Projet Réseaux RE203 : "Simulation d'un aquarium de poissons" Mars 2021

Organisation

- Les élèves doivent s'organiser en équipes de 4-5 élèves. Utiliser la forge Thor
- La programmation se fera obligatoirement en C et un langage orienté objet (C++, Java, C#). Le contrôleur « Aquarium» en C et les programmes d'affichage (GUI) et des Poissons en Java - utilisation de Java FX ou autre est préconisée.
- Un rapport intermédiaire d'avancement est à remettre à la fin de la 5ème séance.
- La 10ème séance est réservée aux soutenances du projet. Le rapport final doit être rendu avant la soutenance (à la fin de la 9ème séance).
- L'accès à la forge doit être fourni au début du projet et les fichiers source à la fin du projet sous forme d'une archive (tar.gz) à la fin de la 9ème séance.
- Privilégier une gestion du projet en utilisant la méthode Agile.

Mises à jour du protocole

Consulter régulièrement cette page, des modifications seront publiées si nécessaire :

Présentation

Le but du projet est de réaliser un aquarium centralisé de poissons. Chaque équipe devra programmer un programme d'affichage et un programme contrôleur. Le contrôleur permet de centraliser la gestion de l'aquarium, il pourrait être supprimé dans une version distribuée. Chaque poisson a pour mission de se déplacer dans l'aquarium selon un modèle de mobilité prédéfini (vitesse, direction, etc.). Le programme contrôleur s'occupe d'informer les programmes d'affichage (Vue) de la position actuelle des poissons. On optera pour une programmation suivant le modèle MVC. Le modèle contient les données sur un poisson, sa taille, son pattern de mobilité, etc. La vue, c'est l'IHM, elle représente l'affichage graphique de l'aquarium. Le contrôleur permet de faire le lien entre le modèle et la vue et il permet de contrôler les données à transmettre d'un affichage à l'autre selon la topologie de l'aquarium - voir Figure 1.

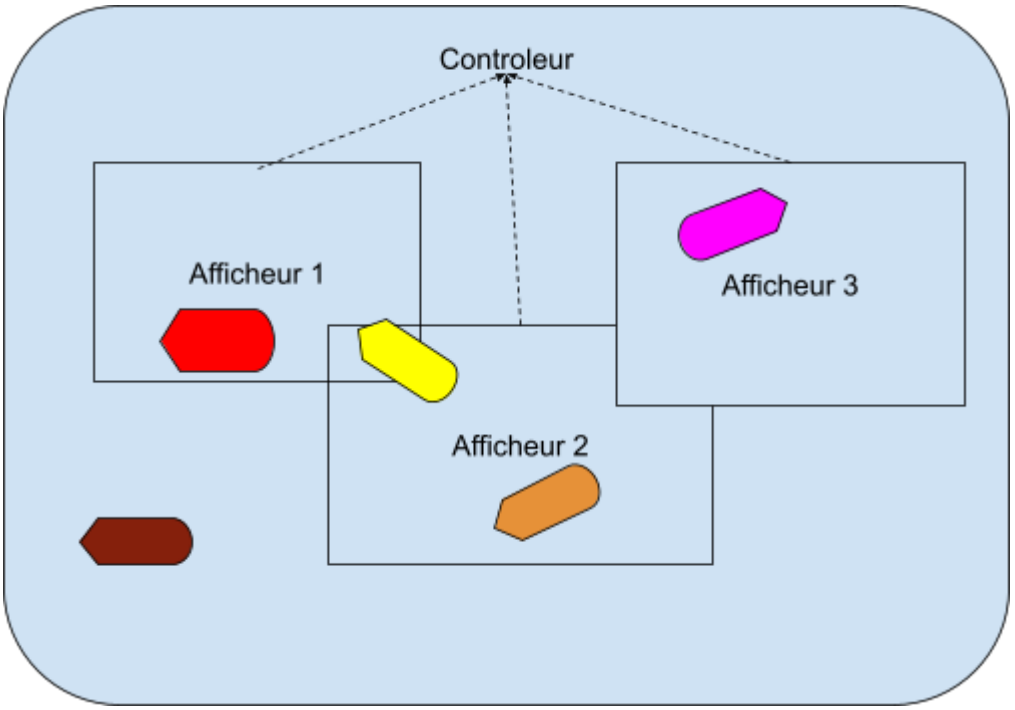

Figure 1: Topologie Réseau d'un aquarium

L'interaction entre les programmes est illustrée dans la figure 2 suivante :

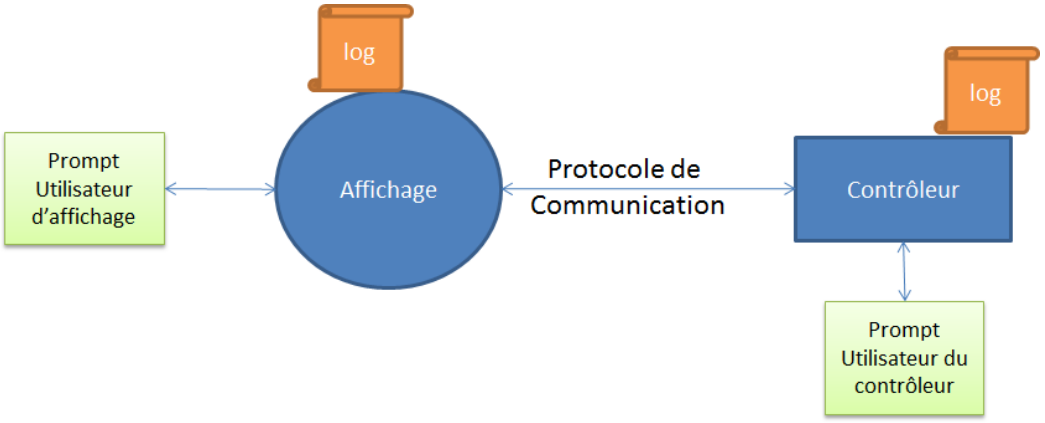

Figure 2: Illustration de l'interaction entre utilisateur, l'affichage et le contrôleur

Programme "contrôleur"

Le contrôleur est un programme de type serveur, c'est à dire en attente de connexions clientes. Au démarrage, il charge une liste de programmes d'affichages, c'est-à-dire les futurs clients impliqués dans l'aquarium. Le contrôleur chargera l'aquarium sous le format de fichier suivant:

```
Aquarium_width x Aquarium_height
N1 vue_x x vue_y + vue_width + vue_height
N2 vue_x x vue_y + vue_width + vue_height
N3 vue_x x vue_y + vue_width + vue_height
N4 vue_x x vue_y + vue_width + vue_height
```

Par exemple:

1000x1000
N1 0x0+500+500
N2 500x0+500+500
N3 0x500+500+500
N4 500x500+500+500

Le programme contrôleur devra permettre des commandes simples pour charger une topologie, la modifier en cours du temps, et la sauvegarder si besoin. Les commandes suivantes doivent être implémentées:

($ Signifie l'invite de commande ou le prompt du Contrôleur)

- Chargement de l'aquarium

```
$ load aquarium1
    -> aquarium loaded (4 display view)!
```

- Affichage de la topologie

```
$ show aquarium
```

```
1000x1000
N1 0x0+500+500
N2 500x0+500+500
N3 0x500+500+500
N4 500x500+500+500
```

- Ajouter une vue.

```
$ add view N5 400x400+400+200
    -> view added
```

- Supprimer une vue.

```
$ del view N5
    -> view N5 deleted.
```

- Enregistrer / exporter les données de l'aquarium actuelle dans un fichier.

```
$ save aquarium2
    -> Aquarium saved !( 5 display view)
```

A noter que plusieurs vues peuvent être chevauchantes, dans ce cas les poissons s'afficheront potentiellement sur chacune des ces vues. Au moment de l'ajout, le contrôleur vérifiera que le vue rentre bien dans l'aquarium.

Programme "Affichage"

Le programme d'affichage est un programme client qui se connecte au contrôleur au démarrage du programme et reste connecté durant toute la durée de vie de la session. Il a pour objectif d'échanger des données avec le contrôleur et de les interpréter pour un affichage. Ces données concernent les poissons à afficher. Par la suite, toutes les coordonnées sont des entiers correspondant à un pourcentage de la largeur et de la hauteur de l'écran. Ainsi, si l'écran fait 1280x720 Pixel, la coordonnée 0x0 correspond au point 0x0, 100x100 au point 1280x720, et 50x50 au point 640x360.

Toutes les commandes sont rentrées au niveau de l'affichage, communiquées au contrôleur, exécutées au niveau du contrôleur, et la réponse d'exécution est communiquée à nouveau à l'affichage.

1ère commande: Status

Au niveau de chaque affichage, l'utilisateur peut envoyer un message de demande statut :

```
$ status
    ->OK : Connecté au contrôleur, 2 poissons trouvés
Fish PoissonRouge at 90x70,10x5

    Fish PoissonClown at 20x40,12x4
$ statut
  -> NOK : commande introuvable
```

```
$
```

Ici 90x70 : c'est la position du coin supérieur gauche du rectangle qui enveloppe le poisson. La taille du rectangle d'enveloppe est de10x5 (en pourcentage de taille d'écran).

Il est possible de compléter la réponse de la commande "status" en ajoutant l'état des poissons dans l'aquarium (started / notStarted)

Exemple

```
$ status
    ->OK : Connecté au contrôleur, 2 poissons trouvés
    Fish PoissonRouge at 90x70,10x5 started
    Fish PoissonClown at 20x40,12x4 notStarted
$ statut
   -> NOK : commande introuvable
$
```

2ème commande: AddFish

Cette commande permet à l'utilisateur d'ajouter un poisson à l'aquarium :

```
$ addFish PoissonNain at 61x52, 4x3, RandomWayPoint
     -> OK
$ addFish PoissonNain2 at 45x4, 10x10, PathWay
     -> NOK : modèle de mobilité non supporté
```

En effet, les poissons sont ajoutés par les clients puis gérés par le contrôleur (serveur).

La commande ci-dessus ajoute un poisson ayant un nom PoissonNain, enveloppé dans un rectangle virtuel de taille 4x3. Ce rectangle est positionné aux coordonnées 61x52 (coordonnées du coin supérieur gauche). Ce poisson suivra un modèle de mobilité appelé RandomWayPoint qui sera implémenté de base dans ce projet. D'autres modèles de mobilité peuvent être supportés.

Le modèle RandomWayPoint donne la possibilité au contrôleur de faire bouger un poisson vers un point précis de l'aquarium de manière aléatoire. Cette mobilité devrait durer un temps bien précis.

2ème commande: delFish

```
$ delFish PoissonNain
     -> OK
 $ delFish PoissonNain3
    -> NOK : Poisson inexistant
```

Permet de sortir un poisson l'aquarium.

3ème commande: startFish

Cette commande permet à l'utilisateur de démarrer un poisson.

```
$ startFish PoissonNain
     -> OK
$
```

Protocole des communications internes

Chaque programme d'affichage communique régulièrement durant toute sa durée de vie avec le contrôleur. Il n'y a pas de communication directe entre les programmes d'affichage dans cette version centralisée. Le protocole suivant définit la communication interne entre l'affichage et le contrôleur. Cette communication est transparente à l'utilisateur.

Conventions:

- Dans ce qui suit, le signe ">" indique le message qui a été envoyé, et le signe "<" le message reçu en réponse.
- Tous les messages sont en texte et se terminent par le caractère "\n".

Protocole de communications Affichage <–> Contrôleur

Les communications s'effectuent avec le protocole TCP. Le programme d'affichage (i.e. Client) initie la demande de connexion avec le contrôleur (i.e. serveur) en suivant les étapes suivantes :

Initialisation

```
> hello in as ID
< greeting ID
```

À l'initialisation, le programme d'affichage communique au contrôleur son identifiant qui doit correspondre à un vrai nœud dans la topologie de l'aquarium. L'identifiant est fixé au démarrage du programme d'affichage ou lu depuis le fichier de configuration.

- Si l'identifiant demandé est libre, le contrôleur autorise son utilisation.

```
> hello in as N3
< greeting N3
```

- Si le programme d'affichage n'a pas spécifié d'identifiant, le contrôleur lui attribue un identifiant libre (le reste de la commande n'est pas nécessaire dans ce cas).

```
> hello
< greeting N2
```

- Si l'identifiant demandé n'existe pas dans la topologie, ou lorsqu'il est déjà attribué à un autre affichage, le contrôleur renvoie si possible un autre identifiant libre sinon une erreur

```
> hello in as N404
< greeting N4

> hello in as N505
< no greeting
```

Demande périodique de poissons

Première demande

```
> getFishes
< list [PoissonRouge at 90x4,10x4,5] [PoissonClown at 20x80,12x6,5]
```

- Le contrôleur retourne au programme d'affichage la liste des poissons que doit gérer ce dernier.
- Ici le programme d'affichage courant doit gérer deux poissons : PoissonRouge et PoissonClown. Le PoissonRouge doit nager depuis sa position courante vers la position 90x4 (en pourcentage de la taille de l'écran). Cette mobilité (nage) doit durer 5s. A noter que les paramètres de mobilité entre la position actuelle du poisson et sa nouvelle position sont à la discrétion de chaque programme d'affichage. Le contrôleur ne fait que communiquer la position finale du poisson et la durée d'exécution de la mobilité.
- Attention: les coordonnées peuvent être négatives dans le cas d'un poisson qui entre ou sort de la vue de l'afficheur.
- **Indiquer une valeur de second =0s pour afficher instantanément un poisson à une position par exemple lorsqu'il rentre dans une vue.** `list [PoissonRouge at 90x4,10x4,0] [PoissonClown at 20x80,12x6,5]` ⇒ *ici PoissonRouge s'affiche dès la réception de cette commande à la position indiquée.*

Demande continue de Poisson

```
> ls
```

```
< list [PoissonRouge at 92x40,10x4,5] [PoissonClown at
22x80,12x6,5]
< list [PoissonRouge at 70x40,10x4,5] [PoissonClown at
10x90,12x6,5]
< list [PoissonRouge at 30x30,10x4,5] [PoissonClown at
30x30,12x6,5]
```

- Le contrôleur retourne au programme d'affichage la liste des poissons que doit gérer ce dernier en continu, chaque x seconds une nouvelle réponse est générée qui prend en compte les aspects de mobilité des poissons.
- A noter que le contrôleur peut communiquer *a priori* le chemin complet que doit prendre un poisson en suivant des points bien précis. Dans ce cas, le programme d'affichage doit parser toutes les positions et les exécuter une à la suite des autres.

Remarques

Les algorithmes de déplacement (RandomPathWay par exemple) sont à destination du contrôleur: il détermine la façon dont le serveur va déplacer le poisson : une nouvelle position toutes les 5 secondes par exemple. On pourra imaginer un algorithme HorinzontalPathWay qui fait qu'un poisson se déplace toujours horizontalement et toujours dans le même sens (il devrait ainsi tourner entre les clients).

Concernant getFishesContinuously, plusieurs solutions sont possibles côté contrôleur:

```
> getFishesContinuously

< list [PoissonRouge at 92x40,10x4,5] [PoissonClown at
22x80,12x6,5]

< list [PoissonRouge at 70x40,10x4,5] [PoissonClown at
10x90,12x6,5]

< list [PoissonRouge at 30x30,10x4,5] [PoissonClown at
30x30,12x6,5]
```

Dans cet exemple, le contrôleur envoie un "liste" chaque fois qu'un poisson arrive à destination (ici au bout de 5 secondes). On pourrait également implémenter un contrôleur qui envoie un "list" toutes les secondes:

```
> getFishesContinuously
< list [PoissonRouge at 92x40,10x4,5] [PoissonClown at
22x80,12x6,3]
< list [PoissonRouge at 70x40,10x4,4] [PoissonClown at
10x90,12x6,2]
< list [PoissonRouge at 30x30,10x4,3] [PoissonClown at
30x30,12x6,1]
```

Du point de vue du client, cela ne devrait pas changer l'implémentation. Dans le deuxième cas, on confirme les destinations régulièrement, ce que l'on ne fait pas dans le premier cas.

Déconnexion

```
> log out
< bye
```

Si au bout d'un certain temps, le contrôleur ne reçoit aucun message d'un programme d'affichage, celui-ci est supprimé de l'aquarium. Pour éviter d'être considéré comme déconnecté par le contrôleur, il est donc nécessaire de le contacter à intervalles réguliers. On utilisera pour cela une commande Ping :

```
> ping 12345

< pong 12345
```

Ajout d'un poisson

```
> addFish PoissonRouge at 90x40,10x4, RandomWayPoint
< OK
```

Si le nom du poisson existe déjà dans l'aquarium:

```
> addFish PoissonRouge at 90x40,10x4, RandomWayPoint
< NOK
```

Suppression d'un poisson

```
> delFish PoissonRouge
< OK
```

Si le nom du poisson n'existe pas dans l'aquarium:

```
> delFish PoissonRouge
< NOK
```

Démarrage d'un poisson

```
> startFish PoissonRouge
< OK
```

Si le nom du poisson n'existe pas dans l'aquarium:

```
> startFish PoissonRouge
< NOK
```

Log

Tous les programmes doivent contrôler le degré de verbosité en affichant des informations dans un fichier log ou une console de log.

Visuels

Les poissons sont affichés en utilisant des visuels dans un répertoire indiqué dans le fichier de configuration (voir ci-dessous). Ainsi, le poisson *PoissonRouge* sera affiché selon le fichier *PoissonRouge.png.* Si aucun visuel n'est trouvé alors un visuel au hasard ou par défault sera choisi pour représenter le poisson à l'écran. Lors de la recherche de visuel, il faudra ignoré les caractères après un _ (underscore) dans le nom. Ainsi pour le poisson *PoissonClown_2*, il faudra chercher un fichier *PoissonClown.png*. Tous les visuels seront des fichiers au format png.

Configuration des programmes

Chaque programme devra s'appuyer sur un fichier de configuration disponible dans le même dossier du fichier exécutable avant le lancement du programme.

Fichier de configuration (*affichage.cfg*):

```
# Adresse IP du nœud contrôleur
controller-address = a.b.c.d

# Identifiant du programme d'affichage
id = N1

# Numéro de port TCP d'écoute du nœud contrôleur
controller-port = 12345

# Valeur par défaut pour la transmission du ping.
display-timeout-value = 30

# Répertoire relatif ou absolu où se trouve les visuels pour les
poissons.
resources = ./fishes
```

Fichier de configuration (controller.cfg):

```
# Numéro de port TCP d'écoute
controller-port = 12345

# Temps entre 2 requêtes au-delà duquel un programme d'affichage
est retiré du réseau.
display-timeout-value = 45

# Intervalle en secondes pour l'échange périodique de poisson. cf.
commande GetFishesContinuously
fish-update-interval = 1
```