PROJET INF431 Set Multijoueur

Jiyingmei WANG, Fan JIANG

1. Introduction

1.1. Règle du Jeu Set

Jeu Set est constitué de 81 cartes toutes différentes qui se distinguent selon 4 caractéristiques :

- Nombre : 1, 2 ou 3 objets.
- Couleur : rouge, vert ou bleu,
- Remplissage : vide, hachuré ou plein,
- Forme : ovale, rectangle ou losange.

Un set est un ensemble de trois cartes qui sont soit toutes les trois identiques, soit toutes les trois différentes, pour chacune des caractéristiques prises séparément. 12 ou 15 cartes piochées parmi les 81 cartes sont disponibles sur la table.

Jeu-solo : Une fois le joueur trouve un bon set, elle/il prend les trois cartes et les garde dans sa pile de gain. On complète les cartes sur la table avec trois cartes de la pioche. Le jeu se déroule jusque au moment où il n'y a plus de cartes dans la pioche et il n'y aucun set sur la table. L'objective est de trouver tous les sets le plus rapidement possible en jouant face à la montre.
Le jeu de multi-joueurs : Tout le monde joue en même temps. Le joueur qui trouve un bon set prend les trois cartes et les garde dans sa pile de gain. Le gagnant est celui qui a trouvé le plus de sets.

1.2. Objective

Ce projet consiste à développer une application Android seul-joueur/multi-joueurs du jeu Set y compris la manipulation de l'interface graphique d'Android et la configuration de serveur.

2. Menu principal

2.1. Fonction de Menu

Les joueurs sont laissés décider à jouer « Single Player Game » ou « Multi Player Game » en cliquant sur les boutons correspondants dans menu du jeu. L'interface de jeu va apparaître directement après que les joueurs font leur choix.

2.2. Réalisation de l'Interface du Menu

L'interface est réalisée par la classe **MainActivity.java**, qui contient les méthodes consistant à construire l'interface graphique et à réagir aux choix des joueurs. Les configurations détailles de l'implémentation de UI(User Interface) sont codées

dans le fichier **activity_main.xml.**

3. Single Player Game

Single Player Game est réalisé principalement par la classe **OnePlayer.java**

3.1. Variables Importantes

- **Stack<Integer> cards** :
  La pioche de carte qui dépose 81 cartes au début du jeu.
- **ArrayList<Integer> cardsOnTable** :
  L'ensemble des 12 ou 15 cartes sur table
- **LinkedList<ImageView> cards_selected** :
  Les cartes sélectionnées par joueurs
- **LinkedList<ImageView> cards_toBeUpdate** :
  L'ensemble des cartes sur la table dont le couleur du fond a besoin de mettre à jour.
- **Lock lock** :
  Le verrou qui contrôle LinkedList<ImageView> cards_selected.
- **Condition lessThan3Cards** :
  Variable de condition consistant à vérifier si les joueurs ont bien choisi 3 cartes.

3.2. Application de Concurrence/ Structure de Programmation

OnePlayer.java dispose des threads différents responsables aux travaux différents.

- **Thread UI**
  Le thread UI est le thread principal pour manipuler le UI du jeu lors que l'on appelle la méthode *onCreate*. Elle s'occupe de dessiner une grille 3 x 4 et les 12 ou 15 premières cartes piochées sur la table du jeu, de transformer les clics de joueurs aux éléments ajoutés au *LinkedList<ImageView> cards_selected*, et de changer le couleur du fond de la carte pour marquer qu'elle est choisie. *onCreate* aussi crée deux nouveaux threads : *initCards*, *check*.
- **Thread initCards**
  Le thread *initCards* est un thread dont le mission est de piocher 12 ou 15 carte parmi toutes 81 cartes randomisé au début du jeu.
- **Thread check**
  Le thread *check* est un thread important dans le cas de jeu-solo. Il s'agit de récupérer les cartes dans l'ensemble des *cards_selected* lorsque le nombre de lesquelles atteint 3 et de changer leur couleur de fond lorsqu'elles s'avèrent qu'elles forment un bon set ou non. Dans *check*, le handler *callback* appelle l'objet de Runnable *success_update* ou *fail_update* pour mettre à jour les cartes dans les différents cas.

4. Multi Player Game

Dans le cas de Multi Player Game, le travail se divise en deux parties qui sont fournies respectivement par les portables des joueurs et le serveur, l'ordinateur du développement dans notre cas.

4.1. Programme Exécuté sur Portable

Le programme exécuté sur le portable de joueur ressemble beaucoup au cas de Single Player Game. Les modifications sont précisées comme le suivant :

- **Thread UI**
  Le thread UI fait les mêmes travaux qu'avant. La seule différence est qu'au lieu de *initCards*, la méthod *onCreate* crée un nouveau thread : *fromServer*.
- **Thread check**
  Le thread *check* ne vérifie pas si les trois cartes choisis par le joueur forment un bon set ou pas. En revanche, il envoie les trois choix de joueur au serveur.
- **Thread fromServer**
  Le thread *fromServer* consiste à recevoir et analyser le message envoyé par le serveur, et à modifier l'interface du jeu d'une manière correspondante. Elle commence par créer un socket et essayer de connecter au serveur. Les messages reçus contiennent les 12 cartes initiales lors de la connextion, le résultat de set choisi, le bon set choisi par les opposants, les nouvelles cartes ajoutées et la fin du jeu.

4.2. Programme exécuté sur serveur

4.2.1. Variables Importantes

- **private LinkedList<ClientThread> client** :
  Des threads indépendants qui s'occupent des connexions avec les portables de différents joueurs.
- **HashMap<ClientThread, Integer> scores** :
  L'emplacement de mémoire où les notes de joueurs sont déposées.

4.2.2. Application de Concurrence/ Structure de Programmation

- **Class ServerThread**
  Le travail de ce thread commence par créer un ServeurSocket pour accepter les connexions de portables. Ensuite, il accepte chaque demande de connexion de joueur et crée chacun un *ClientThread* qui s'occupe de communiquer avec ce joueur.
- **Class ClientThread**
  Ce thread est pour communiquer avec un unique joueur distribué par le serveur lorsque ce thread est créé. Les messages envoyés contiennent les 12 cartes initiales, le résultat de set choisi, le bon set choisi par les opposants, les nouvelles cartes ajoutées et la fin du jeu. Les messages reçus sont les sets de trois cartes choisis par le joueur.

4.3. Connexion et Communication entre Serveur et Joueur

- Socket
  Comme nous avons déjà mentionné, la connexion et la communication entre

Serveur et Joueur se font par Socket. Tous les message envoyé/reçu sont gérés par le PrintWriter et le BufferdReader associé à ce socket. Du côté de serveur, il dispose un ServerSocket et des Sockets connecté aux Sockets de joueurs. Du côté de joueurs, ils possèdent leur propre Sockets.

- Protocole

  Le message transmit entre Sockets ne peut être que de format « String ». Donc, pour les faire se comprendre mieux, il est nécessaire de rédiger une grammaire de message. C'est bien le protocole de la communication.

  1. **LOGIN : __,__, … ,__,__**

  Un message à envoyer au joueurs au début du jeu. Les blancs sont remplis par des integers représentant 12 ou 15 cartes piochées.

  2. **SELECT : __,__,__**

  Un message à envoyer au serveur. Les blancs sont remplis par les Integers représentant les trois cartes choisies par le joueur.

  3. **SUCCESS : __,__,__(,__,__,__)**

  Un message à envoyer au joueur quand il a choisi un bon set. Les blancs sont remplis par des integers représentant 3 ou 6 nouveaux cartes à mettre à table dans le nouveau tour.

  4. **FAIL :**

  Un message à envoyer au joueur quand les trois cartes choisies par lui ne forment pas un set.

  5. **OTHERSUCCEEDED : __,__,__;__,__,__(,__,__,__)**

  Un message à envoyer aux joueurs quand son opposant a choisi un bon set. Les premiers trois blancs sont remplis par trois Integers représentant le set choisi par son opposant. Les blancs après le point-virgule représentent des nouvelles cartes comme dans le cas de Success.

  6. **END : You Win !/ You Loose !**

  Un message à envoyer au joueur quand le jeu est terminé qui aussi indique si le joueur gagne ou perd.

5. Conclusion

En réalisant une application Android seul-joueur/multi-joueurs du jeu Set, nous avons eu l'opportunité d'apprendre des nouvelles connaissances sur la manipulation de l'interface graphique d'Android, la communication entre le serveur et les clients, et de mettre en pratique notre connaissance sur la programmation concurrente.