Augustin Bordy
Nicolas Courvoisier

TPB Master 1 Informatique

Rapport SCS

1. Utilisation du protocole de communication avec le serveur

Lors du développement de notre serveur de jeu et de notre joueur, nous avons suivi scrupuleusement le cahier des charges et le protocole de communication donné dans le sujet. Nous avons des structures de données qui respectent le protocole de communication. Et, lors de chaque réponse du serveur de jeu, nous affichons un message au joueur pour lui indiquer divers informations. Nous lui affichons par exemple, si le coup joué est valide, s'il ne l'est pas, si la partie est bien initialisée, le nom de son adversaire et toutes autres informations qui peuvent être utiles aux joueurs.

2. Mise en place d'un protocole avec le moteur IA

Pour faire fonctionner notre jeu, les informations du serveur sont envoyées aux deux joueurs. Cette communication entre serveur et joueur utilise les sockets en C, les joueurs communiquent ensuite, via Sockets également, avec leurs moteurs IA qui ont été réalisés en Java.

a. Protocole réalisé

Pour commencer, nous échangeons uniquement des entiers entre notre joueur et notre moteur IA. Les différentes données qui sont échangées entre le Joueur et son moteur

IA le sont sous forme d'entiers, des fonctions au niveau de l'envoie et de la réception des deux côtés se chargent d'effectuer les conversions nécessaires pour respecter le protocole d'échange avec le serveur. Le protocole demande d'avoir un type TTypePion pour les pions qui est représenté par une énumération (cylindre, pavé, sphère, tétraèdre). Le cylindre représente l'entier 0, le pavé : 1, la sphère : 2 et le tétraèdre : 3. Nous pouvons donc convertir les TTypePion en entiers et vice-versa. Nous avons le même type d'opération pour savoir sur quelle case le coup est joué. De manière interne au moteur d'IA, cette transformation n'est pas nécessaire, nous conservons donc des entiers pour remplir la grille de jeu de l'IA.

Aussi, nous avons réalisé plusieurs envois successifs pour transmettre les informations au moteur IA. Nous avons dû suivre une certaine logique pour que ces informations arrivent aux endroits souhaitées. Nous avons donc décidé de toujours envoyer et recevoir les informations dans le même ordre. Nous commençons par la ligne, suivi de la colonne, ensuite du pion et du type de coup.

Par ailleurs, nous avons deux cas de jeu, soit le joueur joue en premier, soit en second. Cela à une importance car les informations reçues et envoyées ne se font pas dans le même ordre.

- Si le joueur joue en premier :
    - il commence par dire à l'IA qu'il joue en premier et il rentre dans la boucle de jeu
    - il reçoit via son moteur IA la case, le pion et le coup à jouer
    - Si le coup est valide, il est joué, sinon il envoie à son IA que le coup est invalide et la boucle de jeu s'arrête.
    - il reçoit le coup du joueur adverse, s'il est valide il envoie à l'IA que le coup est valide et qu'il peut continuer à jouer. Sinon, il envoie que le coup est invalide et la boucle s'arrête.
    - ensuite il envoie le coup du joueur.

- Si le joueur joue en second :
    - il commence par dire à l'IA qu'il joue en second et il rentre dans la boucle de jeu
    - il reçoit le coup du joueur adverse, s'il est valide il envoie à l'IA que le coup est valide et qu'il peut continuer à jouer. Sinon, il envoie que le coup est invalide et la boucle s'arrête.
    - ensuite il envoie le coup du joueur.
    - il reçoit via son moteur IA la case, le pion et le coup à jouer
    - Si le coup est valide, il est joué, sinon il envoie à son IA que le coup est invalide et la boucle de jeu s'arrête.

Ces deux scénarios sont joués en boucle, jusqu'à que la partie s'arrête. Dès qu'il y a eu un coup invalide, un problème ou qu'un joueur a gagné, cette boucle s'arrête.

De plus, l'IA possède deux boucles de jeu, une pour la première partie et une seconde pour la revanche. Avant le début des boucles, l'IA reçoit du joueur l'information s'il joue en premier ou non, et, à partir de là, la boucle démarre et en fonction de cette réception

l'IA attend le coup du joueur adverse ou réfléchit au coup à jouer. Quand la première partie est terminée, la boucle s'arrête. Et grâce à la précédente partie, il sait que le joueur blanc va jouer en second au lieu de jouer en premier. Et, le joueur noir va jouer en premier au lieu de jouer en second. La fin de la revanche est détectée de la même manière que la première partie.

b. Problèmes rencontrés et solutions apportées

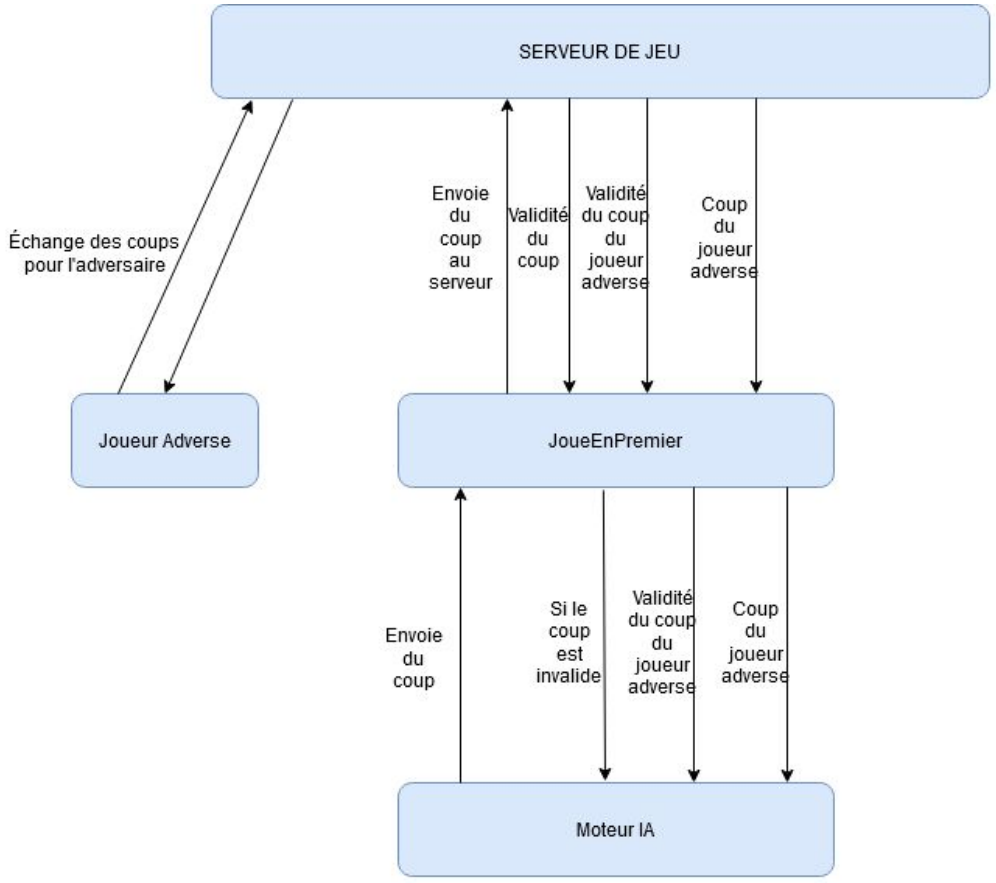
Pour cela, nous avons fait le choix d'utiliser les sockets entre le Joueur en C et son moteur IA en Java. Nous avons utilisé cette technologie car elle est réputée fiable et son déploiement était connu pour nous. Lors de nos échanges Java/C, nous échangeons uniquement des entiers. Quand la communication est dans le sens C vers Java, nous envoyons le coup de l'adversaire (sa position dans la grille et le pion joué). Et, dans le sens Java vers C, nous envoyons la position et le pion que notre IA a calculé. Dans la communication entre Java vers C (et aussi C vers Java), nous ne pouvons pas échanger des structures. La solution de ce problème a donc été de faire plusieurs envois successifs pour envoyer les coordonnées des coups joués, le pion joué et si le coup est terminal.

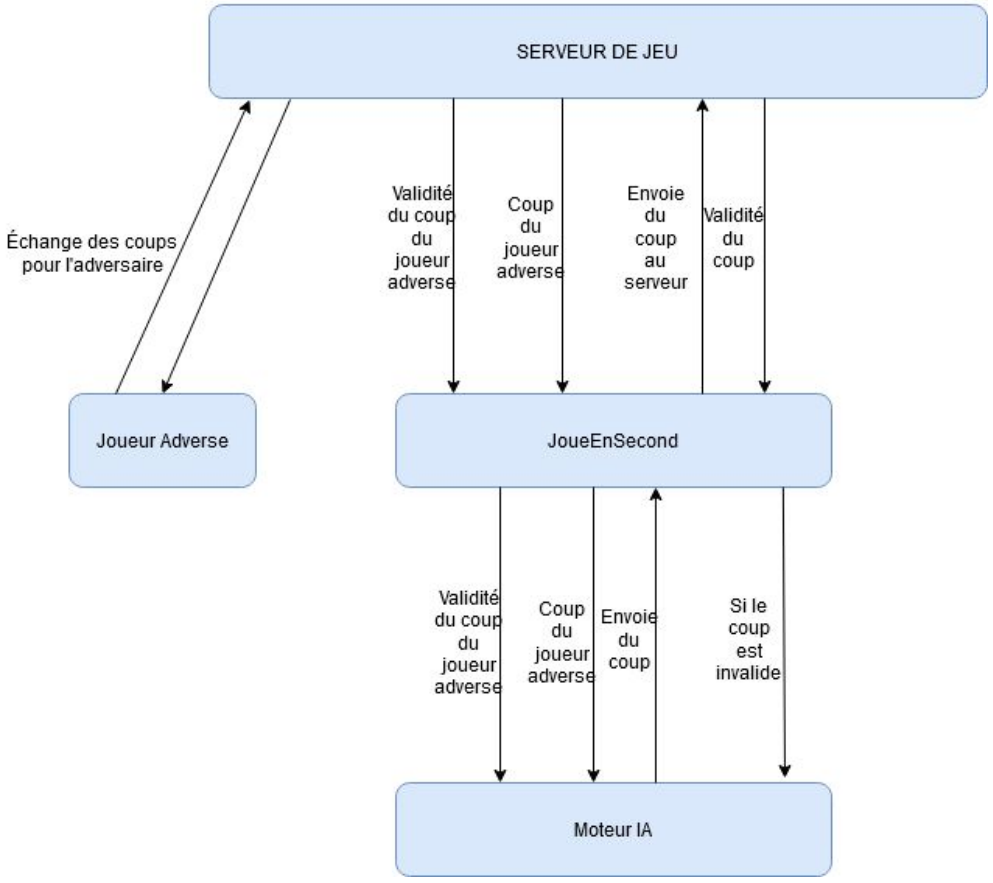
Il y a avait aussi un autre problème pour mettre en place une communication Java vers C (Mais pas dans le sens C vers Java) avec les sockets, il faut faire attention à la bonne réception des message du côté du langage C. En effet, Java étant un langage de plus haut niveau que le langage C, Java s'exécute donc plus lentement. L'envoi de donnée de Java vers C possède donc un problème. Celui-ci est que le langage C commence à recevoir des données qui sont envoyés par le langage Java, alors qu'il n'a pas fini de les envoyer. Pour cela, nous réalisons une boucle de quatre tours, avec l'option MSG_PEEK sur la fonction qui réceptionne les données côté C et qui permet de dire que la réception n'est pas terminé. Et, une autre réception après la boucle, pour pouvoir récupérer la donnée, la reconstruire et terminer la réception. Nous réalisons une boucle de quatre tours, car nous envoyons uniquement des entiers entres ces deux langages et les entiers sont codés sur quatres bits dans ces langages. Et, MSG_PEEK permet d'attendre les données qui sont traités comme non lues et la données sera renvoyé dans la fonction de réception après la boucle.

Aussi, lors de l'envoi et de la réception des données dans le langage C, nous devons les transformer au format du réseau. C'est pourquoi, nous utilisons la fonction ntohl qui convertit les octets réseau (en conservant l'ordre) en octets hôte. La fonction htonl, fait le contraire. Elle convertit les octets hôte en octets réseau.

3. Explications

a. Schémas des échanges


SERVEUR DE JEU
Échange des coups pour l'adversaire
Envoie du coup au serveur
Validité du coup
Validité du coup du joueur adverse
Coup du joueur adverse
Joueur Adverse
JoueEnPremier
Envoie du coup
Si le coup est invalide
Validité du coup du joueur adverse
Coup du joueur adverse
Moteur IA


SERVEUR DE JEU
Échange des coups pour l'adversaire
Validité du coup du joueur adverse
Coup du joueur adverse
Envoie du coup au serveur
Validité du coup
Joueur Adverse
JoueEnSecond
Validité du coup du joueur adverse
Coup du joueur adverse
Envoie du coup
Si le coup est invalide
Moteur IA

b. Structure logicielle joueur

La structure logicielle du joueur respecte le protocole donné avec le sujet. Le joueur va recevoir un certain nombre d'information en ligne de commande. Notamment, l'adresse IP et le port du serveur distant, la couleur avec laquelle le joueur souhaite jouer, le nom du joueur et le numéro de port pour le moteur IA. L'IP, le port serveur et le nom du joueur seront modifiable facilement si le joueur est lancé avec le script sh, ces informations sont définis en paramètre de lancement du script. La couleur et l'IP du moteur IA ne sont pas modifiables, à part en changeant les valeurs dans le script sh. Mais, le port du moteur IA est par défaut écrit dans le fichier et peut être modifié en option via les paramètres du script.

Le joueur commence par vérifier les informations reçues en paramètre du programme. Si, elles sont valide, on crée une requête de jeu pour initialiser une nouvelle partie. Sinon, on retourne un message d'erreur. Cette requête est envoyée au serveur. Ensuite, nous pouvons initialiser la communication avec notre moteur IA. Avec un numéro de port défini précédemment, nous réalisons la connexion avec l'intelligence artificielle. Le serveur attend un second joueur. Une fois qu'il est arrivé, le serveur initialise la partie et retourne une réponse au client. Cette réponse indique s'il y a une erreur dans la demande de partie et sinon un message indique le nom de l'adversaire. Si les deux joueurs ont choisis la même couleur, il est possible que celle-ci soit changé par le serveur. Un message indique aux joueurs la couleur qu'ils vont avoir durant la totalité de la partie.

Nous commençons par envoyer à l'IA si nous commençons ou non à jouer. Et ensuite la boucle de jeu se met en place. Si le joueur joue en premier :

- Il est en attente de la réception du coup de l'IA
- Il envoie le coup au serveur
- Le serveur retourne un message pour déterminer si le coup est valide. S'il y a une erreur, on envoie à l'IA d'arrêter la première manche et nous terminons la boucle de jeu de la première manche aussi côté joueur
- Sinon, on reçoit si le coup de l'adversaire est valide, si oui on continue, sinon on arrête de jouer
- On envoie à l'IA que l'on continue à jouer
- On reçoit le coup de l'adversaire
- On envoie le coup de l'adversaire à l'IA

Si le joueur joue en second :

- On reçoit si le coup de l'adversaire est valide, si oui on continue, sinon on arrête de jouer
- On envoie à l'IA que l'on continue à jouer
- On reçoit le coup de l'adversaire
- On envoie le coup de l'adversaire à l'IA
- Il est en attente de la réception du coup de l'IA
- Il envoie le coup au serveur

- Le serveur retourne un message pour déterminer si le coup est valide. S'il y a une erreur, on envoie à l'IA d'arrêter la première manche et nous terminons la boucle de jeu de la première manche aussi côté joueur

Ces deux fonctions sont réalisés jusqu'à qu'il y ait un problème ou jusqu'à que la partie se termine. Une fois la première manche terminé, le moteur IA prépare la seconde manche en disant que c'est les noirs qui commencent. Le joueur initialise la revanche aussi. Et, la même boucle de jeu se met en place et la fin de partie est détecté de la même manière.

En section 2, nous avons dit que nous échangions uniquement des entiers entre l'IA et notre joueur. Or, les types des coordonnés et des pions sont des énumérations et ne sont pas du type entier. Nous avons réalisés quelques fonctions de conversions pour pouvoir réaliser les échanges entre les deux programmes.

Une fois la partie terminé, nous indiquons au joueur s'il a gagné ou perdu. Et, à la fin des deux manches, on écrit un message de résumé des scores des deux parties et nous fermons proprement tous les communications.

c. Structure logicielle serveur

Une des problématiques principales du développement du serveur était de mettre en place une écoute sur les ports des deux joueurs connectés, et d'ensuite accepter ou non les requêtes en fonction de si elles venaient de joueurs dont on attendait un coup. Pour cela différentes stratégies étaient envisageables, l'une consistant à créer un Thread différencié pour la communication avec chaque joueur, simple à mettre en oeuvre, elle n'a cependant pas été retenue pour ne pas à avoir à gérer de données communes, pour ne garder qu'un seul environnement. S'offraient alors à nous les choix des sockets non bloquantes et du multiplexage, nous somme parti sur ce dernier car son fonctionnement souple basé sur un set de sockets écoutés nous permettait de simplement y ajouter la socket d'un joueur lorsqu'il se connectait.

Réception et traitement des requêtes :

Une foi le multiplexage mis en place dans une boucle infinie, la mécanique de traitement des requêtes entrantes est la suivante :

- Si de l'action a lieu sur la socket de connexion, la fonction *handlePlayerConnection* est appelée, celle-ci, à condition que deux joueurs ne se soient pas déjà connectés, va instancier une nouvelle structure *PlayerData* contenant les informations relatives à un joueur, notamment la valeur de sa socket de communication.
- Si de l'action a lieu sur l'un des sockets de communication des deux joueurs, la fonction *handlePlayerAction* est appelée avec en paramètre la structure *PlayerData* du joueur auquel appartient la socket. Le protocole autorise le joueur à soumettre deux types de requêtes: les *TPartieReq* et les *TCoupReq*, la fonction reçoit alors uniquement l'information de l'identifiant de la requête en utilisant l'option *MSG_PEEK* sur le *recv* pour déterminer sa nature. En fonction de celle-ci, le traitement est redirigé vers la fonction *handleGameRequest* s'il s'agit d'une demande de partie ou

vers *handlePlayingRequest* s'il s'agit d'un coup. Une vérification préalable de la cohérence du type de la requête avec l'état du jeu est également effectuée: Si la partie a commencée et qu'une requête de type *TPartieReq* est reçue, une requête *TCoupRep* avec code d'erreur égal à *ERR_TYP* est retournée, si la partie n'a pas commencée et qu'une requête *TCoupReq* est reçue, une requête *TPartieReq* avec code d'erreur égal à *ERR_TYP* est retournée.

- Si la fonction *handlePlayerAction* a déterminée que la requête est une demande de partie, *handleGameRequest* va prendre le relai, le nom du joueur y est enregistré et sa couleur est validée ou non, si le joueur est le deuxième à envoyer cette requête, la partie peut commencer, des requêtes *TPartieRep* sont envoyées aux deux joueurs pour confirmer le lancement ainsi que leurs couleurs.
- si la fonction *handlePlayerAction* a déterminée que le requête est un coup, *handlePlayingRequest* va prendre le relai. Si la validation des coups n'a pas été désactivée par l'argument *--noValid* au lancement du serveur, l'arbitre est interrogé pour connaître la validité du coup, ceci grâce à une structure *TPropCoup* qu'il retourne. Si la validation est désactivée, on utilise la structure donnée par le joueur.

Déroulement du jeu :

Quand le jeu a commencé, les joueur reçoivent tous une requête de type *TPartieRep,* à partir de ce moment, un pointeur sur une structure *PlayerData* passé en variable globale va désigner le joueur dont le coup est attendu, sa référence est donc mise à jour à la fin de chaque tour.

Après que le coup d'un joueur ai été traité, si il est valide, la fonction *endAction* est appelée, celle-ci envoie tout d'abord le résultat de la validation du coup aux deux joueurs, comme spécifié par le cahier des charges, puis, si le résultat indique que la partie doit continuer, envoie également la requête *TCoupReq* qui vient d'être jouée à l'adversaire. Si le résultat indique en revanche que le joueur gagne/perd ou si il y a match nul, la fonction *endRound* est appelée.

La fonction *endRound* désigne le gagnant, si il y en a un, par un affichage et lance le round suivant si le round actuel était le numéro 1, appel la fonction *endGame* sinon. Il se peut également que le jeu ai été arrêté pour cause d'un joueur se déconnectant, dans ce cas *endGame* est appelée même au round 1.

La fonction *endGame* ferme les différents sockets, de communications et de connexion et passe la variable booléenne 'gameContinue' à False, cette variable est la condition de continuation de la boucle infinie de la fonction principale.

Gestion du timeout :

Le timeout est rendu possible par un timer initialisé dans un processus fils au lancement de la partie. Au bout de 5 secondes, s'il n'est pas réinitialisé, il envoie un signal *SIGUSR1* à son père (le processus principal) qui l'intercepte et annonce le joueur courant comme perdant de la manche.

Si le joueur a joué dans les temps, un signal *SIGUSR2* est envoyé par le processus principal au processus du timer, lequel l'intercepte et remet le décompte à 0.

Une autre approche possible pour mettre en place le timeout était de donner une *struct timespec* en paramètre au select, lui spécifiant donc le délai maximal pendant lequel

bloquer. Le problème était qu'une interaction du joueur n'étant pas censé jouer sur son socket aurait également débloqué le select et donc remis le timer à 0.

Le timer, par défaut activé, peut être désactivé en spécifiant l'option '--noTimeout' au lancement du serveur.

Particularités :

Nous avons pour un usage purement pratique et n'impactant pas le fonctionnement du serveur, rajouté deux options de lancement à celle demandées :

- Une option '--showBoard' pour afficher le plateau de jeu, affichage désactivé par défaut.
- Une option '--debug' pour afficher davantage d'informations sur l'état du serveur au cours de son exécution, cette option nous a été utile lors de la phase de développement et de tests.

d. Répartition des tâches

Les deux membres du binômes ont travaillé sur la partie réseau. Au début du projet, nous voulions commencé par travailler tous les deux sur le joueur. Mais, nous nous sommes vite aperçu que c'était pas la bonne solution. A ce moment, uniquement l'initialisation des parties était possible avec le serveur de jeu fournit. Nous avons donc découpé le travail. Augustin a majoritairement travaillé sur le serveur. Et, Nicolas a majoritairement travaillé sur le joueur. Mais, dès que nous avions des problèmes, nous n'avons pas hésité à travailler tous les deux sur les problèmes rencontrés pour les solutionner au plus vite. Nous avons aussi beaucoup échangé lorsque nous avions un doute sur la réalisation d'une fonctionnalité. A chaque avancé, l'un notifiait à l'autre là où il en était.