INFO0039 - Projet de Programmation Orientée-Objet I

Partie 5

Pierre-Yves Derbaix et Raphael Javaux

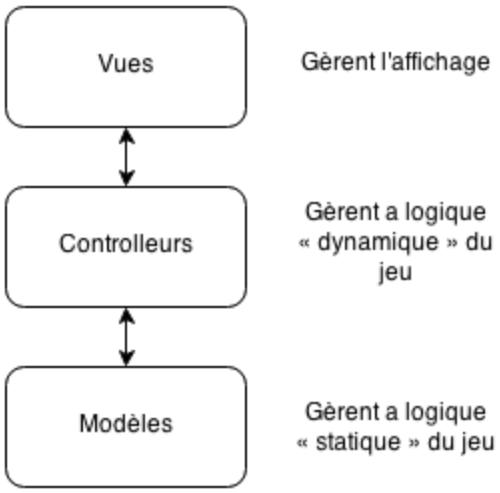
Année accadémique 2013-2014

Introduction

Pour cette dernière partie du projet, nous avons finalisé l'architecture générale de notre application.

Nous avons légèrement dévié du modèle MVC pour atteindre un modèle 3-tiers, plus adapté aux fonctionnalités de jeu en réseau requises par cette dernière phase.

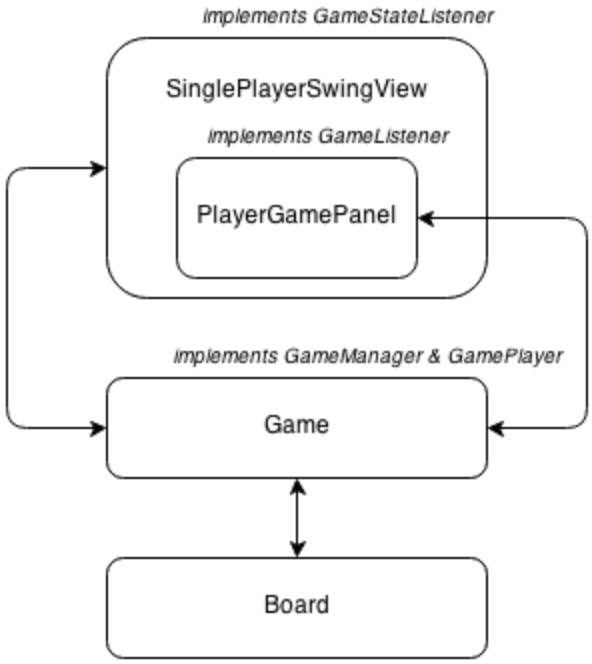
Désormais, les *vues* ne communiquent plus qu'avec le contrôleur, en recevant des *évènements* (changement dans la grille, mise du jeu en pause …) et en déclenchant des *actions* (mouvement de pièce, mise en pause du jeu ...).
Le contrôleur se charge de la partie dynamique du jeu (timers, scores, synchronisation des parties multi-joueurs …) et communique avec les *modèles* (état des grilles, déplacement des pièces …), toujours avec des *événements* (changement dans la grille ...) et des *actions* (mouvement de pièce …).



Nous expliquons le rôle de chacun de ces trois modules pour les trois modes de jeu (jeu à un joueur, jeu multi-joueur ou jeu en ligne) dans les sections qui suivent.

Architecture d'une partie simple joueur

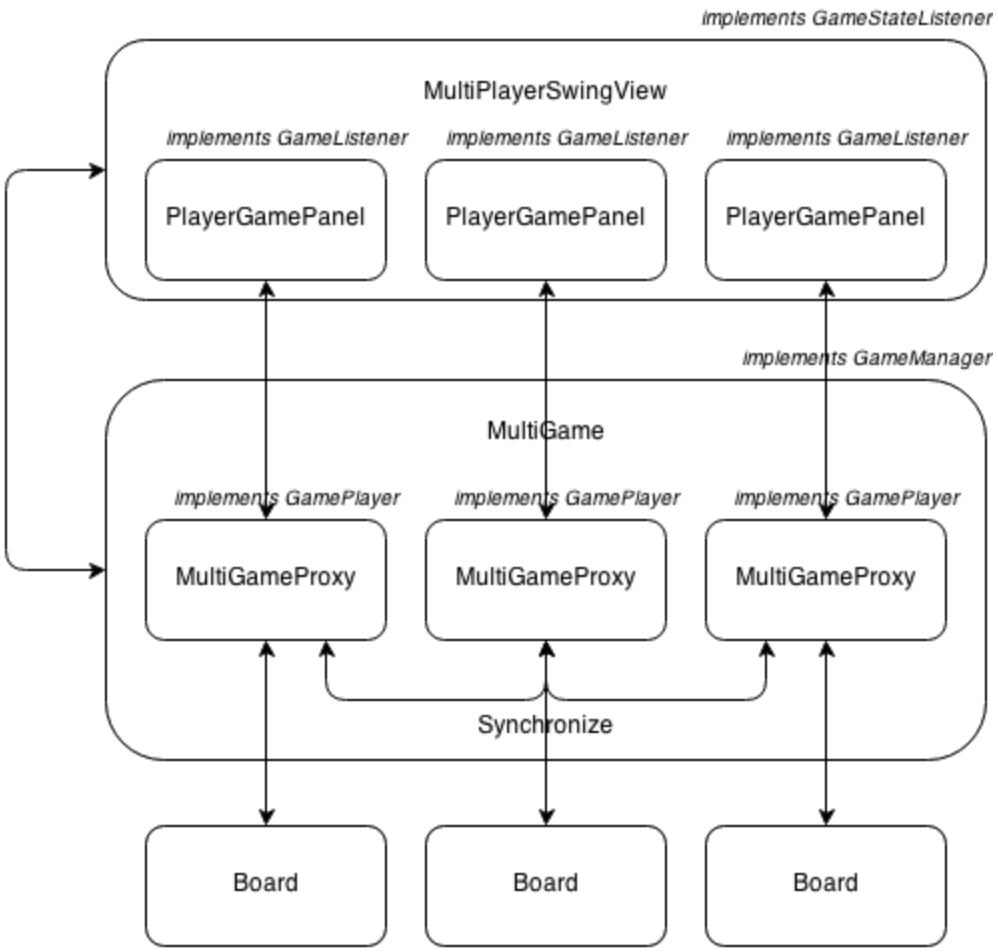
Dans ce mode de jeu, la vue ne contient qu'un seul panneau de jeu (*PlayerGamePanel*). Ce panneau de jeu reçoit du contrôleur (*Game*) les évènements concernant les changements de la grille en implémentant l'interface *GameListener*, alors que la vue reçoit les événements qui concernent l'état du jeu (game over, pauses, évolution du temps de jeu …) en implémentant l'interface *GameStateListener*.
La *SinglePlayerSwingView* interagit avec le contrôleur via l'interface *GameManager* pour tout ce qui concerne l'état du jeu (nouvelle partie …) alors avec le *PlayerGamePanel* interagit avec l'interface *GamePlayer* pour tout ce qui concerne le déplacement des pièces.



Architecture d'une partie multi-joueur

Dans ce mode de jeu, la vue contient plusieurs panneaux de jeu.
Le jeu est géré par un contrôleur principal *MultiGame*. Celui-ci fournit des sous-contrôleur MultiGameProxy pour chaque joueur. Chaque sous-contrôleur est lié à un plateau de jeu (*Board*). Le contrôleur principal a la charge de synchroniser chacun de ces sous-contrôleurs. Chaque panneau de jeu n'interagit qu'avec un seul sous-contrôleur (via l'interface *GamePlayer*) alors que la vue principale (*MultiPlayerSwingView*) interagit avec le contrôleur principal (via l'interface *GameManager*) pour tout ce qui est de la gestion du jeu dans sa globalité (nouvelle partie, pauses, temps de jeu …).

Le schéma suivant montre les interactions dans une partie à trois joueurs :



Architecture d'une partie en ligne

Les interactions lors d'une partie en réseau sont similaires à celles d'une partie multi-joueurs. La différence notable est la présence d'un « proxy réseau » qui va se charger de transmettre les interactions entre les vues et les contrôleurs via une socket TCP.
Comme pour les parties multi-joueurs traditionnelles, un contrôleur principal (*MultiGame*) gère un ensemble de sous-contrôleurs. Cependant, les vues ne vont pas directement communiquer avec ceux-ci.

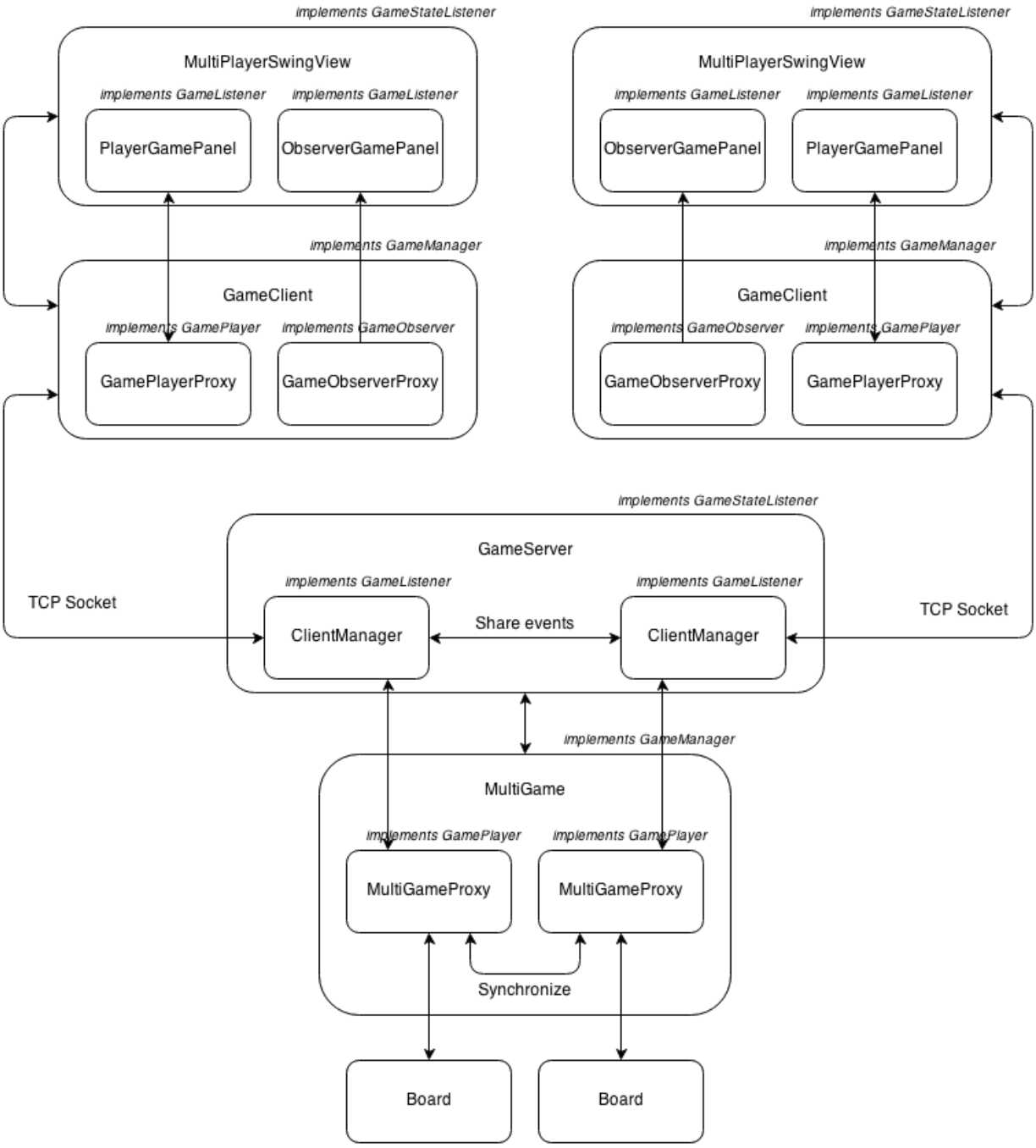
Les évènements des contrôleurs s'exécutant sur le serveur seront « capturés » par l'objet *GameServer* et ses sous-objets *GameManager* (chacun gérant une grille de jeu). Ceux-ci seront en contact, via une socket TCP, avec une instance de *GameClient*, différente pour chaque joueur.

Du coté du client, l'objet *GameClient* apparaîtra comme un contrôleur *GameManager* traditionnel (avec les méthode de gestion du jeu : nouvelle partie, pause, …). Il fournira également aux vues un *GamePlayer* (pour contrôler la grille du joueur) et un ensemble de *GameObserver* (pour observer les grilles des adversaires).

Du point de vue des vues, le fonctionnement d'une partie en réseau est quasiment identique à celle d'une partie multi-joueurs locale, à l'exception près des panneaux des adversaires qui ne sont pas jouables (*GameObserverPanel* à la place de *GamePlayerPanel*).

Le schéma suivant représente les interactions lors d'une partie en ligne à deux joueurs[1] :

[1] Notre implémentation est capable de faire des parties pour un nombre indéfini de joueurs.



Détails d'implémentation

Protocole

Chaque action (du client au serveur) ou évènement (du serveur au client) transmit entre le serveur et le client se fait à l'aide d'un objet du paquetage *network.protocol*.

Chaque classe implémente *Serializable* et est donc transmissible sur le réseau à l'aide d'*ObjectInputStream*s et d'*ObjectOutputStream*s.

Transmission des états des grilles

Lorsqu'un évènement de changement d'état de la grille survient, un objet implémentant l'interface *BoardSection* est créé.
Cette interface définit un ensemble de méthodes pour accéder à une portion de la grille (indice de la portion dans le grille, taille de la portion et une méthode *get(y, x)* pour obtenir le contenu d'une pièce dans la portion).
Dans le cas d'un jeu local, *get(y, x)* va être un appel direct à la grille locale alors que dans une partie en ligne il accédera à une copie locale de l'ensemble des pièces de la section, copie qui aura été effectuée par le *GameClient* et le *GameServer*.
Il n'y a donc aucune copie des cellules de la grille lors d'une partie locale et seules les cellules modifiées sont transmises sur le réseau dans une partie en ligne.

Duplication des logs du serveur

Le *GameServer* accepte dans son constructeur un object *Writer* (de *java.io*) pour y logger les messages transmis.
Afin de pouvoir écrire le contenu du log à la fois dans le fichier de log et dans l'interface graphique observant l'état du serveur, nous avons créé une classe *DuplicateWriter* héritant de *Writer*. En acceptant deux autres *Writer*s en argument de construction, cette classe duplique ses entrées sur ces deux derniers.

Interface graphique

WelcomeView

Deux boutons ont été ajouté au menu principal, à savoir un bouton pour créer un serveur et un bouton pour se connecter à un serveur.
Le client et le serveur forment donc un seul et unique exécutable (que l'on peut lancer avec la commande *ant run*).

Config

Cette classe abstraite fait partie des modèles et correspond à la configuration d'une grille : taille et booléen indiquant si l'on doit utiliser des images pour afficher les pièces.

LocalConfig

LocalConfig hérite de *Config* et contient en plus un nombre de joueurs ainsi qu'une collection de *KeySet* qui correspondent à la configuration clavier de chaque joueur.

OnlineConfig

Comme la classe *OnlineConfig*, la classe *RemoteConfig* hérite de *Config* et contient en plus un mode de jeu. Le mode de jeu est représenté par une valeur de l'énumération *GameMode* interne à la classe *OnlineConfig*.

ServerOptionsView

Cette *JFrame* correspond à la configuration d'un serveur. Elle permet de choisir un fichier où logger les évènements, de choisir le numéro de port, le mode de jeu ainsi que le nombre de joueurs. Cette classe s'instancie à l'aide d'un *OnlineConfig* pour prévenir les clients de la taille de la grille, du mode de jeu, etc.

Une fois le bouton pressé, deux *Writers* sont instanciés, un pour l'écriture dans un fichier et un pour l'affichage dans une fenêtre graphique (*ServerStatusView*). Cette vue est ensuite instanciée et affichée. Le second *Writer* fait donc appel, dans sa méthode write(), à la vue afin quelle affiche les évènements. Un *GameServer* est ensuite instancié à l'aide du numéro de port choisi ainsi qu'un *MultiGame* et de l'*OnlineConfig*. Le *MultiGame* est quant à lui instancié à l'aide de la taille de la grille, du nombre de joueurs et d'un *NintendoGameBoyFactory*.

ServerStatusView

Cette vue comporte une *JTextArea* qui est utilisée pour logger les évènement survenus durant une partie. Elle possède un méthode log() qui est appelée dans la méthode write() du *Writer* cité précédemment.

JoinServerView

Cette dernière vue permet à un client de contacter un serveur en précisant son IP/hostname et son numéro de port. Cette classe est instanciée à l'aide d'une *JFrame* parente et d'un *LocalConfig* qui correspond à la configuration locale du client.

Une fois le bouton pressé, le joueur est mis en attente tant que le nombre de joueurs requis n'est pas atteint. Une fois que c'est le cas, la partie démarre chez tous les joueurs et un *MultiPlayerSwingView* est instancié à l'aide de la JFrame parente et d'un GameClient. La vue est ensuite affichée.