ECOLE POLYTECHNIQUE

Set multijoueurs

Zhixing CAO, Yuxiang LI

1er avril 2015

1 INTRODUCTION

Ce projet consiste à développer une application sur Android en appliquant la programmation concurrente pour réaliser une communication serveur-client. Cette application permet de jouer un jeu appelé **Set** soit en mode solo, soit un multijoueur. On présentera d'abord la règle du jeu et leur correspondance en terme de Java ou Android. On expliquera ensuite plus en détail la réalisation de tous les modules nécessaires à ce jeu, on verra en quoi la programmation multithread permet d'avoir une meilleure performance de l'application.

2 PRINCIPE DU JEU ET PROTOCOLE DE COMMUNICATION

2.1 COMMENT JOUER LE JEU ?

2.1.1 PRÉSENTATION

Le jeu Set est un jeu de cartes constitué de 81 cartes toutes différentes qui se distinguent selon 4 caractéristiques : la couleur, la forme, le type de remplissage et le nombre d'objets. Le but du jeu est de trouver une ensemble de 3 cartes qui ont des caractéristiques soit toutes les trois identiques, soit toutes les trois différentes. Le jeu peut se jouer à un ou plusieurs joueurs, à chaque tour, on dispose 12 cartes sur la table, et le premier joueur qui trouve un bon set de 3 cartes gagne et ces 3 cartes vont être remplacés par 3 autre. S'il n'y a pas de bon set parmi ces 12 cartes, on rajoute 3 cartes supplémentaires en supposant qu'il y ait toujours un set parmi elles.

2.1.2 Choisir les cartes

Le choix des cartes se fait l'une après l'autre, les cartes entourées du bleus sont choisis, reclicker sur la carte déjà choisie permet de l'enveler du set. Une fois qu'on a 3 cartes, on soumet directement la solution.

2.1.3 Score et pénalité

En mode solo, le set est jugé sur place, tout set correct sera coloré en vert et incorrect en rouge. La couleur disparaît après une petite durée. Une fois que le set est jugé, on ne peut plus le toucher avant la disparition de la couleur. Le set correct sera ensuite remplacé par 3 nouvelles cartes. Un set correct rapporte 10 points et on enlève 2 points pour un faux set.

2.2 Comment communiquer avec le serveur ?

2.2.1 Serveur

Le serveur gère tout le déroulement du jeu en mode multijoueur en suivant le protocle suivant :

- Initialiser le jeu : initialiser les cartes
- Changer les cartes
- Attribuer les points suivant l'ordre d'arrivée du message : le deuxième ne reçoit rien
- Débloquer le jeu si tous les joueurs sont bloqués
- Faire sortir un joueur après une longue durée de silence

2.2.2 Client

Le côté client est plus simple.

- Recevoir les message : changer les cartes, renouveller le score ou débloquer le jeu
- Juger le set comme dans le mode solo
- Se bloquer pour un faux set et soumettre le résultat pour un bon
- Quitter le jeu en prévenant le serveur

3 Du côté de Java

3.1 Android

La partie Android se divise principalement en 2 parties : la class `CardView` correspondant à View de **MVC** et `MainActivity` jouant à la fois le rôle de Model et de Controller.

- `CardView` et `Card`
  Bien qu'on n'ait que 81 cartes, on a choisi de les modéliser avec une class au lieu d'un entier, ce choix permet d'avoir une plus grande liberté et clarté du code, la méthode `hashCode` et le constructeur permettent de créer une bijection entre les deux espaces d'états pour faciliter la communication. Tandis que `CardView`, qui implémente la classe `View` gère l'affichage des cartes suivant les paramètres (membres de

la classe) : carte de score, carte gêlée, carte correcte, carte choisie, etc. En conséquence, les `OnClickListener` sont directement associés à ces `CardView`, ce qui évite de compliquer le traitement des événements.

— `MainActivity`
Cette classe se charge d'interagir avec l'utilisateur et de distribuer le travail au sein de l'application (communication Socket, calcul, affichage). Il permet d'augmenter ou diminuer le score, changer les couleurs des 'CardView' et remplacer les cartes selon les cartes choisits par l'utilisateur. Plusieurs classes du type `Thread` et `Runnable` sont définies dans cette partie en rendant l'application multi-tâches.

3.2 Serveur

Le code du serveur est réalisé en-dehors de l'application Android par une nouvelle classe : `Server`. Cette classe gère la création d'un nouveau `Thread` à chaque nouveau joueur, l'attribution du point, le bon déroulement du jeu. Les différents messages sont listés ci-dessous :

— (Serveur) Modifier les cartes : V␣pos_1␣$card_1$␣pos_2␣$card_2$␣pos_3␣$card_3$
— (Serveur) Incrémenter le score : S␣$card_1$␣$card_2$␣$card_3$
— (Serveur) Faire sortir le joueur : E
— (Client) Demander le redémarrage : R
— (Client) Informer sur l'état de blocage : F
— (Client) Soumettre le résultat : S␣pos_1␣$card_1$␣pos_2␣$card_2$␣pos_3␣$card_3$
— (Client) Quitter le jeu : E

4 Réalisation et détails techniques

4.1 Le tas des cartes

Pour créer le tas de carte, on tire aléatoirement 81 entiers entre 0 et 99 à qui on associe les 81 cartes. Et puis, en triant toutes ces paires d'entiers par leur valeur aléatoire, on a une liste des cartes mélangées.

4.2 Vérifier un set et `Class<?>`

La vérification d'un set nécessite de regarder tous les 4 membres de la cartes. Une vérification manuelle est possible, mais cette méthode n'est pas facilement reproductible. On a utilisé ici le mécanisme de réflexion qui permet de trouver tous les membres d'une classe automatiquement par la méthode `getDeclaredFields`.

4.3 Handler

L'application Android limite l'accès à l'interface graphique, toutes les modifications se font directement dans le Thread UI. On utilise donc le mécanisme de Looper / Handler pour gérer toutes les demandes de modifications venant d'autre Thread.

4.4 File d'attente et `ExecutorService`

Le traitement de soumission est quelques choses d'important, car il faut s'assurer que la deuxième soumission n'a pas de points. Pour cela, on utilise une file de travail pour traiter toutes les soumissions. Il existe dans Java une classe `ExecutorService` pour créer un nombre limité de Thread de travail, on a donc utilisé `Executors.newSingleThreadExecutor` pour que les soumissions soient traitées l'une après l'autre.

4.5 `Callable<T>` et `Future<T>`

Le résultat d'analyse du résultat doit être vu par le reste du programme. Généralement on utilise une variable globale pour faire sortir une valeur d'un Thread. Mais une variable ne suffit pas, car nous avons beaucoup de soumissions à traiter et on ne sais pas quand est-ce que la valeur précédent est lue. En alternative, Java propose une interface `Callable<T>` pour retourner une valeur de type `Future<T>`. Cette valeur sera ensuite utilisée pour

4.6 Thread retardé et en boucle

4.7 Thread Client

4.8 Deuxième tour

4.9 Débloquer les joueurs

4.10 Faire sortir un joueur