Rapport de projet

Colonne de trois / Partie SCS

Lucas Mary et Sammy Loudiyi

12/05/2017

Ce document présentera la réflexion et la mise en forme du projet de Système Communicants et Synchronisés en commun avec le module de Méthodes et Outils pour l'Intelligence Artificielle

Sommaire

Introduction

Ce sujet a été réalisé dans le cadre de la première année du Master Informatique à l'UFR-ST de l'Université de Franche-Comté.

Nous tenons à remercier tout particulièrement Guyon Loan pour son aide nous ayant permis de comprendre et de réaliser au plus possible ce projet.

S'en suit un résumé du sujet : Nous devions réaliser deux processus, un de type « Joueur » ou « Client » et l'autre de type « Serveur », respectivement l'un pour jouer et envoyer des données, l'autre pour les traiter, les recevoir et donner des réponses aux « Joueurs ». Ces processus seront codés en C. Un système d'intelligence artificielle a été mis en place grâce au code réalisé en Prolog. Pour finir, il nous a été demandé de réalisé le lien entre le client en C et l'intelligence artificielle en Prolog à l'aide d'un système « Jasper » basé sur le langage Java.

Ainsi, le projet s'est constitué des parties suivantes :

- Client
- Serveur
- Intelligence artificielle
- Logiciel communicant

Dans ce rapport, nous verrons de ce fait le développement de ce projet, en partant des idées de bases puis pour arriver aux améliorations possibles avec une critique du travail réalisé.

I) Client

Comme il l'a été précisé dans l'introduction, nous avons du réalisé un client dans le cadre de ce projet. Ce dernier va donc vous être décrit dans la partie qui suit.

1) Connexion avec le serveur

Le client se lance à l'aide d'une ligne de commande avec pour prototype la suivante :

./client [adresse ip] [numero port] [nom machine]

A l'aide de cette commande, nous avons défini dans le code une fonction de connexion permettant de réaliser la communication entre le client et le serveur, dont nous parlerons plus tard.
Par cette action, on va chercher à initialiser la partie. Une structure TPartieReq sera alors initialisée avec la valeur « PARTIE », spécifiant que le joueur est en attente et désire jouer.
Si la requête est acceptée par le serveur, le client est alors connecté au serveur de jeu ainsi qu'à l'intelligence artificielle développée en Java. Il est alors mis en attente jusqu'à avoir un autre adversaire se connectant au moins sur le serveur de jeu.
Le serveur doit alors renvoyer au client une requête « TPartieRep » lui permettant de recevoir ses données de jeu, telles que la couleur de ses pions et le nom de son opposant si l'opposant en a choisi un.

2) Clients connectés au serveur

Lorsque les deux clients sont connectés au serveur comme expliqué précédemment et que l'intelligence artificielle des deux joueurs est correctement connectée elle aussi, la partie peut démarrer.

Les joueurs se mettent alors chacun dans une situation possible :

- Le joueur possédant la main de départ va envoyer à son adversaire une requête TCoupRep avec les informations concernant le coup que l'intelligence artificielle à décider de jouer.
- Le joueur ne possédant pas la main de départ va se mettre en attente de la requête TCoupRep et va ainsi recevoir le coup décidé par l'adversaire.

S'en suit une validation par le serveur à chaque coup envoyé, permettant de mettre à jour le plateau de jeu du côté des intelligences artificielles.

Cette validation retournera pour l'attribut validCoup différentes valeurs. Si celle-ci est de type « VALID » et que le coup joué « propCoup » est de type « CONT », le client va attendre un deuxième appel du serveur qui contiendra ce coup-ci les informations envoyées par l'adversaire, décrivant son coup.

La tendance s'inverse à chaque coup, le joueur ayant eu la main la passe à l'autre joueur pour qu'il puisse jouer ainsi de suite jusqu'à ce qu'il y ait victoire d'un joueur. Cette victoire peut être affirmée par le serveur grâce aux propriétés suivantes :

- TValCoup peut renvoyer « TIMEOUT » ou « TRICHE », entraînant la disqualification immédiate de l'utilisateur dont le client n'a soit pas été assez rapide, soit a effectué un coup « illégal » (Poser un pion sur une case déjà occupée ou une case inexistante par exemple.
- TPropCoup peut renvoyer « GAGNE » dans le cas où la machine de l'utilisateur arrive à empiler 3 pions de sa couleur.
  Il peut renvoyer « NULLE » dans le cas où chaque machine, l'utilisateur et son adversaire, a effectué 20 tours.
  Il peut enfin renvoyer « PERDU » dans le cas où il s'agit de l'adversaire de l'utilisateur qui réussisse à empiler les 3 pions de sa couleur.

3) Fin de partie

Une fois que le serveur a envoyé et fait comprendre aux joueurs que la partie était terminée, une fonction « exitAll() » permettra de fermer toutes les connexions et de libérer proprement les ports de connexion.

4) Remarques personnelles

Il s'agira ici d'une partie de projet ayant posé des difficultés assez conséquentes. La première difficulté venait d'un manque de connaissances ayant ralenti considérablement le développement du projet et bloquant ainsi l'optimisation de ce dernier.
La seconde difficulté portait sur le code donné. En obtenant les prototypes des fonctions et les différentes structures, nous obtenions ainsi des informations sur la manière dont nous devions les utiliser. Cependant, cela nous a aussi limités dans les possibilités de développement et les idées que nous pouvions avoir, étant restreints à plusieurs structures.

Le développement de ce projet a cependant été très intéressant. La compréhension de certaines fonctionnalités telle que l'application d'un timeout, la communication entre différents langages et le côté ludique de ce projet nous a permis d'approfondir nos connaissances en systèmes communicants.

II) Serveur

Le serveur de ce projet permet de pouvoir effectuer le déroulement d'une partie.

1) Connexion au serveur

Pour démarrer le serveur (depuis la racine du projet) :

./bin/serveur <port>

Le serveur créera alors un socket de communication sur ce port, puis ouvrira deux socket de transmission qui attendrons les deux clients joueurs qui devront se connecter sur la même ip et le même port que le serveur (voir partie Client). Une fois ces derniers connectés au serveur, leur numéro de socket sera stocké dans un fd_select.

Une fois connecté, le serveur attendra les clients qui lui demanderont de participer à une partie. Il lui renverra alors une réponse positive, et sa couleur suivant s'il est le premier à avoir fait la demande (il sera alors de couleur blanc) ou s'il est le second (il sera de couleur noir).

Suite à cela la première partie démarrera et se déroulera suivant les règles définis dans le sujet (je n'ai cependant pas réussi à implémenté le timeout). Le serveur réceptionne alors le coup du premier joueur (le blanc), le traite pour vérifié sa validation, renvoi la réponse de la requêtes du coup aux deux joueurs puis suivant si le coup est bon ou non, il enverra au joueur noir la requête du joueur blanc et poursuivra sur un nouveau tour avec les rôles inversé. Si un coup termine une partie (que ce soit du à une erreur de coup ou à une fin normal de partie), un message s'affichera donnant le résultat de la partie et un int donnant le numéro de la partie est incrémenté permettant de poursuivre sur une seconde partie, et un autre int s'incrémentera également et terminera la partie.

Le serveur démarre alors une nouvelle partie en inversant l'ordre des joueurs. A la fin de cette seconde partie, le jeu s'arrête et le résultat de la seconde partie est affiché, ainsi que le score des deux joueurs.

2) Remarque personnelle

Le serveur n'a pas été la partie du projet la plus compliqué à faire, mais elle à prit du temps. Bien que la quasi-totalité soit fonctionnelle, il est fortement possible d'optimiser et réduite le code. Malheureusement je n'y suis pas arrivé sans déclencher des erreurs. De même, l'implémentation du timeout n'étant pas fonctionnelle (la valeur de retour étant toujours égale à 0 quelque soit le temps que met le client à répondre), je ne l'ai pas activé (j'ai cependant laissé le code de base mais je l'ai contourné).

Mis à part cela, le serveur semble fonctionné correctement, même s'il met du temps à afficher le résultat de chaque tour pour une raison que je n'ai pas trouvé.

Interface C-Java-Prolog

Nous n'avons pas réussi à finir cette partie. Nous avons un premier jet d'un client java de test possédant une stratégie de base simple et qui fonctionne (mais pas avec nos client), mais il n'y a aucun lien avec l'IA en prolog. Pour démarrer le client java (depuis la racine du projet) avec port le port propre au client java. :

Java –cp ./bin interfaceCJavaProlog <port>

Résumé

Ce document contient une approche du développement réalisé pour le projet commun aux modules de SCS, Systèmes Communicants Synchronisés et de MOIA, Méthodes et Outils pour l'Intelligence Artificielle. Ce projet a été réalisé dans le cadre de la première année de master Informatique à l'UFR-ST de Besançon

Summary

This document contains a development approach done for the project common to the modules « SCS », Systèmes Communicants Synchronisés and « MOIA », Méthodes et Outils pour l'Intelligence Artificielle. This project has been done in the context of the first year of Computer Sciences Master, at the UFR-ST, in Besançon.