetica", NULL);
HPDF_Page page = HPDF_AddPage(pdf);
HPDF_Page_SetHeight(page, 14400);
HPDF_REAL width = HPDF_Page_GetWidth(page);
HPDF_REAL tw;
```

Pour écriture du contenu dans la page il faut se déplacer à l'aide d'un curseur. C'est pourquoi lorsque l'on voudra écrire du contenu on utilisera la fonction HPDF_Page_MoveTextPos() qui placera le curseur dans le fichier pdf puis écrira la contenu grâce à la fonction HPDF_Page_ShowText().

```
void addLinePdf(HPDF_Page page, char* str)
{
    HPDF_Page_ShowText(page, str);
    HPDF_Page_MoveTextPos(page, 0, -15);
}
```

Une fois tout le contenu de la partie écrite on enregistre le pdf.

```
void savePdf(HPDF_Doc pdf)
{
    HPDF_SaveToFile (pdf, "History.pdf");
    /* clean up */
    HPDF_Free (pdf);
}
```