FIP2 – Systèmes avancés 2019-2020

Projet de programmation Sockets

Ce projet est à réaliser en **binôme** et la note sera intégrée à la note de l'examen final du module. Un rapport de 10 à 15 pages devra être rendu avec les sources commentées de votre projet.

Ce projet consiste en la réalisation d'un Othello en utilisant l'API Socket en langage C pour faire communiquer le programme de deux joueurs exécutés localement sur la même machine.
Othello est un jeu de stratégie tour par tour à deux joueurs (Noir et Blanc) composé d'un plateau unicolore de 64 cases (plateau 8x8) avec au maximum 32 pions noirs et 32 pions blancs identiques.
En partant de la position initiale (voir image ci-après), l'objectif de chaque joueur est d'avoir un maximum de pions de sa couleur en fin de partie.
C'est toujours le joueur Noir qui commence. A son tour de jeu, chaque joueur doit effectuer un *coup légal*, c'est-à-dire poser (i) un nouveau pion de sa couleur sur une case vide, (ii) de sorte à encadrer un ou plusieurs pions adverses entre le pion qu'il pose et un pion de sa couleur présent sur le plateau. L'*encadrement* des pions adverses peut être réalisé horizontalement, verticalement, ou en diagonale (sans limite de case, excepté celle imposée par le plateau). Les pions ainsi encadrés changent de couleur pour prendre celle du joueur ayant effectué le dernier coup légal.
Durant la partie, aucun pion n'est retiré ou déplacé une fois posé sur le plateau. La partie s'achève lorsqu'aucun des deux joueurs ne peut plus effectuer de coup légal (en général lorsque les 64 cases du plateau sont occupées).

L'approche la plus simple pour réaliser ce projet est de choisir une architecture Peer-to-Peer pour les échanges entre les deux joueurs. A savoir, le programme de chaque joueur aura à la fois une partie cliente et une partie serveur pour les échanges de messages liés aux coups des joueurs. Une seconde partie de ce projet consistera à introduire un serveur stockant les informations sur les joueurs, auquel chaque joueur pourra se connecter afin d'obtenir l'adresse et le numéro de port d'un autre joueur pour démarrer une nouvelle partie. Pour cette partie du projet, vous opterez pour une architecture Client/Serveur.
Les communications devront être effectuées en utilisant l'interface sockets. Nous utiliserons le protocole de transport TCP afin de s'assurer de la bonne transmission et de l'ordre des messages.

Pour implémenter le projet, vous devrez utiliser le langage C et l'API socket.h. Si nécessaire, vous pouvez utiliser les tutoriels (le premier est synthétique et les deux autres sont plus complets) expliquant toutes les bases nécessaires pour la programmation en langage C :
http://franckh.developpez.com/articles/c-ansi/bien-debuter-en-c/
http://emmanuel-delahaye.developpez.com/tutoriels/c/initiation-pragmatique-c/
http://melem.developpez.com/tutoriels/langage-c/initiation-langage-c/

Vous serez amené à vous interfacer avec l'interface graphique fournie réalisée en C avec la librairie graphique GTK+3.0[1] et l'IDE Glade[2].
Le programme *othello_GUI.c* comprenant l'interface graphique GTK+ devra être compilé comme suit (exemple en ligne de commande avec gcc) :

```
gcc –Wall –o othello_GUI othello_GUI.c $(pkg-config --cflags --libs gtk+-3.0)
```

[1] https://developer.gnome.org/gtk3/stable/index.html
[2] https://glade.gnome.org

Chaque programme client pourra être lancé en passant en paramètre le numéro de port à utiliser :

```
./client 55555 &
```

L'interface fournie est décomposée en 4 parties (comme schématisé dans la figure ci-après) :

- Cadre connexion au serveur : permet d'entrée les informations pour la connexion au serveur
- Cadre joueurs : espace affichant les joueurs et permettant de se connecter à l'un d'entre eux
- Cadre score : indiquant la couleur de chaque joueur et son score durant une partie
- Plateau : affichant le plateau de jeu

Afin de simplifier le développement, nous considérerons que les programmes des deux joueurs s'exécuteront sur la même machine. Chacun utilisera l'adresse locale 127.0.0.1, mais un numéro de port différent pour les différencier (par exemple les ports 55 555, 55 556, …).
Le serveur aura la charge de maintenir la liste des joueurs connectés à lui et de leur envoyer une liste à jour, en cas de modification.
Vous êtes libre d'utiliser l'architecture serveur que vous désirez (itératif ou parallèle) en justifiant votre choix, cela sera pris en compte dans la note finale.

Vous aurez la charge de définir les types de messages nécessaires pour les différentes parties de l'application, ainsi que le format d'échange des données (taille messages, type de données échangées ...). Un soin particulier devra être mis sur la gestion des erreurs retournées par toutes les différentes primitives utilisées, ainsi que sur la clarté et les commentaires de votre code.

Votre rapport devra comporter une présentation de vos choix concernant :

- les formats des messages,
- les algorithmes des processus composant le projet,
- les difficultés rencontrées,

et sera accompagné des codes sources commentés de votre projet.