Rapport iSketch Project

Adrien Becchis, Stéphane Ferreira

December 16, 2013

Contents

Introduction:
Ce présent rapport viens décrire le projet de Client Serveur iSketch, réalisé au cours de ce semestre du cours de Programmation Concurrente Réactive et Répartie.

1 Guide utilisation

1.1 Lancement Serveur

Le serveur est invocable depuis la ligne de commande, en appelant la jar, et en spécifiant les options dont voici la liste:

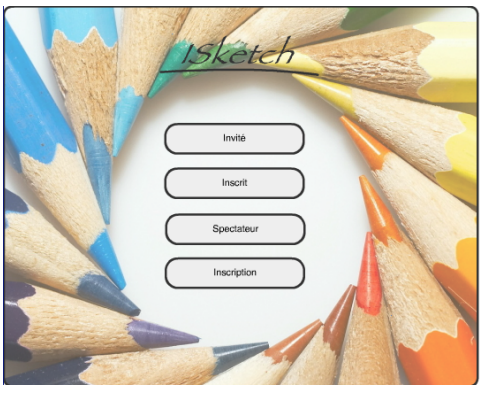
```
Usage: assKetch [options]
  Options:
    -cheatPenalty, --cheatPenalty   Point de malus
                                    Default: 10
        --comptes                   Le fichier des comptes
                                    Default: comptes.ser
    -D, --daemon                    Serveur enchaine les parties (non oneshot)
                                    Default: false
    -dico, --dico                   Les mots à utiliser
                                    Default: dicotest
    -max, --max                     Nombre maximum de joueurs
                                    Default: 4
    -nbwarn, --nbwarn               Nb pour mettre malus joueur
                                    Default: 3
    -port, --port                   Port à utiliser
                                    Default: 2013
    -portstats, --portstats         Port à utiliser pour le serveur de stats
                                    Default: 2092
    -timeout, --timeout             Délai après découverte d'un mot
                                    Default: 30
    -timepause, --timepause         Temps de pause entre parties
                                    Default: 2
    -timeround, --timeround         Durée Max partie
                                    Default: 180
```

La liste des options possibles du serveur est consultable en invoquant le programme avec l'option -h, qui affichera l'usage.

Le serveur est conçu pour tourner avec tout client respectant le protocole, mais doit être lancé avec une option particulière `-a` en mode actionscript pour pouvoir tourner avec le client ActionScript mis au point par stéphane.

1.2 Lancement du client.

Il s'agit juste de lancer la page Html `Main.html` dans un navigateur disposant une d'un player flash.
Le joueur se laissera ensuite naturellement guider par l'interface au design standard.



2 Conception Serveur

par Adrien

2.1 Intro

Le serveur a été écrit en Java, et ceci pour plusieurs raisons, parmi lesquelles le fait qu'on ne compter pas l'utiliser pour réaliser le client et son interface graphique

Une des raisons principales était aussi de pratiquer le Java sur projet conséquent, vu que j'avais qu'une expérience assez limitée avec celui-ci.
Le Java par sa nature haut niveau, et objet était aussi tout adapté pour nous permettre de gérér plus facilement la complexité de l'application, et de produire un code plus structuré, plus lisible et maintenable.

2.2 Architecture

A défaut de beaux diagramme Uml voici une brève explication littéraire de l'organisation mise en place pour le serveur.

Le code est réparti dans trois package: `core` (principalement les threads gestionnaires), `game` (les objets métiers graphiques et joueur), et tools pour les outils/fonctions statiques.
La classe serveur est la classe qui vient tout lier, son thread se charge principalement d'initialiser les données et de lancer les autres threads. Le thread Game Manager va gérer les parties, et les threads JoueurHAndler vont gérer l'intéraction avec les joueurs. à noter qu'à coté un Thread fera officie de serveur de statistiques, et que les thread ConnexionStacker, et Connexion Handler se chargeront de gérer les nouvelles connexion.
incomplet

Du coté des objets métiers, rien de très exotique.
incomplet

Les extensions avaient été prises en compte lors de la conception de l'architecture globale du serveur pour permettre de les mettre en place sans tout chambouler.
Une fois le squelette mis en place les fonctionnalités de bases, puis avancées ont été petit à petit incorporées, testées et debuguées.

2.3 Autres

Voici quelques points dont je pense qu'il est utilise de préciser l'intention et le fonctionnement (et de mieux comprendre l'architecture du programme):

2.3.1 Lanceur, JCommander

Pour gérer facilement les options, j'ai utilisé la bibliothèque/jar Jcommander. Celle-ci va se charger de parser la ligne de commande, de vérifier les options fournies et de gérer la valeur des options par défault.
Les options sont dans la classe interne `Option` de la classe Main `ASServer`.
Jcommander marche avec un système d'annotation au dessus des variables d'instances que l'on veut paramétrer. Ce qui permet un code plus clair et facilement adaptable.
La classe principale contient une unique instance statique du programme ce qui permet à toute classe d'accéder au options si besoin est.

Ceci a été mis en place dès le début pour pouvoir tester le programme le plus rapidement possible

2.3.2 Protocole Parseur, et CommandesExceptions

Pour vérifier les commandes reçues des clients, mais aussi mes construires, j'ai construit une classe Protocol qui se charge aussi de créer les commandes envoyées au client.
Ceci a permis de centraliser tous le code en lien au Protocole, ce qui rend la maintenance et modification apporté

au protocole (nom commande, arité) bien plus pratique.

Une série d'exception a aussi été créer pour représenter les commandes invalides: `IllegalCommandException` qui est précisée par `InvalidCommandException` `UnknownCommandException` `WrongArityCommandException`

Concrètement, la classe contient une hashmap entre le nom de la commande, et les caractéristiques associées à celle-ci (etat légal du joueur, arité de la commande).
Elle fournis aussi une méthode statique `parseCommand` qui va découper la commande, vérifier si elle correspond bien au protocole, et à l'état courant du joueur, avant de retourner un tableau contenant les tokens.
Si la commande est invalide, l'exception adaptée sera émise, puis géré par le module ayant demandé de parser la commande (surtout `JoueurHandler`, et `connexionHandler`)

En plus de ceci, la classe Protocole contient toutes les méthodes pour construire les chaines de caracatères correspondant aux commandes émises, à l'image d'une `factory`

2.3.3 "Automates" client

Après avoir longuement réfléchis sur comment pouvoir gérer les clients de manière simple et en intéraction avec un thread gérant le jeu, je suis parti sur une sorte d'automate (à défaut de meilleur désignation).
Les threads en question, `JoueurHandler` se voient assigner un joueur unique donc il a la charge. Il va Écouter en permance que le joueur va lui dire, et traiter la commande reçu.
Ce traitement sera différent selon le role du joueur à savoir "dessinateur", "chercheur", "indéterminé"... sachant que les commandes reçues
Le thread va donc vérifier si la commande est valide (et si non, informer le client qu'elle est incorrecte), puis invoquer sur le gestionnaire du jeu la méthode associé à la commande.

2.3.4 Loggeur

Afin d'avoir un débugage plus simple, et un meilleur traçage, une classe IO a été mis au point.
Plutôt que d'utiliser des sysout à tout va, on utilise des `IO.trace`, `IO.traceDebug` ce qui permet de rafiner les messages tracé en un seul endroit. La méthode affiche sur le flux de sorti le thread qui affiche le message, ainsi qu'un time stamp.
Les messages debug ne sont affichés qu'en mode débug, et il est possible d'activer désactiver l'ajout d'un time stamp.
La trace peut aussi être enregistrée dans un fichier.

2.4 Problèmes rencontrés

Le plus dur a été de bien organiser le programme, d'affecter les responsabilités, et de séparer clairement les objets "métier", des objets gestionnaires/threads. Une fois l'architecture mis au point, il était bien plus simple et facile de savoir où placer tel information, ou apporter tel changement.

Le débugage en isolation a été bien plus facile que pour le client, avec l'utilisation de telnet, bien qu'en pratique assez fastidieux et répétitif.
L'idéal aurait été bien sur d'utiliser le framework de test, mais le temps était limité pour apprendre à utiliser JUnit.

§here
Choisir le bon niveau de synchronisation sur les objets.

L'adaptation du serveur pour tourner avec actionscript a été d'ailleurs assez difficile, et un mode a du etre fais pour gérer ces spécificités: notamment la gestion de la sécurité (il fallait répondre à une connexion sur la socket qui se contentais de demander un fichier policy), et pour une fin de commande particulière avec un `\0`.

3 Conception Client

par Stéphane

3.1 Initialisation:

Pour tester le client, il suffit d'ouvrir le fichier «Mail.html»
Pour ouvrir les sources du clients, une partie des sources sont disponicle dans les fichier.as, l'autre partie dans le Main.fla qui peut êtdonere ouvert avec Adobe Flash CC (Essais 30 Jours).
Le fichier Main.swf est le fichier compilé.
Pour lancer le client sur un autre serveur il faut que dès la connexion du client, le serveur lui envois un policy file requeste:

```
private final static String ACTION_POLICY_STRING = "<?xml version=\"1.0\"
 encoding=\"UTF-8\"?><cross-domain-policy><allow-access-from domain=\"*\"
 to-ports=\"*\" secure=\"false\" /><site-control permitted-cross-domain-po
 licies=\"master-only\" /></cross-domain-policy>\0";
```

Pour changer le port et l'ip du serveur, il faut ouvrir le fichier Main.fla, allez dans la premeire image d'action du sénario, et changer les 2 variables HostIp et HostPort.

3.2 Fonctionnalités du client:

Lorsque l'on lance le client, on arrive sur un menu.
Si la connexion ne s'établie pas au boue de «timeout», un popup nous indique que le connexion avec le serveur est impossible. Un popUp nous indiquera également si nous avons été expulsé par le serveur ou bien si le serveur a planté. (Les fenêtres popUp sont calculé dynamiquement pour etre centré sur la page et avec le bon rapport taille/texte)
Lorsque l'on clique sur les différents sous menus, on peut soit revenir en arrière en cliquant sur la petite flèche, soit passer à l'étape suivante en cliquant sur le bouton, ou en appuyant sur la touche entrer.
Une fois en attente de la partie, nous avons en haut a droite un bouton pour les options, aide qui va faire pop une fenêtre d'aide, signaler le dessinateur en cas de triche actif que si la partie est commencé, et quitter la partie.
La fenêtre IRC est également active.
Une fois la partie commencé, la fenêtre Réponse devient actives. On peut envoyer des message a l'aide du bouton ou de la touche entrer. Le dessinateur ne peux pas parler dans la fenêtre réponse.
La fenêtre d'information sur la partie est mise à jour en temps réel grâce au serveur.
Seul le dessinateur peux dessiner dans la fenêtre de dessin.
Le premier bouton (en haut) sert à effacer le dessin, le second à augmenter la taille du trait (borné), puis remettre la taille du trait par défaut à 5, puis à diminuer la taille du trait(borné), puis changer la forme de la palette, indique la couleur actuel du trait, choisir la couleur.

3.3 Indications sur le code.

Dans le fichier Main.fla, on trouve l'initialisation du client:

```
//Feuille 1:

//Mettre la tage du wheig et height de la meme taille que la résolution max
import flash.display.StageAlign;
import flash.display.StageScaleMode;
import flash.display.MovieClip;
stage.align = StageAlign.TOP_LEFT;


//adress Ip du serveur
var HostIp:String = "localhost";
//Port du serveur
var HostPort:Number = 2013;
```

```
Security.loadPolicyFile("localhost:2013");

//stage.scaleMode = StageScaleMode.NO_SCALE;
trace(HostIp);
trace(HostPort);
//Instenciation de la connexion socket

//Instenciation de la page d'accueil
var accueil:AccueilPage ;
//Enleve le Zoom sur le client
var mainFenetre:Fenetre;
//Interface Façade
var interf:InterfaceSock = new InterfaceSock(this);
var connexion:ConnexionSocket = new ConnexionSocket(HostIp,HostPort);
accueil = new AccueilPage(HostIp,HostPort);
stage.showDefaultContextMenu = false;
stop();

// Feuille 2:
mainFenetre = new Fenetre();
stop();
```

La classe ConnexionSocket gère toute les connexions entrante et sortante, la classe InterfaceSock regroupe tout les traitement liée au protocole.

La classe Fenetre est un container des autres «modules» (Dessin, InfoPartie, LesCo, IRC, Reponse)

La page d'acceuil est lancé sur le première feuille et la fenêtre, sur la deuxième feuilles.

4 Extensions & Modif protocole.

4.1 Ajout au protocoles

Quelques ajouts au protocole de base on été effectués afin de rajouter de nouvelles fonctionnalités

CLEAR C->S Le dessinateur spécifie au client qu'il veut e

CLEARED S->C Le serveur indique aux chercheurs que le dessin a été effacé

Une autre modification au protocole est une modification des règles de triches. (d'ailleurs plus proche des premières version du sujet).
Si le dessinateur s'avère être coupable de triche, il ne sera désormais plus exclu du jeu comme précédemment, mais affligé d'un malus. (10 point par défault.). Ceci permet not §HERE

4.2 Room Multiples.

La possibilité pour le serveur de gérer à la fois un certain nombre de parties simultanément a été envisagé mais n'a pas été réalisée, faute de temps notamment.

Cependant les modifications à apporter sont assez mineures:
La `ListeJoueurs`, et les `joueurHandlers` pourraient passer directement dans le gameManager, et le serveur aurait plusieurs `gameManager` à la place d'un seul actuellement.
Le serveur remplissant une session/room jusqu'à ce que celle-ci soit pleine, lancant le gameManager avant de passer sur une autre session, dans la limite des places disponibles.

Le protocole devra par contre être modifié et étendus, pour permettre notamment au spectateur de choisir la room à observer, au client de savoir dans quel room il est (et éventuellement demander d'en changer avant que la partie ait commencer)