LO02

Etat final de l'appilcation

Mathieu Pellouard, Louis Magnier

Automne

utt
UNIVERSITÉ DE TECHNOLOGIE
TROYES

Sommaire :

1/ Diagrammes de classes au début du projet et actuel

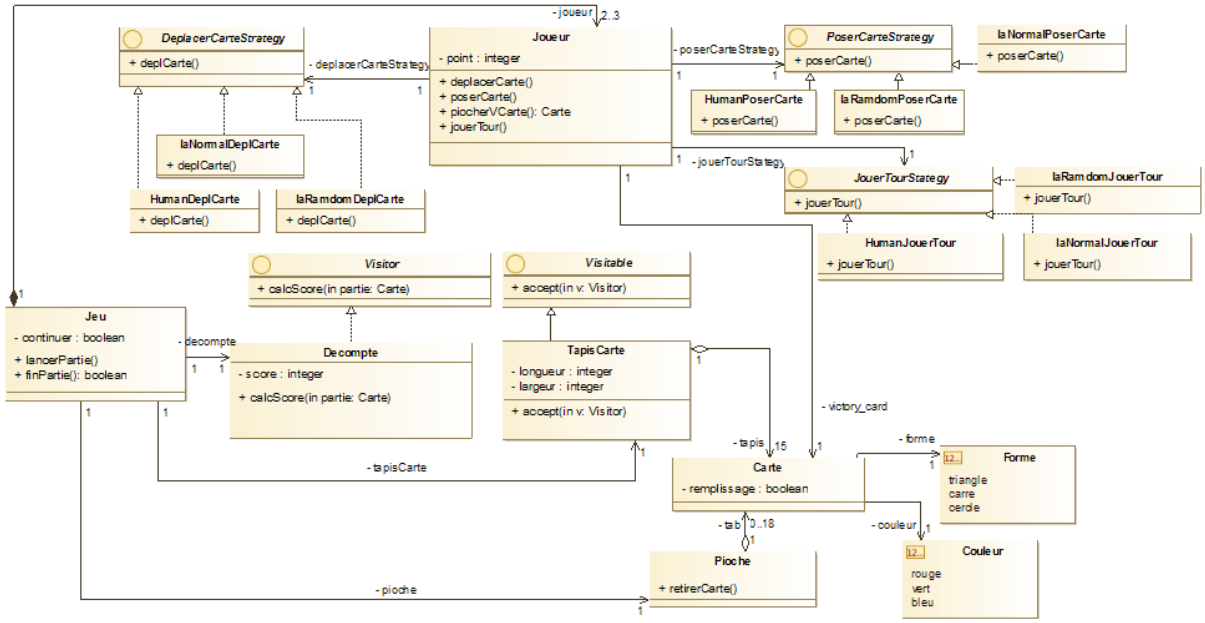

Figure 1 : Diagramme de classe au début du projet

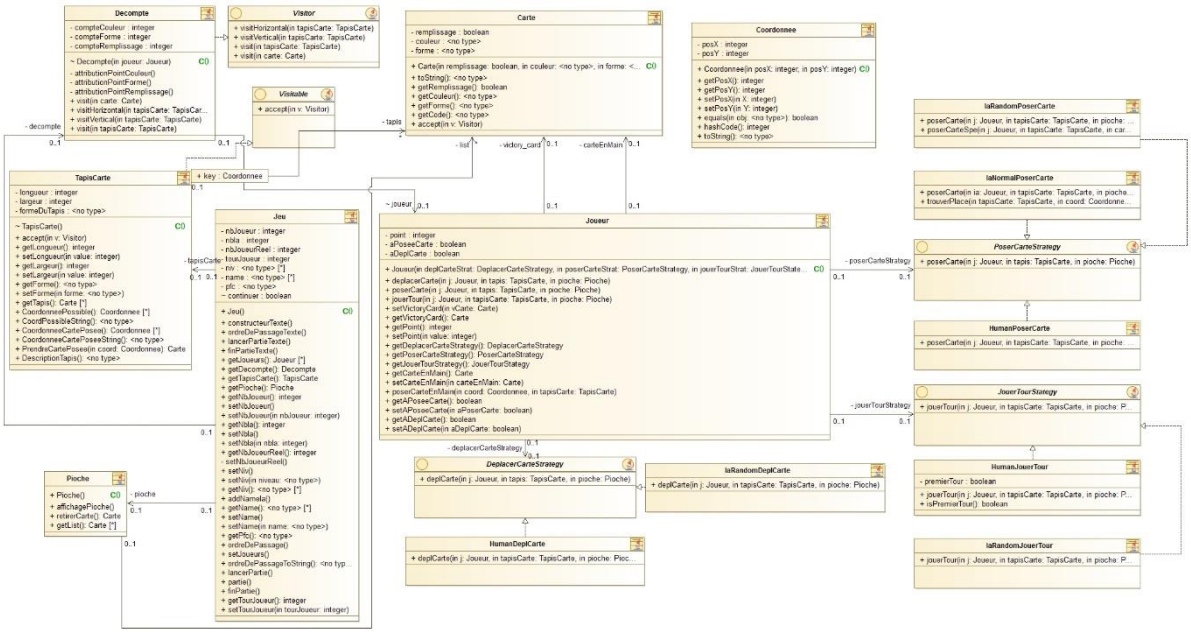

Figure 2 : Diagramme de classe à la fin du projet

2/ Modifications apportées à l'idée de base

Notre idée du projet en fin de ce dernier n'a pas tellement bougé par rapport au début. Le diagramme de classe fait foi et est relativement similaire à ce que nous avions imaginé au début. Cependant nous avons pu apporter certaines modifications en cour de projet qui nous ont permis de nous faciliter la tache et de rentrer dans le cahier des charges. Ces modifications ont été apportée au fil du projet et notamment entre le jalon 2 et la soutenance finale. Nous avions pensé utiliser un simple tableau à double dimension pour représenter notre tapis de carte et ce dernier avait été implémanté de la sorte pour le jalon 2. Cependant nous avons finalement préféré nous rabattre sur une implémentation à l'aide d'une collection en l'occurrence une LinkedHashMap qui nous permet de mettre une carte en argument et une coordonnée en clé. La classe coordonnée quant à elle est une classe que l'on avait pas prévu à l'origine mais qui permet de gérer les coordonnée. Nous avons également eu de gros changement au niveau de la classe Decompte ainsi que de l'implémentation du pattern visitor. Nous avions pour idée de créer une seule méthode permettant le décompte mais nous avons finallement 4 méthodes permettant de bien respecter le patron de conception évoqué plus tôt. Pour ce qui est du pattern strategy, initiallement, nous comptions implémenter deux stratégies de déplacement de carte pour les IAs : une random et une plus réfléchie. Mais nous avons finalement préféré établir qu'une IA de niveau normal pourrait poser des cartes de façon réfléchi et déplacer des cartes de façon random. Nous n'avons donc pas implémenter la classe IaNormalDeplCarte présente dans notre diagramme de classe original. On peut également remarquer une différence importante entre les deux diagrammes de classes sur la page précédente c'est la différence de nombre de méthode. Nous n'aurions pas pensé devoir en faire autant au début et finallement elles sont toutes aussi importante les unes que les autres.

3/ Etât actuel de l'application

Dans cette partie nous allons faire un rapide étât des lieu de notre application.

Pour commencer, nous avons réussi à finaliser le moteur basique du jeu. C'est-à-dire que nous pouvons jouer en tant qu'humain ou alors faire jouer des joueurs virtuels (qu'ils aient une stratégie poussée ou qu'ils jouent de façon random) et cela avec 2 ou 3 joueurs. Notre moteur de jeu est donc totallement jouable en console. Cependant en mode interface graphique, nous n'avons pas réussi à gérer les joueurs humains. De ce fait pour jouer avec l'interface graphique il faudra utiliser uniquement des joueurs virtuels afin de ne pas tomber sur des erreurs. Nous avons pu implanter les différents patrons de conception demandés dans le cahier des charges :

- patron strategy : Pour ce patron nous avons réalisé trois interfaces qui vont régir les tour, les dépots de cartes et les déplacement de carte de chaque joueur, qu'il soit humain ou non. On a donc en tout 3 interfaces et 7 classes pour ce patron. L'implémentation de ce patron très flexible permet de mixer les façons de jouer pour une IA ce qui est vraiment très intéressant en fonction des situations dans le jeu.
- patron visitor : Le patron visitor a nécessité la création de 2 interfaces correspondant au visitor et au visitable. Ce patron est implémenté dans la classe Decompte, TapisCarte et Carte et permet de compter les point carte par carte.
- patron MVC : Le patron MVC quant à lui a été implémenté au niveau de l'interface graphique.

Le code est réparti en 4 package qui sont :

- Modèle : correspond au moteur du jeu en lui-même
- Modèle.shifumi : correspond au moteur du shifumi permettant de définir l'ordre de passage pour 2 joueurs
- Vue : correspond au différentes classes permettant de gérer la Vue texte et l'interface graphique
- Controleur : correspond aux classes permettant l'interaction entre les interfaces et le modèle

Nous n'avons malheureusement pas eu le temps d'implémenter des variantes au jeu que ce soit au niveau de la forme du tapis ou encore au niveau des règles du jeu.