Université de Franche-Comté
Département Informatique

Master 1ère année
Module SCS

Projet jeu Ultimate Tic-Tac-Toe
M1 SCS

BINOME		
Nom	**Prénom**	**Trigramme**
LOTOY BENDENGE	Vianney	LBV
OUSLEYEH BILEH	Yassin	OBY

Année Universitaire 2015-2016

Sommaire :

Université de Franche-Comté
Département Informatique

Master 1ère année
Module SCS

I- Introduction :

L'objectif de ce travail étant l'implémentation de la communication permettant de jouer en réseau au jeu Ultimate tic-tac-toe.
Pour ce faire, le travail réalisé est structuré trois parties dans le repertoire /TicTacToe_Com:

1. Le développement d'un joueur en langage C, destiné à communiquer avec le moteur IA en Java en envoyant les coups joué de l'adversaire et en recevant de lui, les coups qu'il doit jouer à envoyer au serveur arbitre en langace C.

2. Le développement d'un serveur arbitre en langage C, celui-ci assure le bon déroulement de la partie entre les deux joueurs adverses, il reçoit le coups d'un joueurs vérifie sa validité et l'envoie aux joueurs concerné à son adversaire et vice versa. Le serveur arbitre communique avec les joueurs via les sockets.

3. Le développement d'un moteur IA en Java, qui fait l'interface entre l'Intelligence Artificielle (IA) codé en prolog et le joueur, la communication entre l'IA et le moteur IA s'effectue par le biais de l'API Jasper, et par la suite, la communication entre le moteur IA et le joueur se fait par socket.

Outre cette partie, le projet contient en outre 3 fichiers supplementaires à la racine du projet /LotoyOusleyeh_TicTacToe:

- Le fichier README qui explique comment lancer le joueur ou le serveur arbitre;
- Le fichier joueurTicTacToe.sh qui permet de compiler le moteurIA d'abord ensuite le joueur ;
- Le fichier serveurTicTacToe.sh qui permet de compiler le serveur arbitre.

II. UTILISATION DU PROTOCOLE DE COMMUNICATION AVEC LE SERVEUR

De prime abord, on va décrire tous les processus liés à la communication entre le Joueur et le serveur :

1) Le joueur envoie au serveur une requête PARTIE pour lui demander de jouer, en précisant son nom. En réponse, le serveur lui donne le nom de son adversaire et lui indique son symbole. Le premier joueur connecté aura les croix et c'est lui qui débutera la partie.

2) Lorsque c'est à son tour de jouer, le joueur consulte son moteurIA pour connaître le coup à jouer en constituant une requête COUP qu'il enverra par la suite au serveur arbitre, puis en attendra la validation.

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

4) Si la partie n'est pas finie, le joueur informe à son moteurIA du coup de l'adversaire et retourne en 2. Afin de respecter les comportements du joueur décrit ci-haut, un protocole a été mis à disposition dans protocoleTicTacToe.h qu'on doit respecter pour la communication entre le serveur arbitre et le joueur.

III. LE MOTEUR IA

La classe *Moteur* permet de lancer le moteurIA développé en java. Ce moteur possède donc une classe principale qui permet de communiquer avec le joeueur, d'envoyer et de recevoir des coups en utilisant une socket. Outre la communication avec le joueur, cette classe permet également de communiquer avec la classe *CommunicationIA* chargé d'envoyer et de recevoir des informations à la partie IA en Prolog.

1. Présentation de la classe Coup

La classe Coup possède les attributs suivants :

```
public Symbol symbol( Signe (X/O) du coup).
public Position pos (un objet de type Position possédant :
       un Enum numPlat précisant A,B,C.....I,
       un Enum numSousPlat précisant 1,2,3,.....9).
public IdRequest idrequete ( un Enum précisant :PARTIE, COUP).
public int NbrSousPlateauG = 0 (un int étant le nombre de sous-plateau gagné).
public TypeCoup type (un Enum précisant CONT, GAGNANT, NULLE, PERDU).
```

Donc le coup que nous envoyons au joueur, est structuré et bien défini dans l'API java.

2. Présentation de la classe CommunicationClientC

La classe CommunicationClientC est primordiale dans la communication entre le Joueur et l'API java.
Les méthodes **receptionSigne, `receptionToken`** font respectueusement la réception du mot de passe et du symbole du joueur défini dans la structure entre le joueur et le moteurIA dans *protocoleMoteurIA.h* (le token est défini par défaut à 2016 par le joueur).

Les méthodes **`receptionCoup, receptionTypCoup`** font respectueusement la réception du :

- Coup :

```
public static Coup receptionCoup(DataInput dis) throws ReceptionInvalideException,
IOException, EOFException {
       [...]
       id = IdRequest.setIdRequest(dis.readInt());
       symbol = Symbol.setSymbol(dis.readInt());
       numPlateau = TypSousPlateau.setTypSousPlateau(dis.readInt());
       numSousPlat = NumCaseSousPlat.setNumCaseSousPlat(dis.readInt());
       Position pos = new Position(numPlateau, numSousPlat);
       int NbrSousPlateauG = dis.readInt();
       return new Coup(id, symbol, pos, NbrSousPlateauG);
}
```

- Type du Coup (ici nous recevons la validation du coup):

```
public static TypeCoup receptionTypCoup(DataInput dis) throws
ReceptionInvalideException, IOException, EOFException {
	TypeCoup typCoup = null;
	int a = dis.readInt();
	return typCoup;
}
```

Et pour finir l'envoi de notre coup est réalisé dans la méthode **envoiCoup** :

```
public static void envoiCoup(DataOutput dos, Coup coup) throws IOException {
	/*Envoie du IdRequest*/
	switch(coup.idrequete.getIdRequest()){
		case PARTIE:dos.writeInt(0);break;
		case COUP:dos.writeInt(1);break;
	}
	/*Envoie du SymbolJ*/
	switch (coup.getSymbol()) {
		case ROND:dos.writeInt(0);break;
		case CROIX:dos.writeInt(1);break;
		default :break;
	}
	/* Envoi du TypSousPlateau de coup joué */
	switch (coup.pos.numPlat.getTypSousPlateau()) {
		case A:dos.writeInt(0);break;
		case B:dos.writeInt(1);break;
		case C:dos.writeInt(2);break;
		case D:dos.writeInt(3);break;
		case E:dos.writeInt(4);break;
		case F:dos.writeInt(5);break;
		case G:dos.writeInt(6);break;
		case H:dos.writeInt(7);break;
		case I:dos.writeInt(8);break;
	}
	/*Envoie de la case du SousPlateau joué*/
	dos.writeInt(coup.pos.numSousPlat.getStringSPTerm() - 1);
	/*Envoie du nombre de sousplateau gagné*/
	dos.writeInt(coup.NbrSousPlateauG);
}
```

3. Présentation de la classe principale Moteur

La classe Moteur est la classe principal, elle fait appel à tous les autres (CommunicationIA, CommunicationClientC, Coup ...etc).
Elle gère aussi le déroulement du jeux tel que le premier joueur à jouer doit être le joueur X,
et le second serait O.
Elle gère aussi de trouver le bon joueur qui se connectera à notre IA, car le mot de passe est aussi initialisé du coté du moteur IA à 2016.
Donc si jamais le joueur n'envoie pas le bon mot de passe la connexion est interrompu.

IV. JOUEUR

1. Processus de connexion et déconnexion

La connexion est initié d'abord par le joueur selon les arguments passés au lancement (à savoir : hostServeur portServeur nomJoueur portIA) qui une fois valides, ouvrent la connexion entre le joueur et le moteurIA ensuite avec le serveur via les ports définis en argument. Par la suite, le joueur fait une requête de demande de PARTIE en précisant son nom passé en argument lors du précédent lancement au serveur arbitre qui assuré par la fonction ci dessous :

```
/* Demande d'une partie */
rep = lancementPartie(sock, part);
```

Ensuite ladite fonction lui renvoie son symbole attribué par le serveur arbitre avec le nom de son adversaire, s'il se connecte en premier au serveur arbitre son symbole est croix (x) au cas contraire s'il est le deuxième connecté son symbole sera rond (o) qu'il devra envoyé à son moteurIA pour la préparation du premier coup à jouer pour croix et attendre le premier coup de l'adversaire pour rond.
Le symbole du joueur reçu sera ensuite envoyé au moteurIA avec TOKEN d'authentification qui s'il est valide le moteur IA reconnaitrait le joueur avec lequel il doit communiquer sinon le joueur n'est pas reconnu et la connexion est fermée, ceci permet de garantir une communication sécurisée entre le moteurIA et seul joueur autorisé à établir une communication avec lui.
Les données transmises seront au préalable converties en Big endians avant l'envoi via la fonction *htonl* pour permettre à la jvm de mieux les interpréter pour le moteurIA. Lorsque cette opération se déroule normalement, moteurIA prépare le premier coup du joueur si son symbole est croix sinon s'il est rond, il se prépare à recevoir le premier coup de son adversaire et ne se déconnectera que si le jeu se termine par une partie gagnée, un match nul, une partie perdue ou une partie terminée par time out.
La fonction ci-dessous permet d'assurer l'envoi de ces informations au moteurIA :

```
/* Envoi des items au moteur IA pour authentification de la connexion */
    item.token = htonl(TOKEN);
    item.symbole_Joueur = htonl(rep.symb);

 /* Envoi du signe au moteur IA*/
    envoiSigneIA(sock_ia, item);
```

2. Déroulement de la partie

La fonction *deroulement_Partie* assure le déroulement de la partie, elle comporte comme argument deux sockets de communication celle du serveur arbitre et celle du moteurIA ainsi que la réponse à la requête PARTIE reçue précédemment contenant le symbole du joueur qui sera utilisé par cette dernière, selon que le joueur est croix ou rond. En voici le prototype :

```
/* deroulement de la partie */
    deroulement_Partie(sock,sock_ia,rep);
```

Dans cette fonction le déroulement du jeu dépend du symbole du joueur, si son symbole est croix alors la partie débutera pour ce dernier par la réception du coup du moteurIA qui est converti de big endians en little endians avec la fonction *ntohl()* dans les limites de durée d'attente de six secondes sinon recevra un time out du serveur arbitre par la fonction *ReceptionTimeOut_Serveur() ;* ensuite sera transmis au serveur arbitre en faisant appel à la fonction *EnvoiCoupServeur()* qui prend en argument la socket de communication avec le serveur arbitre et la structure du coup reçu du moteurIA.
Voici les prototypes des dites fonctions:

```
/*
* Fonction : reception_envoie_CoupIA
* Paramètres :  int ia_sock, socket de connexion au moteur IA
*                    int sock_serv, socket de connexion au serveur arbitre
* Retour : TypCoupReq
* Description : permet de recevoir le coup a jouer du moteur IA.
*/
TypCoupReq reception_envoie_CoupIA(int ia_sock, int sock_serv)

/*
* Fonction : ReceptionTimeOut_Serveur
* Paramètres :  int socket_serv, socket de connexion au serveur arbitre
* Retour : TypCoupRep
* Description : permet d'indiquer que le joueur a recu un time out du serveur
*/
```

```
void ReceptionTimeOut_Serveur(int socket_serv)
```

```
/* Envois du coup au serveur arbitre        */
EnvoiCoupServeur(sock_serv, RepcoupIA);
```

Et par la suite, le joueur attend de recevoir la validation de son coup du serveur arbitre et va la transmettre directement au moteur IA en appelant la fonction *Reception_ValiditeCoup_Serveur() qui prend* en arguement la socket de communication avec le serveur arbitre et celle du moteur IA.

```
/*
* Fonction : Reception_ValiditeCoup_Serveur
* Paramètres :  int serveur_sock, socket de connexion au serveur arbitre
* Retour : TypCoupRep
* Description : permet de recevoir les informations de validation du coup de l'adversaire
*/
void Reception_ValiditeCoup_Serveur(int serveur_sock, int sock_ia)
```

Ensuite le joueur attend de recevoir le coup joué par l'adversaire du serveur arbitre en faisant appel à la fonction *ReceptionCoup_adversaire* qui renvoie *TypCoupReq* contenant la structure du coup du joueur adverse et sera envoyé au moteurIA par le biais de la fonction *envoiCoupAdvIA() qui prend en argument la socket de communication avec le moteurIA et le coup adverse de type TypCoupReq* et après il attend de recevoir également la validation dudit coup adverse du serveur arbitre par la fonction *Reception_ValiditeCoup_Serveur(),* qu'il enverra par après au moteurIA.

Ci-dessous les prototypes des fonctions :

```
/*
* Fonction : ReceptionCoup_adversaire
* Paramètres :  int serveur_sock, socket de connexion au serveur arbitre
* Retour : TypCoupReq
* Description : permet de recevoir le coup joue par l'adversaire du serveur arbitre
*/
TypCoupReq ReceptionCoup_adversaire(int serveur_sock)
```

```
/*
* Fonction : envoiCoupAdvIA
* Paramètres :  int sockIA, socket de connexion au moteur IA
*                TypCoupReq coupIA, requete du coup adverse recu
* Retour : void
* Description : permet d'envoyer le coup de l'adversaire au Moteur IA
*/
void envoiCoupAdvIA(int sockIA, TypCoupReq coupIA)
```

```
/*
* Fonction : Reception_ValiditeCoup_Serveur
* Paramètres :  int serveur_sock, socket de connexion au serveur arbitre
* Retour : TypCoupRep
* Description : permet de recevoir les informations de validation du coup de l'adversaire
*/
void Reception_ValiditeCoup_Serveur(int serveur_sock, int sock_ia)
```

Ainsi tous ces processus ce feront itérativement jusqu'à ce que la partie se termine par un gagnant, un match nul, un time out ou une partie perdue.

Dans le cas où le symbole du joueur est rond (o), c'est le même déroulement que pour le joueur ayant le symbole croix sauf qu'il ne débute pas de la même façon. Avant d'entamer le processus itératif, le joueur au symbole rond doit d'abord faire trois processus non itératif, recevoir le coup joué par l'adversaire par la fonction *ReceptionCoup_adversaire()* décrite ci-haut, ensuite l'envoyer *envoiCoupAdvIA* au moteurIA pour calculer le prochain coup à jouer, après recevoir la validation dudit coup du serveur arbitre qu'il envoie au moteurIA par la fonction *Reception_ValiditeCoup_Serveur()*. Et tout comme le joueur au symbole croix, la partie ne s'arrête pour le rond qu'en cas de match nul, partie gagnée, partie terminée par time out ou partie perdue.

V. SERVEUR ARBITRE

Serveur arbitre développé 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 demandes de partie en déterminant les symboles des joueurs, selon que le premier à se connecter va demarrer la partie en ayant le signe croix et le second à se connecter aura le symbole rond et sera le second à jouer. Ensuite il informe à chaque joueur de son symbole et du nom de l'adversaire.
Ce fonctionnement est assure par le biais de la fonction ci-dessous : *receptPartie()*

```
/*
* Fonction : receptPartie
*
* Paramètres : int sCom[], tableau des valeurs de socket de communication
*              TypPartieReq req1, requete de demande de partie du Joueur 1
*              TypPartieReq req2, requete de demande de partie du Joueur 2
*              int iterator, valeur d'iteration de la boucle
*
* Retour : TypPartieReq
*
* Description : permet de recevoir les requetes de demande de partie des joueurs
*/
TypPartieReq receptPartie(int sCom[], TypPartieReq req1, TypPartieReq req2, int iterator)
```

Dès que ce processus réussit son exécution, la fonction principale fait appel à la fonction ci-dessous pour recevoir les coups joués par les deux joueurs dans une durée d'attente de six secondes pour chacun d'eux sinon le serveur arbitre renvoie un time out au joueur fautif et la partie se termine.

```
/*
* Fonction : receptCoup
* Paramètres : int sCom[], tableau des valeurs de socket de communication
*              int NumJoueurs, valeur de l'iteration de la boucle
* Retour : TypCoupReq
* Description : permet de recevoir les coups des joueurs pour le deroulement de la partie
*/
TypCoupReq receptCoup(int sCom[], int NumJoueurs)
```

Cette fonction permet au serveur arbitre de recevoir les coups joués par les deux joueurs, ensuite appelle *envoiCoupAdverse()* qui fait l'envoi des coups aux joueurs adverses puis fait appel à la fonction de validation *validationCoupJoueur()* faisant appel à son tour à la fonction de validation

fournie permettant d'evaluer un coup et d'afficher le plateau du jeu en affichant les messages liés à ce coup joué, et renvoie ensuite une valeur de type *TypCoupRep* renvoyé par la suite au joueur et à son adversaire par la fonction *validiteCoupJoueur().*
Tous ces processus se déroulent de manière itérative jusqu'à ce que la partie se termine par un match nul, une partie gagnée, perdue ou un time out et le serveur arbitre se déconnecte des deux joueurs en fermant la socket de communication.

Ci-dessous les prototypes des fonctions citées ci-haut :

```
/*
* Fonction : validationCoupJoueur
*
* Paramètres : int sCom[], tableau des valeurs de socket de communication
*              int iterator, iterateur de la boucle
*              TypCoupReq prop_coup1, coup recu du joueur 1
*              TypCoupReq prop_coup2, coup recu du joueur 2
*
* Retour : TypCoupRep
* Description : permet de valider les coups des joueurs
*/
TypCoupRep validationCoupJoueur(int sCom[], int iterator, TypCoupReq prop_coup1, TypCoupReq prop_coup2)


/*
* Fonction : envoiCoupAdverse
* Paramètres : int sock_arbitre, socket de communication
*              TypCoupReq reqt, requete du coup recu des joueurs
* Retour : void
* Description : permet d'envoyer les coups adverses des joueurs
*/
void envoiCoupAdverse(int sock_arbitre, TypCoupReq reqt)

/*
* Fonction : validiteCoupJoueur
* Paramètres : int sCom[], tableau des valeurs de socket de communication
*              int k, iterateur de la boucle
*              TypCoupRep repValCoup, validite du coup du joueur
* Retour : void
* Description : permet d'envoyer les informations de validations des coups adverses aux joueur
*/
void validiteCoupJoueur(int sCom[], int k, TypCoupRep repValCoup)
```

CONCLUSION

La réalisation de ce projet, nous a permis d'approfondir la notion de communication avec les sockets entre Java et le langage C.
Nous avons également réalisé un nouveau concept du multiplexage, le time out, qu'on n'a pu mettre en application lors des TPs.