Licence 3

Rapport sur le Système et le Réseau

"6 qui prend"

Auteurs :

- DJAWOU Koffi Mario
- DETAIL Matthieu

Sommaire

1. Introduction

Le projet consiste à développer un système informatique pour le jeu de cartes "6 qui prend", où l'objectif est de minimiser le nombre de "têtes de bœuf" à la fin de chaque partie. Chaque manche débute par la formation de quatre rangées avec quatre cartes tirées au hasard, suivie de la distribution de dix cartes à chaque joueur. Pendant chaque tour, les joueurs choisissent une carte à jouer face cachée dans l'une des rangées croissantes. Lorsque certaines rangées atteignent cinq cartes, un joueur ajoutant une sixième carte les ramasse et remplace la rangée.

Si un joueur pose une carte de valeur inférieure à celles déjà présentes, il ramasse une rangée de son choix. Les scores sont calculés en fonction des "têtes de bœuf" ramassées. Le joueur atteignant 66 "têtes de bœuf" perd la partie. L'objectif est de créer une simulation numérique fidèle aux règles du jeu.

Ce projet a pour objectif de mettre en pratique les concepts de Systèmes & Réseaux en développant une version informatique du jeu de cartes "6 qui prend".

2. Architecture du Système

2.1 Vue d'Ensemble

Pour parler de l'ensemble du projet, l'essentiel du projet est fait en C sauf la partie de gestion des statistiques qui est fait lui en bash pour générer un fichier PDF à partir des données du jeu inscrites dans le fichier logStat.txt.
Pour la partie en C on a trois fichiers principaux notamment
JoueurHumain.c : qui s'occupe de faire la connexion de chaque joueur au GesDuJeu.c et lui permettant de jouer chaque manche.

JoueuRobot.c : qui s'occupé de faire la connexion de tous les robots au GesDuJeu et qui jouent automatiquement à chaque manche.

GesDuJeu.c : c'est le chef d'orchestre qui s'occupe de chaque étape du jeu et qui permet de gérer les connexions joueurs et les connexions robots.

fonGesJeu. : Elles contient tous les fonctions nécessaires pour le GesDuJeu.c.

Et on a tous les fichier header (.h) qui comprend la signature des fonctions respectives.
(fonH.h foncR.h fonGesJeu.h)

2.2 Gestionnaire du Jeu

2.2.1 Fonctions principales

- *Fonction pour gérer les cartes :*

void initPaquet(paquet *p, int capacite) :
cette fonction permet de faire une allocation de mémoire avec (malloc) pour chaque paquet de carte qui est une structure composée du nombre de carte, la capacité du paquet et un pointeur de n cartes .

void liberePaquet(paquet *p) :

Cette fonction libère l'espace mémoire associé au paquet, vidant ainsi le paquet de cartes.
void videPaquet(paquet* p) : Cette fonction initialise le nombres du carte du paquet a 0.

void afficheCarte(carte c) :
Affiche les détails de la carte spécifiée, le numéro de cartes et le nombre de tête de bœuf (🐮)
.
void affichePaquet(paquet *p) :
Affiche l'ensemble du paquet de cartes en affichant chaque carte individuellement.

carte creeCarte(int num) : Crée et retourne une carte avec le numéro spécifié en fonction du <num > entré en paramètre .

void melangePaquet(paquet* p) :
Mélange aléatoirement les cartes dans le paquet, changeant ainsi l'ordre des cartes.

void produitJeu(paquet* p) :
Remplit le paquet avec un jeu complet de cartes, en créant les 104 carte une par une.

paquet PrendrePaquet(paquet * p , int nbr) :
Crée un nouveau paquet en prenant un certain nombre (nbr) de cartes du paquet initial. Le paquet initial est modifié en conséquence.

carte PrendreCarte(paquet * p , int i) :
Crée un nouveau paquet en prenant un certain nombre (nbr) de cartes du paquet initial. Le paquet initial est modifié en conséquence.

void distributionDesCartes(joueurs jos , robots ros , paquet *p) :

Distribue les 10 cartes aux joueurs humains et aux robots.

void afficheCarteEnJeu() :

Affiche les cartes en jeu, les 4 lignes qui contiennent chacun de 0 à 5 cartes maximum.

void echangerCartes(carte *cartes, int i, int j) :
Cette fonction échange les cartes situées aux indices i et j dans le paquet de cartes.

- *Fonctions essentielles et fonctions pour gérer le réseau :*

void triABulles(paquet * p , int taille) :
Cette fonction implémente le tri à bulles sur le paquet de cartes pour les ordonner.

int trouverMinimum(int tab[] , int taille) :
Retourne l'indice du minimum dans le tableau d'entiers spécifié.

int getIDwhithNum(paquet *p , carte c) :
Retourne l'identifiant d'une carte spécifique dans le paquet en fonction de son numéro.

void serialiser_carte(const carte *c, char *buffer, size_t buffer_size) :
Sérialise une carte individuelle dans un tampon de caractères.

carte deserialiser_carte(const char *buffer, size_t buffer_size, carte *c) :
Désérialise les données d'une carte depuis un tampon de caractères.

void sendPaquet(int socket, paquet p) :
Envoie le paquet de cartes à travers une socket. En envoyant carte par carte.

paquet receivePaquet(int socket , paquet p) : Reçoit un paquet de cartes à partir d'une socket , carte par carte

carte receiveCarte(int socket , carte c) :
Reçoit une carte à partir d'une socket.

void sendCarte (int socket , carte c) :
Envoie une carte à travers une socket.

void actualisation(carte c , int index ,int idR , int idJ , int choix) : Actualise les lignes des cartes en jeu soit jouer par un joueur ou par un robot .

void initialisation() : initialise les ressources nécessaires au fonctionnement du programme

bool verifierPoints() : Vérifie les points de chaque joueur et de chaque robot au cas où quelqu'un atteins 66 .

2.2.2 Processus de Distribution des Cartes et autres processus

void *connexionHumain(void *args) :
Cette fonction gère la connexion d'un joueur humain. Elle est conçue pour gérer la communication entre le Gestionnaire du jeu avec le joueur humain.
Elle initialise la connexion et envoie les cartes aux joueurs.

void *connexionRobot(void *args) :
Cette fonction gère la connexion d'un joueur robot. Elle est conçue pour gérer la communication entre le Gestionnaire du jeu avec le joueur robot. Elle initialise la connexion et envoie les cartes aux joueurs robots.

void *FaireJouerR(void *args) :
Cette fonction représente le processus de jeu pour un robot. Elle est conçue pour donner la main aux robots pour qu'ils puissent envoyer sa carte.

void *FaireJouerH(void *args) :
Cette fonction représente le processus de jeu pour un joueur. Elle est conçue pour donner la main aux joueurs pour qu'ils puissent envoyer sa carte.

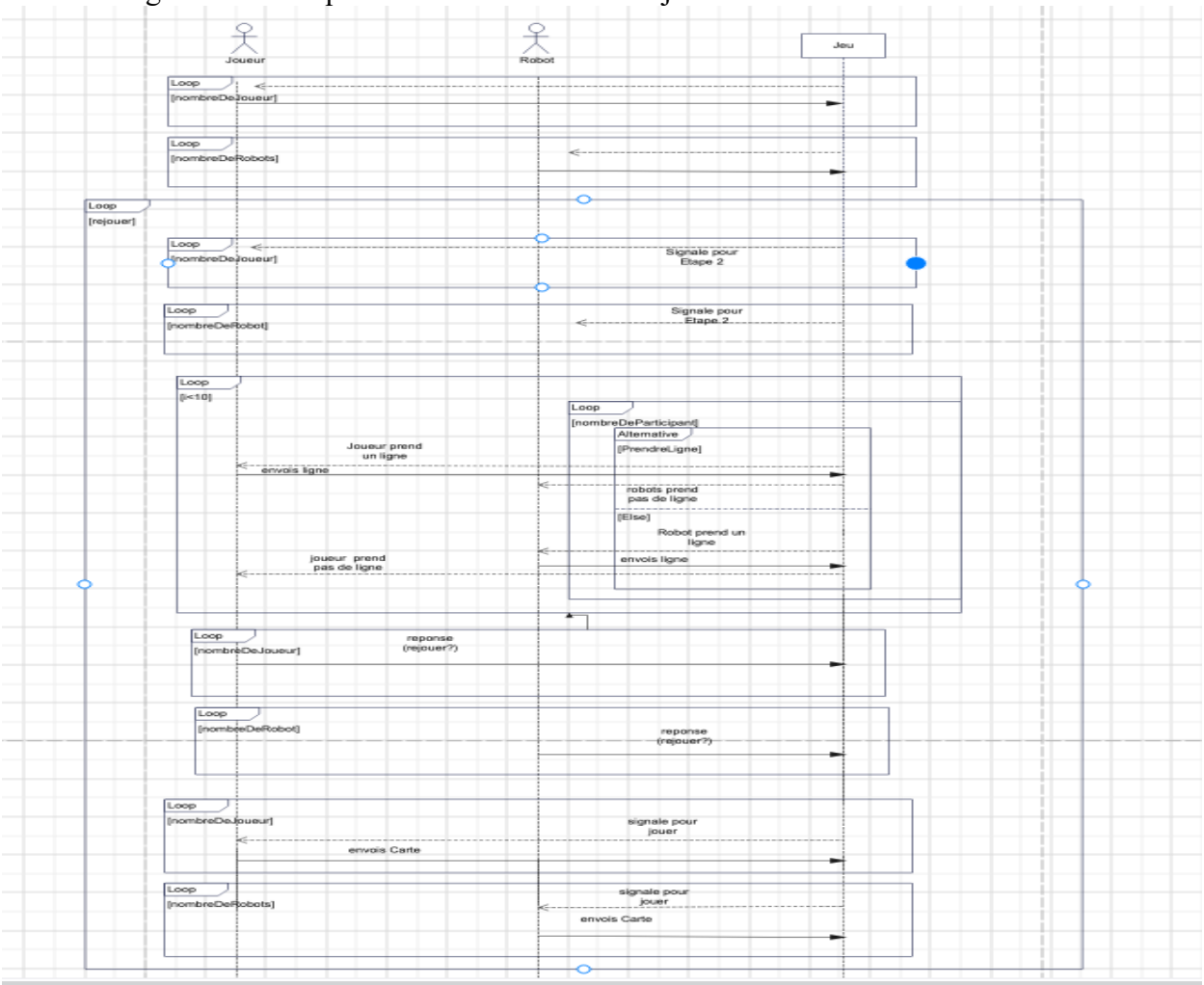
void *casserLigneRobot(void *args) :
Cette fonction gère l'action de casser une ligne de cartes par un robot.

void *casserLigneJoueur(void *args) :
Cette fonction gère l'action de casser une ligne de cartes par un joueur.

void *Jeu() : Fait tout la partie du déroulement du jeu. .

2.2.3 Gestion des Manches et des Joueurs

Voici le diagramme de séquences du déroulement du jeu :

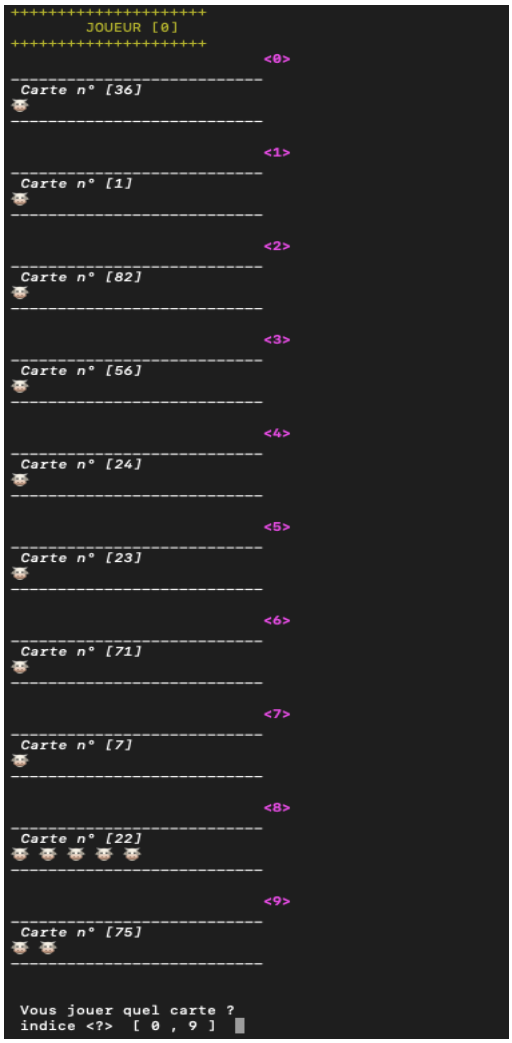


Chaque manche est constituées de 10 parties, à la fin de chaque manche les joueurs ont le choix de continuer ou de quitter le jeu. S'ils décident de continuer le jeu, le jeu se termine si au moins un joueur (robot, humain) atteint 66 points.

2.3 Joueur Humain

2.3.1 Interaction avec l'Utilisateur

A chaque connexion le joueur reçoit ses 10 cartes et qu'il envoie au gestionnaire du jeu avec les indices de carte :



2.3.2 Transmission des Ordres au Gestionnaire

Une fois que le joueur a choisi sa carte, elle envoie cette carte par la fonction sendCarte au Gestionnaire du jeu, qui va s'occuper de ranger la carte dans les cartes en Jeu. Il peut arriver que le jouer envoie une carte plus petite que les cartes en jeu. Dans ce cas il doit prendre une ligne c'est ce qu'il choisit entre 0 et 3 et l'envoie au gestionnaire du jeu.

2.3.3 Affichage des Résultats et des Cartes Jouées

Les cartes jouées sont affichées au niveau de gestionnaire du jeu après chaque envoi.
A la fin de chaque manche les résultats de la manche sont inscrits dans le fichier logStat.txt avec la date de la partie. Si le joueur perd (points>66). Le message (« Le joueur [identifiant] a perdu ») est affiché à l'écran.

2.4 Joueur Robot

2.4.1 Similitudes avec le Joueur Humain

Le joueur robot partage plusieurs similitudes avec le joueur humain. Les deux types de joueurs reçoivent des informations sur l'état actuel du jeu, interagissent activement avec le gestionnaire du jeu pour transmettre des ordres, et attendent leur tour pour jouer une carte. De plus, ils contribuent de manière dynamique à l'évolution du jeu, nécessitant une actualisation de l'interface utilisateur pour refléter les résultats de leurs actions. Bien que la principale différence réside dans la méthode de choix des cartes et des lignes, le joueur robot optant pour un choix aléatoire, ces similitudes contribuent à une expérience de jeu cohérente, où chaque joueur, qu'il soit humain ou robot, joue un rôle actif dans le déroulement de la partie.

2.4.2 Stratégies de Jeu Automatique

La stratégie de jeu automatique que nous avons adopter pour le joueur robot est caractérisée par des choix totalement aléatoires. Contrairement au joueur humain qui recourt à des stratégies basées sur l'évaluation des cartes en main, l'anticipation des actions adverses et la prise de décisions calculées, le joueur robot ne planifie pas ses coups. Sa stratégie consiste simplement à sélectionner des cartes au hasard, apportant ainsi une dimension de hasard et d'imprévisibilité à ses actions. Le joueur robot participe de manière aléatoire au déroulement de la partie.

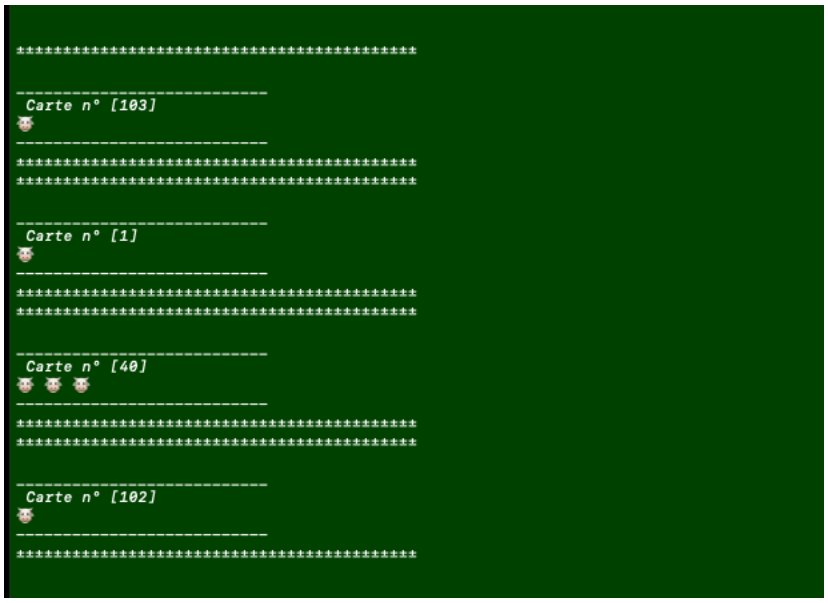
3. Fonctionnalités du Système

3.1 Accueil des Joueurs :

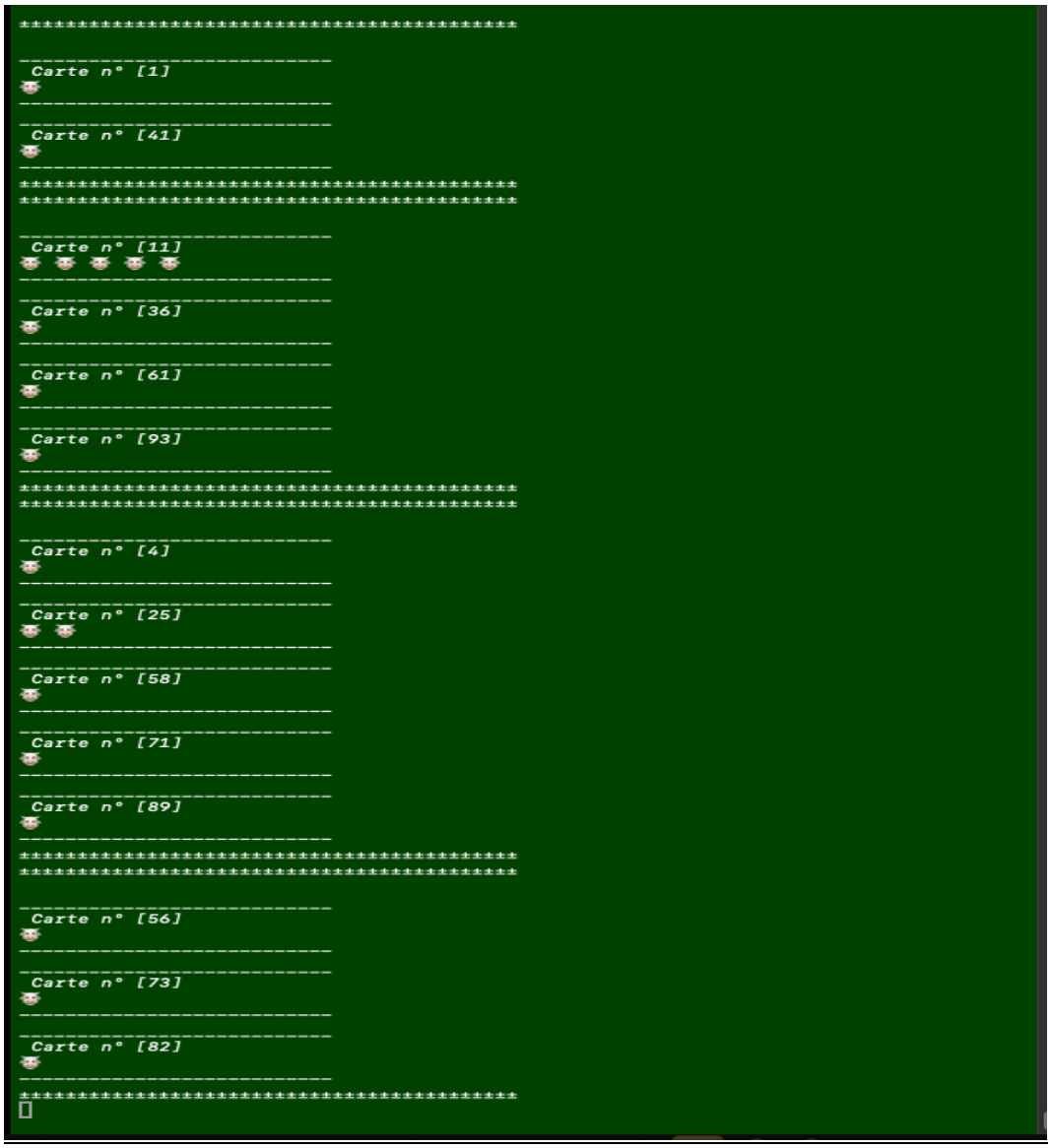
A chaque connexion des robots ou des joueurs, il y a une réception automatique des cartes mélangés qui sont différents pour chaque joueur et robots.

3.2 Dépôt des Cartes sur les Rangées

Les cartes sont placées dans une rangée de manière ascendante, c'est-à-dire que chaque carte doit avoir une valeur numérique plus élevée que la carte précédemment placée dans la rangée. Si la carte placée par un joueur ou un robot a un numéro strictement inférieur à toutes les cartes présentes dans les rangées, le joueur ou le robot choisit la rangée à prendre. Le nombre de "têtes de bœuf" sur cette rangée est alors ajouté au total du joueur ou du robot.



carte en Jeu au début du jeu.



Carte en Jeu à la fin d'une manche

les Lignes sont numérotées de 0 à 3 (Ligne 0..3) elles n'apparaissent pas sur l'image parce qu'elles clignotent

3.3Comptage des Têtes de Bœufs par Joueur :

Chaque carte "tête de bœuf" prise par un joueur ou un robot au cours de la manche contribue à son total de "têtes de bœuf". Chaque carte "tête de bœuf" a une valeur spécifique qui est ajoutée au score du joueur.

3.4 Communication d'Informations aux Joueurs

Les joueurs reçoivent des retours immédiats sur les conséquences de leurs actions, que ce soit la prise d'une rangée, le dépôt d'une carte spéciale, ou tout autre événement notable. Ce feedback renforce l'interaction et la réactivité du jeu.

4. Infrastructure Réseau

4.1 Topologie du Réseau

Nous avons utilisé une topologie client serveur avec un serveur central qui gère la logique du jeu et communique avec plusieurs clients, représentant les joueurs humains et éventuellement les joueurs robots.

Serveur de Jeu : Le serveur central assume le rôle de gestionnaire du jeu. Il est responsable de distribuer les cartes, de maintenir l'état global du jeu, de gérer les actions des joueurs, et de communiquer les mises à jour à tous les clients connectés.

Clients (Joueurs Humains et Robots) : Chaque joueur humain ou robot est un client connecté au serveur. Les clients interagissent avec le serveur en envoyant leurs actions, telles que le choix de la carte à jouer, et reçoivent en retour les mises à jour de l'état du jeu. Les clients sont également informés des actions des autres joueurs.

Communication Réseau : La communication entre le serveur et les clients est réalisée via un socket TCP.

Événements Multijoueur : Le serveur gère les événements multijoueur tels que le dépôt des cartes, la prise de rangées, et le comptage des "têtes de bœuf". Il communique ces événements à tous les joueurs pour assurer une expérience de jeu cohérente.

4.2 Protocoles de Communication

La décision d'opter pour le protocole TCP s'aligne avec la nécessité de garantir une communication stable et sans perte entre le serveur et les clients. TCP assure la livraison des données dans l'ordre, ce qui est essentiel pour maintenir la cohérence du jeu, en particulier lors du suivi des actions simultanées des différents participants.

4.3 Échanges d'Informations entre Processus

Chaque joueur, qu'il soit humain ou robot, communique avec le gestionnaire du jeu à travers les processus qui exécute les fonctions comme FaireJouerH, FaireJoueR, casserLigneJouer… Tous ces processus sont exécutés par le processus principal gestionJeu de la fonction Jeu() suivant le diagramme de séquence du jeu. .

5. Structure de Données

Une cartes :

```
typedef struct
{
    int points;
    int numero;
```

```
} carte;
```

un paquet

```
typedef struct
{
    int capa;
    int nbr; // nombre de  carte
    carte *tab;
} paquet;
```

Un Joueur

```
typedef struct {
int id  ;
int socket  ;
paquet   p  ;
int points ;
int score  ;

}joueur  ;
```

robots

```
typedef struct {
int id  ;
int socket  ;
paquet   p  ;
int points ;
int score  ;

}robot  ;
```

liste de joueurs humain. 10 au maximun

```
typedef struct {
    int nbr; // nombre de  joueurs
    joueur tab[10];
}joueurs ;
```

liste de joueurs robots 10 au maximun

```
typedef struct {
    int nbr; // nombre de  robots
    robot tab[10];
}robots ;
```

6.Génération de Statistiques

Dans cette section, nous avons élaboré un script shell qui est lancer à la fin du jeu, qui utilise le fichier crée par le gestionnaire de jeu (logStat.txt) pour générer des statistiques, telles que les points obtenus, les parties gagnées/perdues, et crée un document PDF illustrant le déroulement du jeu "6 qui prend". Le script échange des informations avec le gestionnaire de jeu et présente des fonctionnalités clés, notamment :

Statistiques Globales : Suivi des points par partie, parties gagnées et perdues, défaites du joueur et du robot, et totaux de points gagnés.

Affichage Clair : Une fonction affiche les statistiques de manière lisible, couvrant les points moyens par partie, le nombre total de parties, les défaites des joueurs et des robots, et les totaux de points gagnés.

Sauvegarde dans un Fichier Log : Les détails de chaque partie sont sauvegardés dans un fichier de log (parties.log), offrant une rétrospective détaillée.

Génération d'un Document PDF : Le script crée un fichier LaTeX (rapport.tex) pour générer un document PDF présentant les statistiques de jeu de manière structurée.

Le script analyse le fichier de log généré par le gestionnaire de jeu, extrait les informations pertinentes, calcule les statistiques, puis crée un document PDF. Cela offre une vision détaillée des performances des joueurs et des robots, facilitant l'évaluation des stratégies et l'amélioration de l'expérience du jeu "6 qui prend".

Voici un aperçu du fichier PDF :

Jeu ”6 qui prend”

DJAWOU DETAIL

Friday 22nd December, 2023

Statistiques de jeu

Points moyens par parties

Le nombre total de points par parties est : 1292.

nombre total de Manche

Le nombre totale de manche jouées : 33.

Défaites des joueurs

Le nombre de Défaites des joueurs est : 1.

Défaites des robots

Le nombre de Défaites des robots est : 0.

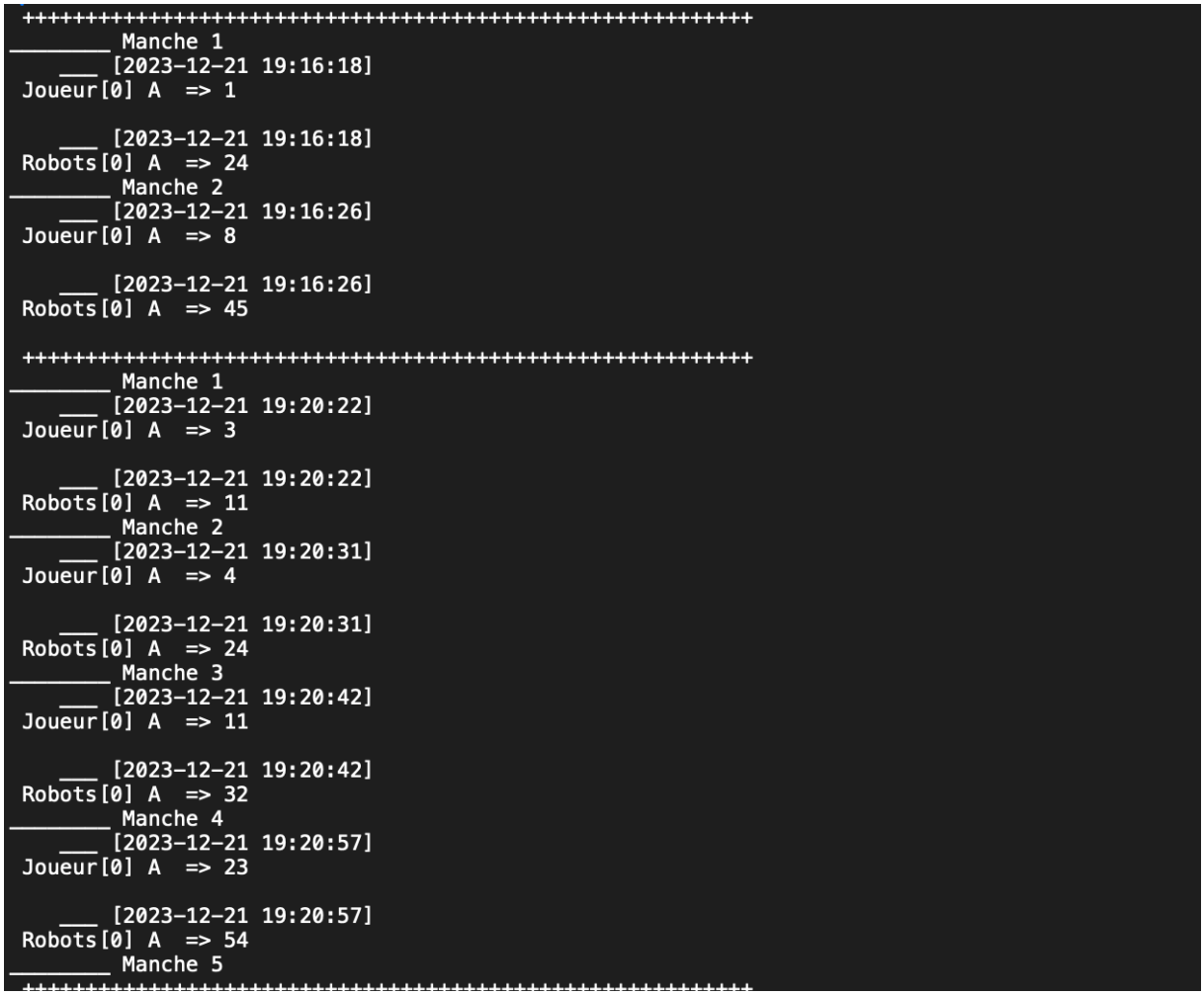
Total de points gagnés par le joueur

Le nombre points Total gagnés par le joueur : 518.

Total de points gagnés par le robot

Le nombre points Total gagnés par le robot : 774.

Un aperçu des données inscrit par le gestionnaire du jeu :



7. Conclusion

Le projet nous a permis de développer un système informatique fidèle aux règles du jeu "6 qui prend". L'utilisation d'une architecture client-serveur a facilité la communication entre les joueurs, humains et robots, et le gestionnaire du jeu. La mise en œuvre de stratégies de jeu aléatoires pour les robots a ajouté une dimension imprévisible au déroulement du jeu. L'infrastructure réseau, basée sur le protocole TCP, a assuré une communication stable entre le serveur et les clients, garantissant la cohérence du jeu. La génération de statistiques à la fin de chaque partie a ajouté une dimension analytique, permettant aux joueurs d'évaluer leurs performances et de visualiser le déroulement du jeu.

Récapitulation des Réalisations

- Architecture client-serveur pour la communication entre joueurs et gestionnaire du jeu.
- Stratégies de jeu aléatoires pour les joueurs robots.

- Fonctionnalités complètes du jeu, de l'accueil des joueurs au comptage des points.
- Génération de statistiques détaillées et création d'un document PDF.

Possibilités d'Amélioration Future

- Intelligence Artificielle Améliorée pour le robot
- Interface Graphique