CRETE Jonathan - DINDANE CHAHINAZ

PROJET HANABI

ANNEE 2018-2019
INFO-1

Sommaire:

CRETE Jonathan - DINDANE CHAHINAZ

Présentation:

Dans le cadre de notre projet de Java, on avait pour mission de coder un jeu de carte: **Hanabi**.

Hanabi est un jeu coopératif dont l'objectif est de réaliser le plus beau feu d'artifice possible. Le jeu contient 3 jetons rouges, 8 jetons bleus, 60 cartes de différentes couleurs(rouge, bleu, vert, jaune, blanc et une extension multi-couleurs).

- Le principe du jeu est tel que suite:

Le joueur qui possède les vêtements les plus colorés commence la partie. Les joueurs jouent ensuite à tour de rôle, dans le sens des aiguilles d'une montre. Quand vient son tour, un joueur doit accomplir une et une seule des trois actions suivantes (passer son tour n'est pas autorisé):

1/ Donner un indice: Pour accomplir cette action, le joueur doit retirer un jeton bleu du couvercle de la boîte (il le place à côté, avec les jetons rouges). Il peut alors donner une information à un coéquipier sur les cartes que celui-ci tient en main. On peut donner de type d'indices: nombre ou couleur. Le joueur indique clairement - en les montrant du doigt - où se trouvent les cartes sur lesquelles il donne une information.

2/ Défausser une carte: Accomplir cette action permet de remettre un jeton bleu dans le couvercle de la boîte. Le joueur défausse une carte de sa main et la place dans la défausse (à côté de la boîte, face visible). Il pioche ensuite une nouvelle carte, sans la regarder et l'ajoute à sa main. Si tous les jetons bleus sont dans le couvercle de la boîte, cette action ne peut pas être effectuée. Le joueur doit obligatoirement en effectuer une autre.

3/Jouer une carte: Le joueur prend une carte de sa main et la pose devant lui.Deux cas de figure se présentent alors : • Soit la carte commence ou complète un feu d'artifice : elle est alors ajoutée à ce feu d'artifice.• Soit la carte ne vient compléter aucun feu d'artifice : elle est alors défaussée et un jeton rouge est ajouté dans le couvercle de la boîte.Il pioche ensuite une nouvelle carte, sans la regarder et l'ajoute à sa main.

A savoir, il ne peut y avoir qu'un seul feu d'artifice de chaque couleur:

• Les cartes d'un feu d'artifice doivent être posées dans l'ordre croissant (1, puis 2, puis 3, puis 4 et enfin 5).

• Dans chaque feu d'artifice, il ne peut y avoir qu'une seule carte de chaque valeur (5 cartes au total donc).

Une fin de partie d'Hanabi a lieu :

• Si le troisième jeton rouge est placé dans le couvercle de la boîte, la partie prend fin immédiatement et elle est perdue

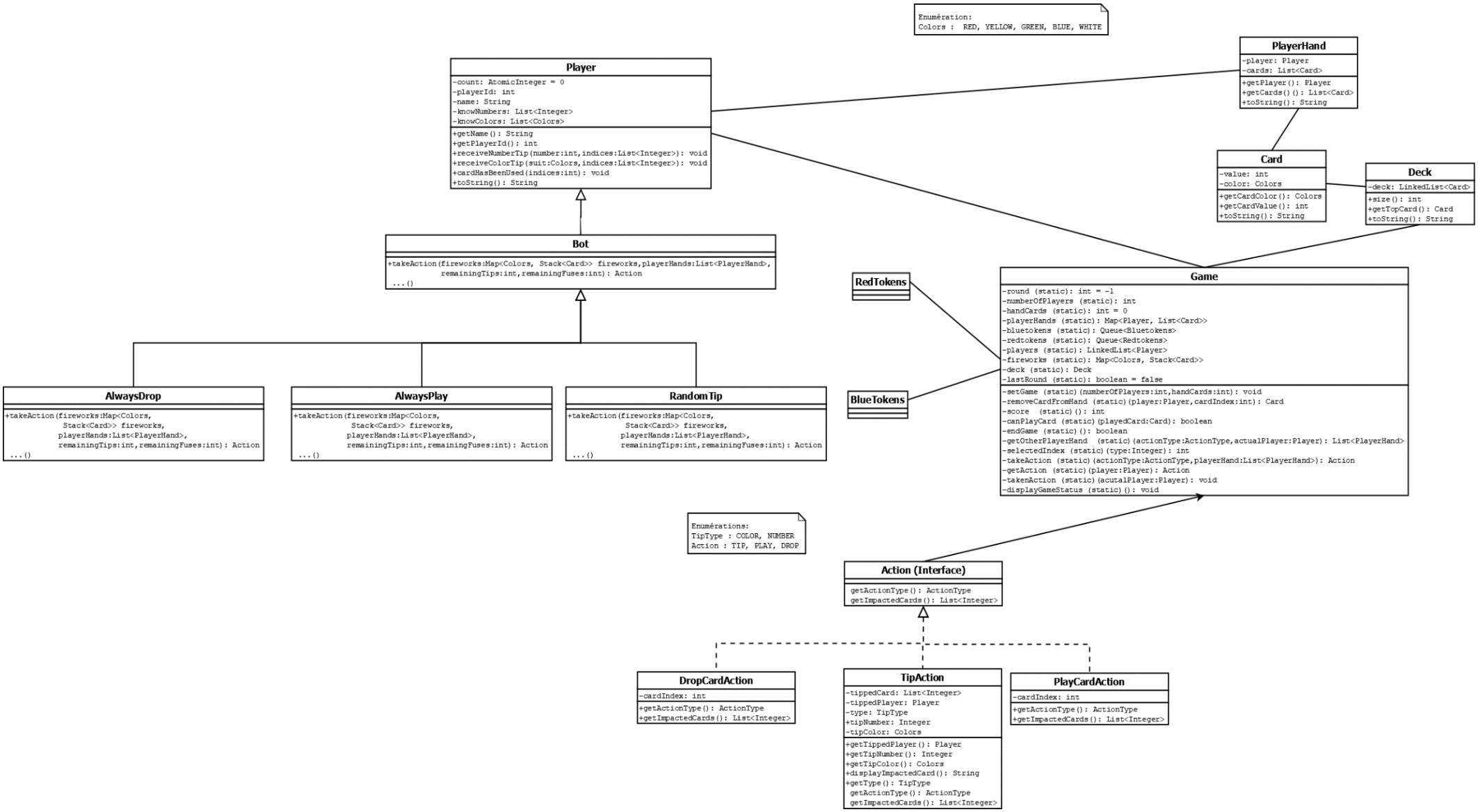
.• Si les artificiers sont parvenus à compléter les 5 feux d'artifice avant la fin de la pioche, le spectacle prend fin immédiatement et c'est une victoire éclatante. Les joueurs obtiennent alors le score maximal de 25 points.

• Si un artificier pioche la dernière carte de la pioche, la partie touche à sa fin : chaque joueur va jouer une dernière fois, celui qui a pioché la dernière carte compris. Durant ce dernier tour de jeu, les joueurs ne pourront pas piocher de cartes pour compléter leur main (la pioche étant vide).

Pour calculer leur score, les joueurs font la somme de la carte de plus haute valeur de chacun des 5 feux d'artifice. Leur performance artistique est évaluée grâce à l'échelle de référence de la Fédération Internationale des Artificiers :

Points	Qualité de la prestation
5 OU -	Horrible, huées de la foule...
6 - 10	Médiocre, à peine quelques applaudissements.
11 - 15	Honorable, mais ne restera pas dans les mémoires...
16 - 20	Excellente, ravit la foule.
21 - 24	Extraordinaire, restera gravée dans les mémoires !
25	Légendaire, petits et grands sans voix, des étoiles dans les yeux.

CRETE Jonathan - DINDANE CHAHINAZ



Implémentation (voir fichier *Diag_hanabi.png*)

Organisation du programme et choix techniques:

Le développement du jeu a été organisé en trois parties:

1/**Classes d'actions**, permettent de créer **des actions selon leur type**. Toutes les classes actions implémentent l'interface *Action* qui oblige chaque classe à override son type d'action et les cartes impactés. Les classes d'actions sont:

TipAction	Permet de créer un un *indice* soit de couleur soit de nombre grâce à ***TipType***. Pour cela, notre classe a besoin de connaître le joueur à qui on donne l'indice, le type de l'indice et le nombre de cartes impactés.
PlayAction	Permet de créer une action de type play. Le joueur choisit une carte (par index) et la dépose sur le plateau de jeu.
DropAction	Permet de créer une action de type *discard*. Permet au joueur de se défausser de la carte choisis (par index).

2/ **Classes bots**:

Bot	Classe abstraite qui permet de créer un bot, à partir du modèle player.
AlwaysDrop	Classe qui fait en sorte qu'un joueur défausser toujours une carte, sans pouvoir jouer ou donner un indice
AlwaysPlay	Classe qui permet au bot de toujours jouer sans pouvoir défausser une carte ou donner un indice.
RandomTip	Classe qui permet de donner un indice sur le nombre ou la couleur d'un joueur..

3/ **Classes principales** dans le package controller:

Colors	Enumération, qui enum les différentes couleurs des cartes.
BlueTokens	Classe vide, permet modéliser les jetons bleu pour les *Tips*.
RedTokens	Classe vide, permet de modéliser les vies des joueurs.
Card	Classe de carte, contient comme champs une valeur qui représente le numéro et une autre la carte la couleur.
Deck	Créer un deck de 50 cartes (avec 3 cartes #1, deux cartes de #2, #3 et #4 et une seule carte #5; et cela pour chaque couleur) puis on le mélange.
Player	Classe qui crée un joueur avec un identifiant qui s'incrémente automatiquement (**AtomicInteger**), un nom, et deux listes de numéro connues et couleurs connues.
PlayerHand	Classe qui pour chaque joueur, return sa main (liste de cartes).
Game	Classe principale qui contient l'algorithme du jeu. Cette classe contient une fonction qui crée les joueurs, leur mains et met en place les feux d'artifices (**setGame**). Une fonction qui calcule le score des joueurs s'ils ont complétés le jeu (**score**). une fonction qui renvoie true si le jeu est terminé (**endGame**).

	Fonction principale du jeu **takenAction** qui permet selon le choix entré par l'utilisateur, faire une action.

Dès le départ, nous avons souhaité faire du code simple et pour cela on a du beaucoup nous documenter sur les choix à prendre, comme par exemple, quand utiliser **Queue**, **Stack** et **List** (Array/Linked/Normal).

Nous avons, avant de nous lancer dans le projet, réaliser un diagramme de classe afin de décider du type, des champs et méthodes à utiliser pour chaque classe.

Pour connaître notre avancement, le projet est aussi disponible sur github ici.

Tout le projet a été réalisé en work-in-pair en binôme sur le même ordinateur (ou presque).

Possibilités d'améliorations:

Faute de temps, nous n'avons pas pu implémenter la partie graphique. On aurait certes souhaité lancer l'application via **zen5** et utiliser la bibliothèque <**.axt.***> pour finaliser tout cela (Bien que nous ayons scannés une à une les cartes du jeu pour cette partie).

En revanche, nous avons implémentés des "semi-bot", qui réalise les actions souhaités (**AlwaysPlay** qui joue toujours une carte, **AlwaysDrop** qui se défausse d'une carte et **RandomTip** qui une fois sur deux donne un tip sur la couleur ou le numéro de la carte).
Le joueur aura ensuite juste la main pour le choisir des index des cartes ou le bot à qui donner un tip.

On peut voir la version bot comme une version simplifiée qui aide les novices à mieux comprendre le jeu.

Problèmes rencontrés:

Au cours de notre projet, le plus difficile était de trouver du temps pour pouvoir travailler ensemble, mais grâce aux efforts de chacun, nous avons pu correctement avancer sur notre projet, tout en étant d'accord sur toutes les implémentations faites.
Bien sûr, chacun a dû produire de la javadoc de son côté pour un gain de temps et de compréhension.

L'autre difficultée rencontrée, était le choix d'implémentation de listes, queues et piles (useful link).
Sur certains objets du jeu, le choix était naturel (Map pour les *fireworks* par exemple) alors que sur d'autres on devait y réfléchir plus longuement, et voir comment changer d'implémentation au cours du développement (cas de **PlayerHand** qui était implémenté comme un champ de Player et représenté sous forme de **List<Card>** et qui au final est devenue une classe à elle seule, car plus facile d'utilisation pour les implémenter les **Tips**(la différence est visible entre la phase une et la phase deux du projet).

Implémenter le coeur du projet (**Actions** dans **Game**) était une difficulté, car il fallait implémenter une nouvelle structure au lieu d'utiliser de simple 'Integers' comme dans la phase une .

Conclusion:

Le projet était vraiment intéressant et amusant à faire, on aurait juste souhaité plus de temps pour implémenter la partie graphique.

Grâce à toute la documentation qu'on a dû faire, nous avons compris plusieurs principes fondamentaux de la programmation orienté objet, en JAVA.

Le travail en binôme nous force à faire du teamwork, c'est à dire de produire du code propre et compréhensible, afin de pouvoir compter les uns sur les autres.