Guillaume MONTAVON
Benoît MEILHAC

SCS

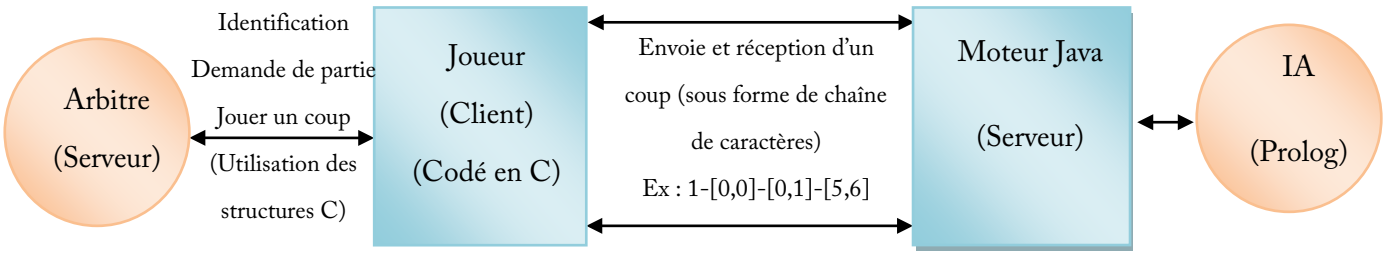
Rapport de projet : Le jeu de Yoté

1. Présentation de la structure du projet

Notre programme se décompose en 2 grandes parties :

- le **joueur** qui va communiquer avec l'arbitre (codé en C), et qui enverra et recevra des coups. Cette partie se trouve dans le répertoire *src/joueur/*.
- le **moteur Java** qui va faire la transition entre l'IA (codé en Prolog) et le joueur (connexion au joueur via des sockets, connexion au Prolog via Jasper). Cette partie se trouve dans le dossier *src/moteurJava/*.

Schéma de la communication dans notre projet de Yoté :



3 autres fichiers, se trouvent à la racine de notre projet :

- **README** : explique comment lancer notre programme
- **Compile.sh** : script shell qui permet de compiler le moteur Java ainsi que le joueur
- **Joueur.sh** : script shell qui lance notre programme avec en paramètre l'adresse de l'arbitre suivi du port de l'arbitre (suivi éventuellement le port du moteurJava).
  - **./joueur.sh lx12 5000** lance le programme en se connectant à l'arbitre se trouvant sur lx12 avec le port 2222
  - **./joueur.sh** lance le programme avec les configurations de l'arbitre décrites dans ce même fichier aux lignes 17 et 18 (portArbitre=5000 et adresseArbitre=lx12)

Les parties suivantes vont aborder, la présentation des 2 parties et des protocoles utilisés par ces dernières.

2. Présentation du joueur

2.1. Le sujet

Le code C permet la communication entre le code Java, qui récupère les résultats de l'intelligence artificielle développée en Prolog, et l'arbitre accessible sur le réseau.

En reprenant les spécifications demandées, voici comment doit fonctionner la partie en C :

1. il envoie à l'arbitre une requête **IDENTIFICATION** pour s'inscrire ;
2. lorsqu'il est prêt, il envoie une requête **PARTIE** à l'arbitre pour lui demander de jouer. En réponse l'arbitre lui donne l'identificateur du joueur avec lequel il va jouer et lui indique s'il doit commencer. Si le tournoi est fini, l'arbitre en informe le joueur qui doit terminer son application ;
3. lorsque c'est à son tour de jouer, le joueur consulte son moteur pour connaître le coup à jouer. Il constitue une requête **COUP** qu'il envoie à l'arbitre, puis il en attend la validation ;
4. il attend le coup de l'adversaire ;
5. il informe son moteur du coup joué par l'adversaire ;
6. si la partie est finie, le joueur retourne en **2** sinon il retourne en **3**.

Afin de mettre en place cette spécification, le fichier *protocolArbitre.h* a été mis à notre disposition. Il contient le protocole d'accès à l'arbitre. Ce protocole est défini dans le fichier pdf contenant l'énoncé du sujet.

2.2. Explication sur le code

2.2.1. Processus de connexion et déconnexion

La connexion à l'arbitre se fait en respectant le protocole demandé.

La fonction **creationConnexion** permet d'établir une connexion sur une machine donnée selon le port voulu. On passe en référence en entier qui contiendra, si la connexion est bien établie, le descripteur correspondant à la socket nouvellement créée. Cette fonction retournera un code de type **RetourFonction** permettant de savoir si la demande a été correctement effectuée.

La fonction **deconnexion** permet de fermer proprement la socket liée à l'arbitre. Le joueur est informé de l'état de cette déconnexion, si jamais elle ne se passe pas correctement, alors le code de type **RetourFonction** correspondra à un échec de déconnexion.

2.2.2.Processus d'identification

L'identification sur l'arbitre se fait en respectant le protocole demandé.
La fonction **identification** permet de récupérer l'identifiant que l'arbitre associe au nom de compte du joueur. Pour se faire, on envoi sur la socket donnée en paramètre le nom du compte, et on modifie le pointeur donné en paramètre qui contiendra, si aucune erreur n'est retournée, l'identifiant du joueur pour la partie.
Cette fonction retournera un code de type **RetourFonction** permettant de savoir si la demande a été correctement effectuée.

2.2.3.Demande de nouvelle partie

Le processus de demande de nouvelle partie nécessite, pour respecter le protocole fourni, d'avoir l'identifiant qui est récupéré après avoir effectué une identification. Cette demande est effectuée par la fonction **demandeNouvellePartie**.
A la suite de cette demande, si le joueur est autorisé à continuer, il joue soit en premier, soit attend son tour en fonction du retour de l'arbitre. S'il n'est pas autorisé à jouer, le joueur lance le processus de déconnexion.
Cette fonction retournera un code de type **RetourFonction** permettant de savoir si la demande a été correctement effectuée.

2.2.4.Déroulement d'une partie

Le déroulement d'une partie s'effectue en fonction de la demande de nouvelle partie qui est expliquée ci-dessus.
Si le joueur est autorisé à jouer en premier, alors se lance la fonction **debutePartie**.
Cette fonction lance le moteur Java et permet successivement de jouer un coup avec la fonction **jouerUnCoup** et de recevoir un coup avec **recevoirUnCoup**. Et ce, tant que le code de retour de type **RetourFonction** de ces fonctions reste correct.

Si le joueur n'est pas autorisé à jouer en premier, alors se lance la fonction **attendPremierCoup**.
Cette fonction lance le moteur Java et contrairement à **debutePartie**, elle reçoit un coup puis joue un coup successivement.

Un entier est passé en référence des fonctions **jouerUnCoup** et **recevoirUnCoup**. Cet entier permet de compter le nombre de coup sans prises qui est limité à 50. Si jamais ce chiffre est atteint, alors la fonction concernée retourne le code d'erreur associé.

Les fonctions **debutePartie** et **attendPremierCoup** retourne le code de type **RetourFonction** associé au déroulement d'une partie. Ce code est remonté ensuite pour être vérifié.

2.2.5. La gestion des erreurs

Afin de pouvoir avoir un suivi des erreurs pendant le débogage et même durant une exécution, un protocole de gestion des erreurs a été mis en place en plus de celui disponible dans le fichier *protocolarbitre.h*.

Cette gestion se fait avec une énumération **RetourFonction** qui permet de fixer différents types d'erreurs qui peuvent apparaître durant le fonctionnement du processus.
Ainsi, il s'est dégagé une trentaine d'erreurs potentielles durant le développement.

Mais cette gestion seule ne serait pas réellement utile si on ne sait pas quel numéro correspond à quelle erreur. De ce fait, une fonction permettant de suivre le code d'erreur a été mise en place.
Cette fonction **traitementSiErreur**, qui prend en paramètre un code de type **RetourFonction**, affiche sur la sortie d'erreur un message correspondant au code passé en paramètre de fonction.

3. Utilisation du protocole de communication avec l'arbitre

Cette partie explique le déroulement de la fonction **main** du code C.
Cette fonction prend en paramètre au minimum 2 éléments : l'adresse de l'arbitre sur le réseau et le numéro de port sur lequel se connecter.
Elle peut en prendre 2 de plus qui sont un port pour le moteur Java et une adresse sur le réseau éventuellement. Par défaut, ces valeurs sont respectivement mises à *2202* et *localhost*.

La demande de connexion au moteur Java est ensuite effectuée et est répétée dans le cas où elle échoue.
La procédure est la même pour l'identification sur le moteur Java.
La fonction de détection des erreurs **traitementSiErreur** est lancée après chaque exécution de fonction.
De cette manière si une erreur est détectée, on en sera informé, dans l'autre cas, rien n'est affiché.

La procédure est reprise pour la connexion et l'identification à l'arbitre.

Concernant la demande de nouvelle partie, on effectue juste un contrôle sur le retour de type *FIN_DE_JEU*. La demande est effectuée jusqu'à ce que la fonction retourne le type de fin de partie.
C'est donc l'arbitre qui met fin au processus C de cette manière.
Le moteur Java est relancé tant que la fin de partie n'est pas détectée.

4. Présentation du moteur Java

Le moteur Java de notre programme se trouve, comme décrit précédemment, dans le dossier *src/moteurJava* et les fichiers .class dans le dossier *bin/moteurJava*. 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 modèle (plateau, pion des joueurs, dernier déplacement, …). La classe qui permet de lancer le moteur Java se nomme *Lanceur*.

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 *ConnexionJoueur*. Cette doit donc faire des demandes de coups à l'IA (programmé en Prolog), via la classe *ConnexionProlog*. Cette dernière possède une méthode *demandeCoups* qui permet de retourné un coup choisi par l'IA. La classe principale possède également une référence vers le modèle du programme.

Le modèle du programme est représenté par la class *Modele* et permet de gérer le plateau avec tous les coups, qu'il soit adverse ou de notre part. Ce est donc composé d'une liste de pions que possèdent le joueur 1 et une liste de pions que possède le joueur 2. Le plateau peut donc être représenté par ces deux listes de pions. Dans notre projet les pions sont représentés par la classe *Position* qui est composée d'une coordonnée en x et une autre en y. Le modèle possède également les autres informations indispensables à la bonne gestion d'une partie : nombre de pions restants dans la main de chaque joueur, dernier déplacement de chaque joueur, … Il possède également de nombreuses méthodes nécessaires au joueur comme à l'IA :

- Vérifier si notre joueur a gagné la partie
- Redémarrer une partie
- Vérifier la validité d'un coup
- Afficher le plateau pour savoir où on en est
- Jouer un coup (déplacer, prendre, poser)

Le modèle peut à tout moment afficher le plateau actuel en utilisant les 2 listes de pions des 2 joueurs. Exemple : le modèle affichera le plateau ci-dessous avec les configurations suivantes :

- **pionsJoueur1**: [[0,3],[1,1],[1,3],[1,4],[2,3],[2,4]]
- **pionsJoueur2**: [[1,0],[1,2],[3,1],[3,3],[4,3]]

Coordonnées en Y

Coordonnées en X	0	1	2	3	4	5
0				1		
1	2	1	2	1	1	
2				1	1	
3		2		2		
4				2		

Les coups jouées par l'adversaire ou ceux jouées par nous-mêmes, sont représentés par la classe Coups. Cette dernière possède donc comme pour les coups envoyés par l'arbitre, les 4 informations suivantes :

- Le type de coups (représenté par des entiers allant de 0 à 4)
- Les coordonnées de la case de départ
- Les coordonnées de la case d'arrivée
- Les coordonnées du 2ème pion joué

5. Mise en place d'un protocole entre le moteur Java et le joueur

Afin de transmettre les données entre le joueur et le moteur Java, nous avons instauré un protocole entre ces derniers. Comme il est impossible d'envoyer les structures C, reçues de l'arbitre au moteur Java, nous avons décidé que le joueur et le moteur s'enverraient l'un après l'autre une seule chaîne de caractère car il est très simple d'envoyer ou de recevoir des chaînes de caractères en Java et également en C. La communication entre les 2 se fait, pour le moteur Java, dans la classe *ConnexionJoueur* et pour le joueur dans le fichier C : *connexionArbitre*. Le protocole que nous avons instauré permet d'envoyer 3 types de requête que nous allons étudier par la suite :

- Identification
- Nouvelle partie
- Jouer un coup

5.1. Identification

Au départ, le joueur doit se connecter au serveur du moteur Java et lui envoyer une requête d'identification. Pour cela, le joueur lui envoi une chaîne de caractère contenant un mot de passe présent dans le code C ainsi que dans le code Java afin d'être sûr que le joueur qui se connecte est bien le nôtre. Le moteur Java lui renvoie ensuite la chaîne « **OK** » si le mot de passe est correct. Le joueur peut alors demander une nouvelle partie.

5.2. Nouvelle Partie

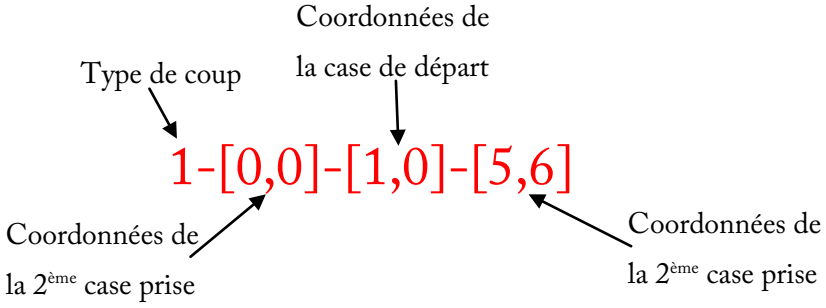
Lorsque le joueur souhaite démarrer une partie, il doit prévenir le moteur s'il commence la partie (le moteur devra alors demander une partie à l'IA et l'envoyer au joueur) ou si c'est au tour de l'adversaire de commencer (le moteur devra alors attendre un coup). Pour cela, le joueur envoie au moteur la chaîne « **commence** » si il démarre la partie ou « **attend** » s'il doit attendre un coup de l'adversaire.

La partie se déroule ensuite indéfiniment par une suite d'envoie de coup et de réception de coup (voir chapitre suivant).

Lorsqu'une partie est terminée, si le joueur souhaite redémarrer une partie, il peut à tout moment envoyer une requête pour réinitialiser le plateau du moteur Java. Il envoie alors la chaîne « **restart** » au moteur, et devra, à nouveau, envoyer s'il commence la partie ou non.

5.3. Jouer un coup

L'envoie des coups, joués par nous-mêmes ou par l'adversaire, entre le joueur et le moteur Java sont codés dans une seule chaîne de caractères pour faciliter la communication entre C et Java. La requête est composée de cette manière (exemple) :



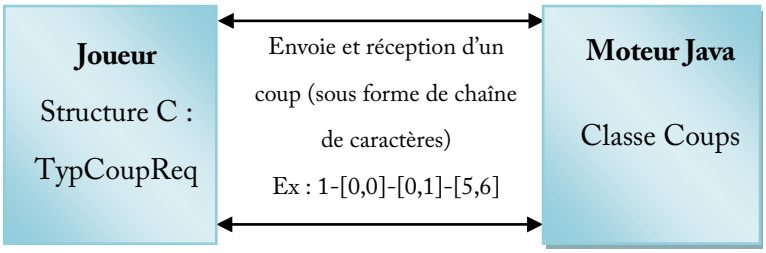
La requête contient bien les informations indispensables : type de coup, case de départ, d'arrivée et 2ème case prise. Ceux-ci sont séparés par un tiret afin de bien les délimiter.

Les coordonnées des cases correspondent aux coordonnées en x et en y de cette dernière selon le plateau données dans le paragraphe *4 : Présentation du moteur Java.* Ces coordonnées ne peuvent pas dépasser la taille maximale du plateau (vérification dans le moteur Java : méthode valider()). Les coordonnées de la main sont représentées par [5,6]. Lorsqu'une case n'a aucune importance, exemple : lors d'un déplacement la 2ème case prise, on envoie au joueur les coordonnées de la main ([5,6]).

Le type de coup est représenté par un entier entre 0 et 4 (défini dans la classe *Modele*) et représente :

- 0 → POSE
- 1 → DEPLACE
- 2 → PRISE
- 3 → NULLE
- 4 → GAGNE

Schéma qui explique les structures de données utilisées pour représenter les coups :



Le joueur a besoin de convertir les coups reçus et envoyer en chaîne de caractères afin de les envoyer au joueur. Pour cela, nous utilisons deux fonctions :

- typCoupReqToString : convertit le coup sous forme de structure reçu de l'arbitre (typCoupReq) par une chaîne de caractères qui sera envoyée au joueur.
- stringToTypCoupReq : fonction qui fait l'inverse de la fonction précédente

Le moteur a également besoin de convertir les coups reçus et envoyer en chaîne de caractères afin de les envoyer au joueur. Pour cela, la classe *Coups* possède une méthode *getReq()* qui convertit un coup en une chaîne de caractères. Pour convertir, de l'autre sens, il suffit de donner en paramètre au constructeur de *Coups*, la requête reçue du joueur.

A chaque fois que le moteur Java reçoit ou envoi un coup, il vérifie la validité de ce dernier avec la méthode *valider()* de la classe *Modele*. Lorsque le moteur demande à son IA de choisir un coup, le moteur vérifie si le coup permet de terminer la partie par une victoire (méthode *aGagne()* de la classe *Modele*). Si c'est le cas, il modifie le coup par 4 (GAGNE), avant d'envoyer au joueur la requête.

Table des matières