Le Voyageur de Commerce

FALTREPT Bérénice et ESPIAUT Marc-Alexandre

année 2014/2015

Table des matières

Présentation

notre projet est centré sur un aquarium distribué. L'objectif étant de que les items mobiles de l'aquarium soient dépendant d'un aquarium créateur mais visibles sur les divers ordinateurs reliés au serveur.

Travail effectué

Nous avons mis en place un exécutable pouvant être soit serveur soit client en fonction de la demande de l'utilisateur via les paramètres. Dans les deux cas, l'utilisateur a accès à un aquarium disposant de ses classes locales capable de se reproduire et les deux classes fournies nativement ont la fonctionnalité *piscivore* : une classe est capable de manger l'autre.
Une fois le contact établit entre le serveur et un client, les classes et les items mobiles créés dans le client, puis ceux déjà existants dans le serveur, sont transmit pour être présents dans les divers aquariums.

Avantages

Nous avons actuellement un aquarium animé dont une partie des items mobiles proviennent effectivement d'autres poste. Certaines fonctionnalités permettent d'apporter du changement dans la "faune" de l'aquarium pour lui apporter quelques intérêts.

A revoir

Nous avons un problème de sur-reproduction parmi l'espèce dominante que nous voudrions résoudre.
Nos transmissions entre serveur et clients est imparfait : actuellement les items transmit sont limités en nombre, mais surtout la transmission des positions est correcte mais leur application ne se fait pas.
Les poissons arrivés n'ayant pas de target précis, ce n'était pas nécessaire sachant que leurs positions sont supposés être mises à jour régulièrement depuis le client qui définit leurs caractéristiques, ils ont tendance à quitter l'aquarium par la droite.

Les Modules

Nombre de classes indispensables à ce projet nous ayant été fournis, nous n'allons pas les décrire en détail ici.

Les classes statiques

Main

Cette classe sert, comme son nom l'indique, au lancement de l'exécution du programme. elle permet notamment de tenir compte des arguments donnés par l'utilisateur pour lancer soit un serveur, soit un thread client.

Constante

Cette classe est une énumération des commandes acceptées par notre protocole. Elle permet d'assurer la conformité des commandes qui seront passées, mais surtout de rendre le code du protocole plus lisible.

Protocole1

Il s'agit de l'implémentation de notre protocole. Il formate les informations pour envoyer les données et récupère les informations depuis les String reçus. Suivant le protocole choisit sur le framapad commun, nous avons optés pour un transfert des informations par String où la séparation entre les commandes est faite par le symbole "#" et la séparation des données au sein d'une commande se fait par "!".

Les classes instanciables

ServerThread

Le ServerThread est la classe instancié si l'utilisateur lance un serveur via le main.
Elle créée les variables nécessaires au contact avec les clients mais aussi à la gestion de l'aquarium initial du serveur.
Pour s'assurer que le nombre de clients ne posera pas de problèmes durant l'exécution, nous utilisons la classe *ScheduledExecutorService* qui va créer 10 Threads. Ces Threads traiteront les envois et réceptions des sockets des clients, quel que soit leur nombre.

Client

La classe Client est instancié lorsque l'utilisateur demande à se connecter en tant que client à un serveur. C'est une classe qui étend la classe Thread et qui va, elle aussi, utiliser une instance de *ScheduledExecutorService*, mais cette fois avec seulement trois Threads car il n'a qu'un seul

contact à traiter.

ClassesMobiles

Cette classe est utilisée pour stocker les différent types, donc classes, d'items mobiles utilisés dans l'aquarium. Une liste de ClassesMobiles se trouve dans l'aquarium. Elle contient toutes les classes actuellement présentes dans l'aquarium.
Cette classe a été initialement conçue pour permettre de transférer les images des différentes classes et les stocker.
Ainsi nous aurions pu ne les envoyer qu'une seule fois et faire une économie de bande passante. Malheureusement, nous n'avons pas eu le temps d'implémenter le transfert d'images, elle a donc actuellement un usage moins essentiel.

Aquarium

La classe aquarium nous a été fournie, mais de nombreuses modifications sont intervenues et ses fonctionnalités ont changées. Tout les accès aux variables de la classe se font via des accesseurs qui limitent les risques de mauvaises manipulation par le/les protocole(s) qui peuvent être utilisés.
L'organisation des items a été changée, il y a à présent trois listes.

- Une reprenant les items initiés dans cet aquarium, mobiles ou non.
- Une seconde contenant une représentation simplifiée des instances mobiles créées dans un aquarium en contact avec celui-ci.
- La dernière contient à chaque instant la liste des classes d'éléments mobiles actuellement présentes dans l'aquarium.

Ceci pour permettre d'avoir un premier tri des items et de ne pas avoir à parcourir l'ensemble d'une liste contenant des items créés localement et d'origines éloignés pour accéder à un item en particulier.

Conclusion

Le sujet, à la fois simple et visuel, est intéressant pour un tout premier projet en réseau. Pour des débutants, il est satisfaisant de voir les effets concrets de nos modifications sur le code du projet. Mais surtout, faire de la programmation objet était un souhait pour l'un d'entre nous. ce fut donc intéressant.
Ceci dit, nous regrettons de ne pas avoir aboutit nos plans, qui se sont avérés trop grands. Nous avions prévus de faire passer les images des poissons entre les différents protagonistes, mais cela n'a pas été possible à cause d'une organisation limitée.

Plus gênant, des fonctionnalités plus basiques que nous avons commencés à implémentées n'ont pas encore l'action souhaitée de façon optimum : surpopulation, non transfert de tout les poissons.
Ceci dit, cela reste une bonne expérience en réseau comme pour la programmation orienté objet et nous espérons continuer à nous améliorer par la suite.

Annexe