Université de Franche-Comté
Département Informatique

Master 1ère année
Module SCS

RAPPORT SCS Jeu Quixo

BINOME		
Nom	**Prénom**	**Trigramme**
LOTOY BENDENGE	Vianney	LBV
NTUMBA WA NTUMBA	Patient	NTP

Année Universitaire 2014 - 2015

TABLE DES MATIERES

INTRODUCTION

L'objectif de ce travail étant l'implémentation de la communication permettant de jouer en reseau au jeu quixo dont les énoncé sont dans le sujet transmis par les enseignants.

Pour atteindre les objectifs, nous avons structuré la réalisation en trois grandes aprties parties :

- ✔ Le joueur developpé en langage C, qui va communiquer d'une part avec le moteur Java en lui transferant les coups joué par l'adversaire et en recevant du moteur Java, le coup qu'il devra jouer pour faire evoluer le jeu. Cette partie se trouve dans le repertoire joueur/bin/.

- ✔ Le moteur Java, developpé en langage Java, fait la transition entre l'Intelligence Artificielle (IA) codé en prolog et le joueur, la communication entre l'IA et le moteur Java est réalisé avec l'API Jasper tant disque le moteur Java et le joueur communique via les sockets. Cette partie se trouve dans le repertoire moteurJava/.

- ✔ La troisième partie restant est l'arbitre, cette partie regule le jeu entre deux adversaires (joueurs), il reçoit le coups d'un joueurs verifie sa validité et l'envoi à au joueur adverse ainsi de suite, tous les roles de l'arbitre sont definis dans le sujet. L'arbitre communique avec les joueurs via les sockets et ils developpés en langage C aussi. Cette partie se trouve dans le repertoire joueur/bin/.

Le projet contient en outre 3 fichiers supplementaires à la racine du repertoire :

- ✔ le fichier **README** qui explique comment lancer le joueur ;
- ✔ le fichier **compile.sh** qui permet de compiler le joueur, l'arbitre et le moteur java et permet de lancer le joueur, le lancement du joueur sous entend aussi le lancement du moteur Java.

D'une manière succinte, nous allons presenter les trois grandes parties de notre travail dans les paragraphes qui suivent.

I. LE JOUEUR

Avant de presenter notre joueur, nous allons decrire tous les comportements de son processus :

1) Le joueur envoie au serveur une requete PARTIE pour lui demander de jouer, en precisant son nom. En reponse, le serveur lui donne le nom de son adversaire et lui indique le signe de son cube. Le premier joueur connecte aura les ronds et c'est lui qui commence la partie.

2) Lorsque c'est à son tour de jouer, le joueur consulte son moteur Java pour connaître le coup à jouer, il constitue une requete COUP qu'il envoi à l'arbitre, puis attend la validation.

3) Il attend ensuite des informations sur le coup de l'adversaire, transmis par l'arbitre.

4) Si la partie n'est pas finie, le joueur informe à son moteur Java le coup de l'adversaire et retourne en 2.

Afin de respecter les comportements du joueur decrit ci-haut, un protocole a été mis à notre disposition se trouvant dans

le protocolQuixo.h que l'rbitre et le joueur doivent respecter pour pouvoir communiquer. L'ensemble de specification du processus joueur se trouve dans le sujet transmis par les enseignants.

1. Implémentation du joueur

L'implémentation du joeuer inclus le joueur.c contenant la fonction principale, le fichier fonctionJoueur.c contenant les fonctions implementant la lgique metier de notre joueur et une bibliothèque fonctionTCP pour l'appel des fonctions de de communication par socket en mode connecté.

1) Processus de connexion et déconnexion

La connexion étant la première action que initie notre joueur, selon le nom du serveur arbitre, le port de ce serveur et le nom du joueur qui lui sont transmis à l'execution, dès que ces trois éléments sont valides, le joueur ouvre premièrement une communication avec le moteur Java via le port 8888 dès que la connexion avec ce dernier sera établi, le joueur fait ensuite une demande de connexion à l'arbitre avec les paramètres passer en argument lors de son execution.

Une fois que la communication avec l'arbitre sera établie, le joueur constitue une requete de demande PARTIE en precisant son nom passé en paramètre preécédemment, la fonction ci-dessous implemente cette action

```
// Demande d'une partie au serveur arbitre
repPartie = demandePartie(sock_arb, reqPartie);
if(repPartie.err != ERR_OK)
{
    printf("Joueur: EChec demande de partie \n");
    exit(4);
}
```

de cette requete il réçoit une reponse contenant le nom de son adversaire ainsi que le signe avec le quel il va jouer Ceci signifie que la demande de PARTIE a été accptée.
Ces informations sont ensuite transmises avec un TOKEN au Moteur Java qui permettra ainsi au moteur d'identifier si réellement c'est son joueur dans le cas contraire le Moteur Java va retourner un accès refusé et va fermé la connexion.

```
// Envoi des info au moteur IA
reqInfo.token = htonl(MYTOKEN);
reqInfo.signeCube = htonl(repPartie.signe);

err = informeIA(sock_ia, reqInfo);
if(err != 1)
{
    printf("Joueur: Erreur de reception d'information du moteur IA \n");
    exit(5);
}
```

Les données transmise au moteur Java, doivent etre transformées en Big endians soit via la fonction htonl pour que ces derniers bien interpretés par la JVM du moteur Java.

Au cas où il y a succès, le Moteur Java selon qu'il est le premier à jouer va calculer le coup et l'envoyer au joueur si non il va attendre que le joueur lui envoi le coup joué par l'adversaire.

La deconnexion intervient soit en cas d'erreur ou soit si la partie du jeu arrive à sa fin.

2) Envoi du premier coup

Le joueur ayant le signe rond sera le premier à jouer, de ce fait la fonction aLeSigneRond qui pour argument la socket de communication de l'arbitre et celle du moteur Java et va ainsi envoyer une demade de coup au moteur Java, ce dernier va lui en retourner une via la fonction demandeIaCoup qui prend en argument que la socket et retourne un structure de type TypeReqCoup, les données reçues du moteur Java doivent etre transformé en little indian pour que gcc puisse bien les exploiter, cette transformation se fait via la méthode ntohl. Une fois que le joueur reçoit le coup du moteur il va le transferé à l'arbitre, via la fonction ci-dessous :

```
TypCoupRep envoiArbitreCoup(int fd_sock_arbitre, TypCoupReq reqCoup)
```

qui retourne la validation du coup joué, si la coup est valide, alors la fonction aLeSigneRond va retourner une valeur 0 dans le cas contraire la fonction va retourner une valeur differente de 0.

3) Deroulement du jeu

C'est la fonction deroulerPartie ayant pour argument deux sockets de communication celle du moteur Java et celle de l'arbitre ainsi que le signe du joueur qui l'execute, qui va executer le deroulement du jeu. Ci-dessous le prototye de la dite fonction :

```
/**
 *  derouler La partie
 * */
void deroulerLaPartie(int fd_sock_ia, int fd_sock_arbitre, char monSigne);
```

Cette fonction debute par attendre premièrement la validité du coup joué par l'adversaire et ensuite le coup de l'adversaire, ceci est réalisé grace la fonction demandeArbitreCoup, qui prend en paramètre la socket de communication avec l'arbitre et retourne un coup de type TypeReqCoup valide si il a été validé, dans le cas contraire il ne sera pas valide et le deroulement du jeu va s'arreter.

Ayant un coup valide de l'adversaire, le jouerur va envoyer ce dernier au moteur Java via la fonction ci-dessous :

```
/**
 *  envoi le coup au moteur IA et retourne un coup que l'IA a calculé et joué sur le plateau
 * */
TypCoupReq envoiIACoup(int fd_sock_ia, TypCoupReq reqCoup);
```

Prenant en paramètre la socket de communication avec le moteur Java et le coup joué par l'adversaire, cette fonction envoi le coup au moteur java et ce dernier retourne un coup qu'il aura calculé, le moteur Java defini au préalable toutes les propriétés du coup avant de l'envoyer et sur base de la verification de propriétés du coup, la fonction deroulerPartie est en mesure de savoir si le coup est gagnant, nulle, perdu ou deplace cube et va executer une action selon qu'il lui est permis lors de l'implementation.

Une fois le coup reçu du moteur Java, la fonction va ainsi faire appel à la méthode envoiArbitreCoup, déjà expliqué à la deuxième action du joueur.

La fonction deroulerLaPartie est executer recursivement jusqu'à ce qu'on atteigne la fin de la partie qui est declenché dès la condition ci-dessous n'est plus respecté :

```
}
while(reqCoup_ia.propCoup == DEPL_CUBE && repCoup.validCoup == VALID);
```

4) Quelles que particularité d'implémentation

Le joueur peut aisement detecté une reponse de TIMEOUT provenant de l'arbitre meme si le Moteur Java est entrain de calculer le coup, et de cette manière le jeu sera arreté et le moteur sa requete n'arrivera pas.

2. Difficultés rencontrées

Si nous avons connu des difficultés c'est par ce que nous n'avons pas bien lu le sujet et respecter le protocole, et la reception des données provenant de Java étaut aussi un casse-tete mais que nous avons arriver à resoudre en utilisant un buffer en le remplissant de toutes les données qui sont envoyées par le moteur Jaava juqu'à ce que la taille des données reçues correspondent réellement à ce que le joueur s'attend. Le code ci-dessous illustre cette implementtation.

```
char BufferIA[1024];
memset(BufferIA, '\0', sizeof(BufferIA));
int err = 0;
printf("Joueur: Attente du coup provenant de l'IA ! \n");
while ( err < (sizeof(int) * 5))
{
   int nbytes = recv(fd_sock_ia, &BufferIA[err], 1, 0);
   if ( nbytes >=  0 ) { err += nbytes;}
}
```

II. LE MOTEUR JAVA

Le moteur Java de notre programme se trouve, comme décrit précédemment, dans le dossier moteurJava/src et les fichiers .class dans le dossier moteurJava/bin. Ce moteur permet d'envoyer au joueur (codé en C), les coups choisis par l'IA et également de recevoir depuis l'arbitre les coups joués par l'adversaire afin de mettre à jour le plateau.

La classe qui permet de lancer le moteur Java se nomme JeuQuixo. Ce moteur possède donc une classe principale qui permet d'envoyer et de recevoir des coups avec le joueur, en utilisant des sockets, qui se nomme JeuQuixoCom. Cette classe doit donc faire des demandes de coups à l'IA (programmé en Prolog), via la classe QuixoProlog. Cette dernière possède une méthode calculerCoup qui permet de retourner un coup choisi par l'IA. D'une manière succinte nous allons decrire quelques classes dans le paragraphes suivants.

1. Présentation de la classe Protocole

La classe Protocole nous permet de respecter le protocole de communicatione entre le Moteur Java et le Joueur developpé. Cette classe permet de faire une parfaite correspondance avec le fichier protocole fournit pour la communication entre le joueur et l'arbitre.

2. Présentation de la classe QuixoProlog

Cette classe a son importance qui est celle d'assurer la communication entre Java et Prolog via l'API Jasper, en dehors du constructeur quellesques méthodes usuelles, cette classe implémente les méthodes ci-dessous :

- ✔ initPlateau : qui permet d'initialiser le plateau de jeu quixo ;
- ✔ une méthode booléenne calculerCoup qui permet de lancer alpha-beta pour que l'IA puisse calculer un coup, si cette méthode reussier, elle met à jour un structure HashMap qui contiendra notre coup que nous allons jouer sur le plateau initialiser au paravant.
- ✔ La méthode jouerCoup qui prend en paramètre un plateau, permet de jouer un coup soit provenant de l'adversaire soit celui calculer par l'IA, cette methode retourne un nouveau plateau mis à jour.
- ✔ La methode booléenne gagne qui prend en paramètre un plateau et un signe permet d'evaluer si le plateau est à un état gagnat avec le signe passé en paramètre, si il reussi, la fonction retourne une valeur vraie dans le cas contraire elle retourne une valeur fausse.
- ✔ La méthode signecaseDepart permet de retourner le signe d'une case de mon plateau lorsqu'on lui passe une numéro de case.

3. Présentation de la classe Console

Cette classe nous permet d'affciher tous les message, les erreurs et le plateau.

4. Présentation de la classe QuixoCom

Cette classe assure la communication entre l'IA et le Moteur Java en soit via la classe QuixoProlog et entre le moteur Java et le Joueur via le socket.

Pour la transmission des données nous utilisons l'object de type primitif des octets DataInputStream et

DataOutPutStream, la socket java est lancée sur le port 8888 et la mise en place de cette communication n'a pas vraiment des difficultés mise à par une bonne gestion des exceptions.

Ainsi cette classe permet de derouler les jeux avec les méthodes ci-dessous :

- ✔ la méthode deroulement effectue le deroulement du jeu en recevant les données provenant du joueur les met à jour sur le plateau, teste si le plateua n'est pas gagnat, verifie s'il peut annoncer qu'il y a match null ou pas, elle fait une démande de calcul de coup au moteur IA, le recupère met de nouveau à jour le plateau prepare les données à envoyer selon l'état du plateau et enfin envoi les donnée sur le réseau à destination du joueur.

- ✔ La méthode envoiReseau, assure la transmission des coups vers le joueurs via les sockets.

- ✔ La méthode run, qui est la méthode prinviapale, traite la première reque du joueur, fait appel à la méthode d'authentification et ensuite envoi le résultat au joueur via la méthode envoiReseau, ensuite la méthode run s'assure de faire une démande de calcul de coup si le joueur est de signe rond en l'envoi sur le réseau et ensuite fait appel à la méthode deroulement pour continuer le jeu.

5. Présentation de la classe principale JeuQuixo

Cette classe instancie la classe QuixoCom, et fait appel à la méthode run sur cet objet et le moteur Java en marche.

III. L'ARBITRE

Serveur de jeu, developpé en C assure la communication entre deux joueurs, il est developpé avec une architecture de multiplexage avec select des sockets de communications afin de gerer nos deux joueurs.

Il gère les la demande de la partie en determinant les tours au joueurs, selon que le premier à se connecter va demarrer la partie en ayant le signe rond et le second à se connecté aura le signe croix et sera le second à jouer. En dehors de l'initialisation de la partie il informe à chaque joueur son signe et le nom de l'adversaire. Ce fonctionnement est mis en œuvre grace à la fonction ci-dessous : attenteDemandePartie.

Une fois que cette étape est executée avec succès la fonction principale fait appel à la fonction ci-dessous :

```
/**
 *  Gère toutes les communications des coups envoyés par les joueurs, les faits valider et
 *   informe à chaque joueur du deroulement du jeu
 * */
int receptionCoup(int tab_socket[], fd_set * fdset);
```

Cette fonction gère les coups joués par les entagonistes, grace à la fonction d'évaluation fournie, elle evalue un coup et affiche le plateau du jeu en affichant les messages liés à ce coup joué.

La reception étant bloquantes, nous avons mis en œuvre une méthode d'attente des données avec un delai de 6 secondes au délà de ces delai l'arbitre retourne une erreur de TIMEOUT au joueur qui a trainé à envoyer le coup et la partie s'achève.

CONCLUSION

La mise en œuvre de ce projet, nous a permis d'approfondir la notion de communication en via socket entre Java et le langage C.

Nous avons aussi mis en œuvre des nouveaux concepts de multiplexage avec timeout chose que nous n'avons pu mettre en œuvre pendant les TP.