Rapport du Projet Tournoi d'Intelligence Artificielle
Jeu Colonnes de trois en réseau

Partie communication

BARADI Farah, MONTEIRO Thiago

Master 1 Informatique

2021-2022

Nous souhaitons dans un premier temps remercier tous nos enseignants, qui nous ont permis de progresser dans ce projet tout en répondant à nos questions.

Introduction :

Ce projet a été effectué lors du deuxième semestre de Master 1 durant les mois de mars jusqu'à mai, dans le cadre de l'UE « Projet Tournoi d'Intelligence Artificielle » unissant les matières de "Systèmes Communicants et Synchronisés" et "Méthodes et Outils pour l'Intelligence Artificielle" encadré par BOUQUET FABRICE et FELEA VIOLETA

L'objectif de ce projet consistait à implanter le jeu "colonne de trois" avec des règles de jeu correspondant aux celles expliquées dans partie d'intelligence artificiel et d'implémenter la communication entre le client et le serveur ainsi avec le serveur d'inélégance artificiel permettant de jouer en réseau

Le jouer (le client) et le serveur est implémentés en langage C et le serveur IA est développé en Java.

La communication au départ consiste de deux parties : une demande de participation envoyer par le jouer au serveur, dès que c'est accepté, on passe à la deuxième partie qui est le jeu de coup par les deux joueurs.

Chaque joueur fait appel au moteur IA pour connaitre le bon coup à jouer dès que le jeu est toujours possible selon les règles de joue et lui informe aussi de coup joué par son adversaire.

Les coups envoyés par les joueurs sont vérifiés et administrées par un processus serveur, arbitre du jeu, écrit en langage C.

A noter que les processus joueurs communiquent à travers des sockets avec le serveur.

Les joueurs et le serveur utilise un protocole de communication entre eux ce protocole est prédéfini dans protocolColonne.h

Participation :

Pour cette première partie chaque joueur envoie une demande de participation avec son nom (ou Login) et il reçoie comme réponse le couleur de ces pions (blanc ou noir) ainsi le nom de son adversaire. (Avec le code d'erreur bien évidemment)

Selon les règles de joue qui consiste qu'elle est à jouer en deux partie (aller et retour) pour le premier match le joueur blanc commence à jouer le premier coup et au deuxième match c'est le noir qui commence, en tenant compte que les joueurs ne change pas leur couleur c'est justement le tour qui change

Cette partie de communication est seulement entre le jouer et le serveur C avec une protocole spécifié entre eux. Et sans aucune communication avec le serveur IA.

-Toutes les fonctions créées dans le client et le serveur expliquées ici servent à améliorer l'organisation du code et le rendre plus lisible.

Client

On a créé la fonction doHandshake pour l'envoie et la réception de participation.

On spécifie les éléments de la requête de participation avec une instance du protocole (TPartieReq), le client saisi son nom, et on envoie la requête sur la socket de communication avec le serveur.

Le joueur reçoit la réponse sur la même socket de communication, et il la stock sur une instance du protocole (TPartieRep). Ensuite on a créé une fonction handleParticipationAck qui le prend en paramètre. Cette fonction et selon le code d'erreur reçu, elle affiche la réponse reçue du serveur comme des messages clairs au joueur indiquant son couleur et le nom de son adversaire et elle return le couleur du jouer (mais comme an Integer (blanc = 0 et noir = 1) pour les raisons de simplification du code)

Serveur

Le serveur reçoit la requête de demande de participation sur la socket de communication qui ne prend que deux clients au maximum.

Après la réception de demande des deux joueurs en envoie la réponse aux deux joueurs un par un.

Pour le serveur, on a créé la fonction doHandshake qui prend la socket de commination en paramétrés. On commence par la réception de la demande de participation, et on la stock dans une instance du protocole (TPartieReq). Et on stock le nom de chaque jouer reçu dans un tableau diffèrent. Et à l'envoie de la réponse de cette requête de participation, on inverse les deux tableaux pour envoyer à chaque jouer le nom de son adversaire, Pour la construction et l'envoie de la réponse, on utilise une instance du protocole (TPartieRep).

À noter que on vérifie que le type de requête reçu est bien une demande de participation et pas une demande de jouer un coup. Sinon on lui envoie un message indiquant l'interdiction de jouer un coup pour le moment.

Alors on a créé une socket de communication au côté serveur qui n'accepte que deux joueurs au maximum. Et donne le premier jouer connectée sur la socket le couleur blanc et le deuxième le couleur noir.

Jouer des coups :

Cette partie est consisté de deux sous-parties : un positionnement des pions et un déplacement après que tous les pions des deux joueurs sont placés sur le tableau (board)de jeu

Client

Le premier joueur consulte le moteur IA pour savoir la meilleure place sur le tableau pour placer son premier pion, et le moteur IA lui envoie la réponse. On utilise ici un protocole qui a été implémenté pour la communication entre eux (la mise en place de ce protocole est expliquée dans la deuxième partie de ce rapport)

-Dès que le jouer reçois la réponse du moteur IA, il l'envoie au serveur C en utilisant le protocole établis entre eux et une utilisant l'instance (TCoupReq),

À l'envoie de coup, il faut fournir le couleur des pions pour ce joueur ainsi le type de coup (positionnement, déplacement ou même une passe) avec des coordonnes de ligne et de colonne de la place de ce pion et évidemment le type de requête qui est ici un COUP

Après avoir reçue la réponse de son coup du serveur C (y compris la réponse de validation de son coup et du coup joué par son adversaire), et si et seulement si le coup de l'adversaire est valide (alors le match est toujours en cours), il va recevoir de serveur C les informations concernant le coup de son adversaire, Et alors le jouer de sa part il informe le moteur IA des détails de coup joué par son adversaire

On crée la fonction makeMove Pour l'envoie de coup au serveur C et la réception de la réponse de son coup à lui.

-Pour le traitement de la réponse de coup, on a créé trois fonctions.

HandleOwnPlayValidation est la fonction pour la l'affichage de la réponse de coup de joueur courant et aussi pour arrêter le match s'il le faut selon les règles de joue, sinon on retourne que le match est toujours en cours. Et alors on peut recevoir la réponse de serveur C concernant le coup de son adversaire par la fonction handleOponentPlayValidation, qui fait la exactement la même chose que la fonction précédant en ce qui concerne l'arrête et la continuation du match courant.

Alors, si le match est toujours en cours, on reçoit les informations concernant le coup de son adversaire et on utilise la fonction handleOponentPlayInformation pour les envoyer au serveur IA.

À noter que les deux fonctions HandleOwnPlayValidation et handleOponentPlayValidation prennent en paramétré une instance du protocole de répondre au coup requête (TCoupRep).Et la fonction handleOponentPlayInformation prend (TCoupReq)en paramètre car on reçoit les informations de coup qui a été envoyer comme une requête de demande de coup de cote adversaire.

Serveur

Le serveur commence par l'initialisation de partie. Ensuite et selon le tour de joueur, il reçoit la requête de coup et il la traite.

Il vérifie si le coup reçu par le joueur et valide en utilisation la fonction donnée validationCoup, le serveur agi selon la réponse de cette fonction.

Si la réponse indique que le coup est valide, il return la propriété de coup (selon l'état actuel de jeu par cette coup) soit CONT qui n'indique qu'aucun n'a gagné ou GAGNE si ce joueur a gagné le match actuel.

Sinon, si la réception de coup du jouer qui a le tour, a pris plus de six second. La fonction va retourner que le coup n'est pas valide (c'est un TIMEOUT) et NULLE.

Un dernier cas : s'il y une triche, par exemple une demande de positionnement après un déplacement, la fonction retourne (TRICHE) et PERDU comme propriété de coup

On utilise la fonction validateAndBuildPlayResponse qui prend en paramètre la réponse de la fonction de validation et la valeur qui indique à qui le tour et la même instance liée au protocole concernant les propriétés du coup (TCoupRep). Dans tous les cas où le coup n'est pas valide on arrête le matche courant et on envoie l'état du match (continue ou arrêté)

Ensuite, et dans un premier temp, on envoie la réponse de validation quel que soit les résultats (valide ou non) aux deux joueurs (celui qui a envoyé ce coup et à son adversaire). Avec la même instance de protocole (TCoupRep) utilisée dans la validation du coup.

Puis, dans la condition que le coup est valide on informe l'adversaire des détails de coup jour, c'est à dire toutes les informations envoyées par le jouer qui a demandé ce coup (le couleur des pions, le type de coup, la ligne et la colonne, ...etc.), Donc on utilise la même instance de protocole (TCoupReq)

Ce processus est fait pour chacun des deux joueurs par son tour (en alternant entre les deux) jusqu'à la fin du match, et le même cas pour le deuxième match.

La mise en place d'un protocole avec le moteur IA :

On crée le protocole protocolJava

Qui a une structure de réponse de serveur IA contenant trois éléments, le type de coup TCoup (positionnement, déplacement ou passe), et les cases TCase et le TDeplPion du protocole protocolColonne utilisé pour la communication entre le joueur et le serveur C.

Au départ le client se connecte au serveur IA et puis il l'envoie son couleur par startAI, et une demande de jouer un coup par requestAI où il envoie son couleur et il reçoit toutes les autres informations du coup, ici on fait la référence de la réponse reçue concernant son coup à leurs protocole (protocolJava).

Ensuite on construit le coup en faisant l'équivalence entre les deux différence protocoles par la fonction buildPlayRequest

Et envoie le coup reçu du serveur IA au serveur C pour la valider.

Ensuite quand on reçoit les informations du coup de l'adverse, en informe le serveur IA, par la fonction sendOpponentMoveToAI. Et on prend en paramètre (TCoupReq) parce que ces informations sont en réalité une requête de coup envoyer par son adversaire. On a envoyé chaque information en plusieurs (send) fonctions.

À la fin du match, on envoie une demande au serveur IA de se préparer pour le deuxième match par la fonction setNextStateAI.