Compte rendu

Guillaume CAU & Tanguy Traina

Lancement du projet :

Pour lancer le projet, il suffit d'exécuter la commande suivante dans un Terminal :

java -jar Hanabi.jar <nombre_joueur>

Organisation du rendu :

Hanabi.jar : Projet compilé.

src: Dossier contenant les sources du projet,

javadoc : Dossier contenant la Javadoc du projet.

Git : github.com/Caucorico/Hanabi

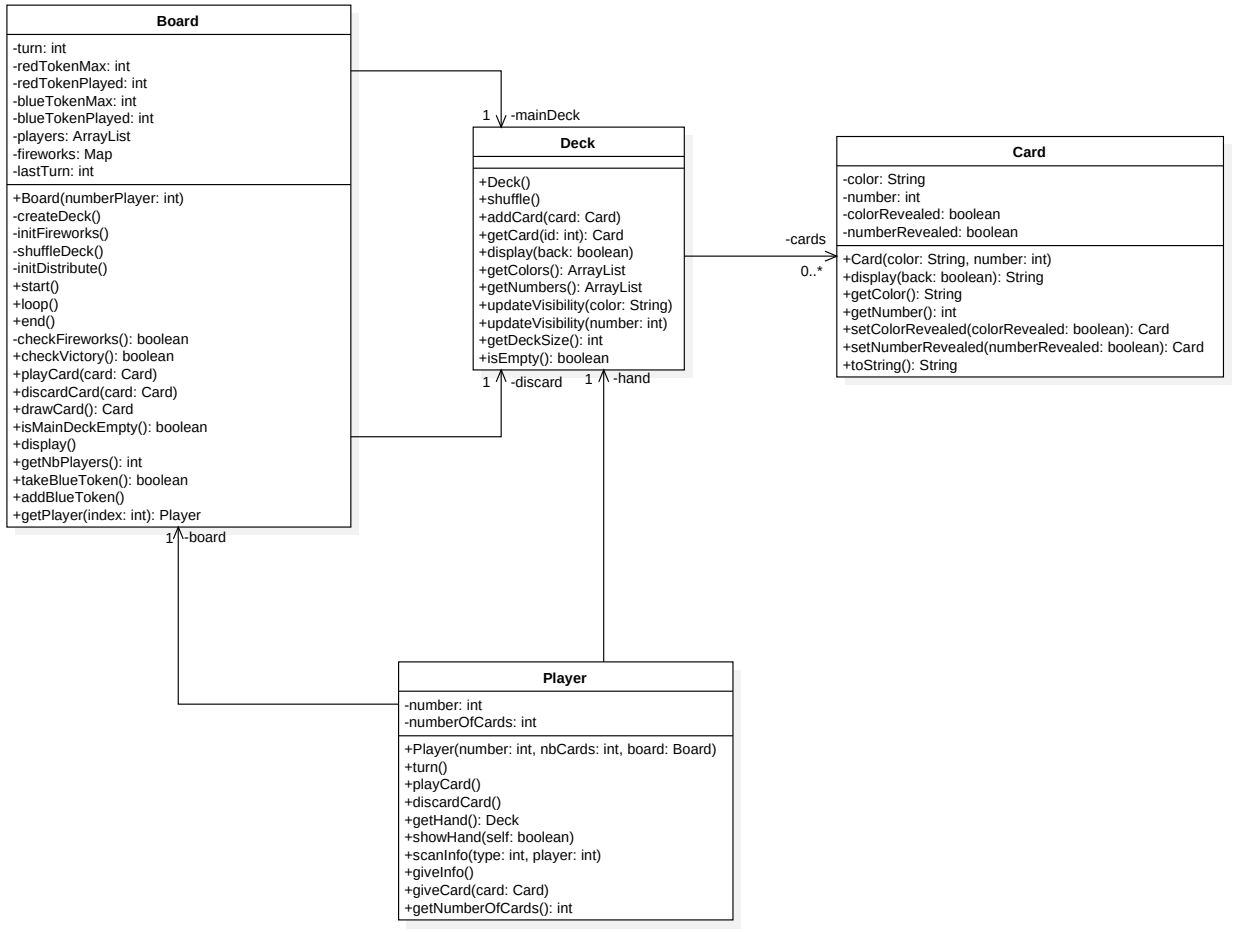
Fonctionnalités implémentées :

- Gestion des feux d'artifices :
  Chaque joueur, avec les cartes qu'il possède, peut jouer une carte. N'importe quelle carte de la main du joueur peut être jouée. Si la carte peut compléter un feu d'artifice, alors elle est ajoutée au feu d'artifice correspondant. Par exemple, si le feu d'artifice est le bleu et que les cartes 1,2 et 3 ont déjà été jouée, la carte bleue 4 est acceptée.
  En revanche, si la carte posée par le joueur ne peut pas être jouée, celle-ci est ajoutée à la défausse.

- Gestion de la main des joueur :
  Chaque joueur à une main, c'est-à-dire les cartes dont il est le responsable. Il ne peut pas voir celles-ci, elle sont masquées tant que le joueur n'a pas reçu d'informations. Il peut faire ce qu'il veut avec, jouer une carte ou défausser une carte.
- Gestion de la défausse :
  Les joueurs peuvent consulter la première carte de la défausse. Toutes les cartes défaussée finissent dans cette défausse.
- Gestion de la pioche :
  Les cartes du jeu sont générées puis mélangées dans la pioche. Ensuite, un nombre déterminé de carte est distribué à chaque joueur. Enfin, chaque joueur peut piocher une carte dans la pioche tant que celle-ci n'est pas vide.

- Gestion des indices
  Chaque joueur peut donner une information à l'un des autres joueurs. Cette information peut être une information de couleur ou une information de chiffre. Si le joueur décide de donner une information de chiffre, alors le chiffre de toutes les cartes du même chiffre sont désormais visibles pour le propriétaire de ces cartes. De la même façon, le joueur peut aussi donner une information de couleur et révéler, à un autre joueur, toutes les cartes d'une même couleur dans sa main.
- Gestion des jetons bleus et rouge :
  Un nombre limité de jetons bleus et rouges sont disponibles dans le jeu. Si un joueur décide de donner une information, alors un jeton bleu est retiré du pot. Si un joueur décide de défausser une carte, alors un jeton bleu est ajouté au pot. Finalement, si un joueur se trompe lorsqu'il joue une carte, un jeton rouge est retiré du pot.
- Fin de partie :
  Lors de la fin de la partie, lorsque la pioche est vide, chaque joueur peut jouer une dernière carte avant le décompte des points.

Organisation du programme :



Nous avons opté pour une organisation particulièrement simple. Chaque élément du jeu possède sa propre classe et ses propres attributs. De cette manière, les fonctionnalités sont réparties là où elles affectent la partie.

Nous avons créé quatre classes en tout :

- Une classe Board qui représente, en quelque sorte, le plateau. Il contient tous les paquets de cartes et les joueurs. C’est aussi elle qui gère les tours entre les différents joueurs.
- Une classe Deck qui représente un ensemble de cartes. C’est dans cette classe que les cartes peuvent être mélangées, piochées et défaussées.
- Une classe Player qui représente un joueur et les cartes qu’il a en main.
- Une classe Card qui représente une carte ainsi que ses caractéristiques.

Choix techniques

Lors du début du projet, nous avons choisi de modéliser notre application avec le modèle UML (seulement diagramme de classe). De cette façon, la répartition des tâches a été plus aisé et nous n'avons quasiment pas rencontré de problèmes lors du développement de la phase 1 et de la phase 2.

Les problèmes qu'on a rencontrés sont liés au modèle MVC. Nous n'avons pas réussi à organiser notre application convenablement avec cette méthode. Les contrôleurs finissaient toujours par devenir statique. (Voir branche feature/struct-mvc sur le git)
De plus, pour développer l'interface graphique avec cette méthode, notre solution demandait de faire de la programmation événementielle, ce que nous n'avons pas eu le temps de faire.

Conclusion

Ce projet nous a permis de mettre en pratique presque toutes les notions que nous avons vues. En ajoutant un contexte dans l'utilisation de toutes ces notions, nous comprenons plus en détail l'intérêt de les utiliser que ce soit en lisibilité du code, rapidité ou concision.

Ce projet nous aide donc à construire un socle solide nous permettant de poursuivre notre formation en deuxième année avec plus de facilités.