Rapport pour le projet Kuba en JAVA

Ragousandirane RADJASANDIRANE

Master BI-IPFB
Année Universitaire 2020-2021

Pour ce projet, il y a au total 13 fichiers à compiler. On trouve des classes et des énumérations que l'on va rapidement présenter. Nous analyserons quelques méthodes non triviales ainsi que l'idée derrière certains algorithmes / approches.

1 Présentation des fichiers

1.1 codeErreur.java

Ce fichier comporte simplement des codes erreurs qui vont permettre de décrire certaines situations plus simplement. La plupart des codes erreurs ont des messages qui en dit plus sur l'erreur en question.

1.2 Contenu.java

Nous avons ici le contenu de chaque case de notre plateau. Chaque case est donc constituée d'une bille Blanche, Noire, ROUGE ou pas de bille (null). Il y a un champ **int team** qui est là pour faciliter la manipulation entre bille adverse (il y a probablement une meilleure façon).

1.3 Case.java

On créer un objet **Case** qui sera l'objet principal de notre plateau. Chaque case aura donc un champ de type **Contenu** qui va contenir une **bille**.

1.4 Plateau.java

La classe principale de ce programme, celle qui contient l'encodage du plateau. Nous avons une matrice principale qui sera le plateau où les coups seront joués, et 4 autres plateaux secondaires. Nous en verrons l'utilité plus tard. On retrouve également comme champs les dimensions du plateau, et un int pour suivre le nombre de coup.

1.4.1 init()

Cette première fonction va créer un objet **Case** dans chaque case de notre plateau afin d'y placer une bille. On initialise par la même occasion les autres plateaux secondaires en copiant le plateau principal.

1.4.2 placerBilles()

On va ensuite placer les billes en 3 parties. On s'occupe d'abord des 4 coins du plateaux avec la méthode **carre()**. On va ensuite placer les billes rouges en 2 parties, d'abord le carré de dimensions 3x3 au milieu du plateau, puis, les 4 billes restantes, qui ne sont pas dans le carré 3x3, manuellement.

1.4.3 horsPlateau()

Une méthode surchargée afin de vérifier si la case actuelle est hors plateau ou non. Si on lui donne une direction elle fera la vérification pour la case suivante en suivant la direction donnée.

1.4.4 mouvementCheck()

Une méthode importante qui détermine si un coup est réglementaire ou non selon les règles du jeu. Cette méthode retourne plusieurs codes erreurs :

- **PAS DE BILLE** : La position sélectionnée ne contient pas de bille pour être déplacé
- **HORS TAB** : La position est hors du plateau
- **CAMP ADV** :La position correspond à une bille adverse
- **OPPOSE** : Le point de poussé de la bille n'est pas libre, cette bille ne peut bouger
- **PAS LIBRE** : On essaie de bouger une bille vers une position qu'une bille occupe déjà, le mouvement reste possible, on retourne ce code erreur afin de pouvoir procéder au décalage de la rangée de bille avec qui elle va entrer en contact avec la méthode **pousserBille**
- **OK** : Pas de problème rencontré, on peut bouger la bille

1.4.5 pousserBille()

Cette méthode récursive va permettre de décaler toute une rangée de bille. Nous avons un **cas de base** qui permet de vérifier si la prochaine case est **hors du plateau** ou bien est une **case vide**, on peut dans les deux cas commencer à pousser les billes. On fait des appels récursifs en avançant selon la direction jusqu'à arriver à un des deux cas.

Si on atteint les **limites du plateau**, alors la dernière bille doit être éjectée. Si cette bille est du même camp que la bille qui pousse la rangée de bille, on retourne un code erreur **MEME CAMP** car il n'est pas possible pour un joueur, selon les règles, d'éjecter sa propre bille.
Sinon, on procède à l'éjection en remplaçant le contenu de la dernière case visitée par **null**. On retourne ensuite un code erreur pour informer le programme de si c'est une bille rouge (**EJECT ROUGE**) ou une bille adverse (**EJECT**) qui a été éjectée.
Le programme va reculer d'une bille et vérifier si la prochaine case est vide ou non. Comme nous venons d'éjecter une bille, la prochaine case est nécessairement null, ce qui nous amène donc au cas de la **case vide**.
Ce cas là peut être atteint sans effectuer d'éjection, on peut tomber sur une case vide en parcourant la chaine de bille et ne pas être en dehors du plateau pour une éjection.
On procède ici uniquement au décalage de toute la rangée de bille.
Pour cela, on déplace la bille de la case actuelle vers la prochaine case selon la direction en remplaçant son contenu par **null**. Le programme va alors encore reculer d'une bille, et va encore retomber sur le deuxième cas, soit, la prochaine case est vide, on procède alors au déplacement et ainsi de suite jusqu'à revenir à la bille initiale, naturellement grâce à la récursion, qu'on va également déplacer.

1.4.6 codeConfig()

Cette méthode permet de vérifier la règle qui stipule qu'il ne faut pas reproduire la configuration du tour précédent. C'est ici que l'on va se servir de nos plateaux secondaires.

Voici un schéma qui permet de comprendre les liens entre ces plateaux secondaires :

Configuration initiale : p2
1er coup Blanc : T1;T2;p3;p
2e coup Noir :
3e coup Blanc :
4e coup Noir :

Configuration initiale : p2
1er coup Blanc : T1
2e coup Noir :T2;p3;p
3e coup Blanc :
4e coup Noir :

Configuration initiale : p2
1er coup Blanc : T1
2e coup Noir :T2
3e coup Blanc : p3;p
4e coup Noir :

Configuration initiale : p2
1er coup Blanc : T1
2e coup Noir :T2
3e coup Blanc : p3
4e coup Noir : p*

Chaque plateau va enregistrer un coup afin de sauvegarder un tour entier. Les 3 premiers coups vont servir à placer correctement ces plateaux. Ensuite, on pourra pour tous les coups suivants utiliser ces plateaux pour vérifier la règle. Le fonctionnement est simple, pour le 4e coup du Noir (**p***), le coup est provisoire tant qu'il n'y a pas confirmation que la configuration qui sera produite par ce coup n'est pas similaire à la configuration du plateau **p2**. La similitude entre ces deux configuration est vérifiée par un simple parcours des deux plateaux. Si ces plateaux sont différents, alors le coup est accepté et **p*** devient **p**. On doit ainsi procéder au déplacement de tous les plateaux d'un pas.
Sinon, le coup est refusé est on revient en arrière en affectant **p3** à **p**.
Tous les plateaux excepté **p2** ont pour but de donner leurs places au plateau qui est juste derrière. Les plateaux **T1 et T2 (Transition)** sont là uniquement dans le but de permettre à **p2** d'être toujours attaché au reste des plateaux malgré la distance (deux tours) qui le sépare de **p3**. Voici le schéma pour le coup suivant :

Configuration initiale : -
1er coup Blanc : p2
2e coup Noir :T1
3e coup Blanc : T2
4e coup Noir : p3
5e coup Blanc : p*

Et ainsi de suite, on va pouvoir voir si le coup X produit la même configuration que le coup X-4, peut importe le joueur, et dans ce cas, on interdit le coup.

1.4.7 bougerBille()

Cette méthode va permettre à une bille de bouger si toutes les règles sont respectées.
Si le coup est réglementaire, il reste à vérifier si le coup X ne produit pas la même configuration que le coup X-4.
Si oui, on retourne un code erreur (**COORDS**) qui signifie que les coordonnées doivent être renouvelées.
Si la configuration est différente, alors on retourne le code erreur donné par la méthode **pousserBille** si la bille a poussé une rangée de bille, sinon, on se contente d'afficher le plateau pour finalement retourner un code erreur **OK** pour dire que tout s'est bien passé.

1.5 Direction.java

Cette énumération contient les 4 directions ainsi que des valeurs **di et dj** qui vont incrémenter la position **i et j** selon la direction. Chaque direction contient aussi un indice qui est donc leur indice dans le tableau **dirs** qui est là pour faciliter les opérations sur les directions.
Ainsi que le tableau **dirsOpp** qui donne la direction opposée pour chaque direction du tableau **dirs**.

1.6 Coords.java

Un objet qui gère les coordonnées avec la direction.

1.7 Interaction.java

Une classe qui permet les interactions entre le Joueur ou Bot et le programme pour le choix de la bille à bouger et la direction. Ces choix sont gérés par la souris pour le Joueur.

1.8 Joueur.java

Cette classe est destinée aux Joueurs/Bot. Nous avons des champs qui décrivent le camp, le nombre de capture de billes rouges et adverses afin de déterminer si un Joueur est gagnant ou non.
Un joueur gagne si il capture 7 billes rouges ou toutes les billes adverses, soit 8. Un champ nom juste pour différencier le Bot du Joueur car ils n'ont pas les mêmes opérations. Si le joueur a comme nom "Bot", les coordonnées et la direction seront donné de manière aléatoire.

La méthode principale est **coup()** qui va permettre à un joueur d'effectuer un coup.
Tant que la position donnée n'est pas réglementaire ou incorrecte, on répète les demandes.
Si nous avons une éjection d'une bille adverse ou rouge, il est possible que le Joueur gagne, on vérifie cela et si c'est pas le cas, le Joueur peut rejouer.
S'il s'agit d'un Bot, le choix est random, sinon, le choix se fait via des boutons "Oui" "Non" qui apparaissent sur la fenêtre en demandant au Joueur si il veut rejouer ou non. Pour le Bot, on ajoute un temps de 500 ms de délai pour pas que son coup soit instantané.

1.9 Vue.java

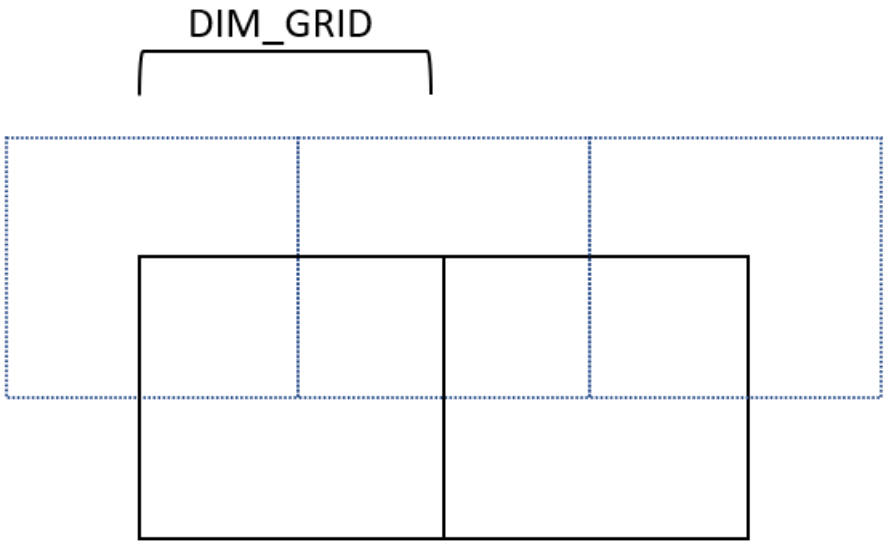
Cette classe a servi avant que l'interface graphique soit opérationnelle. Elle est désormais inutile mais elle permettait d'afficher simplement les billes du plateau.

1.10 Fenetre.java

Une autre classe phare du programme, l'interface graphique.
Nous avons différents champs pour des stocker les dimensions des objets, des images... Les images utilisées ici sont créer avec **PhotoShop** sauf pour les images des flèches directionnelles, récupérées sur Google

1.10.1 painComponent()

Cette méthode va permettre de dessiner différents objets comme par exemple la grille qui est sur le plateau. Pour cela, nous allons avoir comme point de départ pour la grille les points (33,35) car c'est ce qui donne un résultat correct. J'avais dans un premier temps tracé une grille 7x7 et placé les billes au centre de chaque case de la grille. Puis je me suis rendu compte de mon erreur et du fait qu'il fallait tracer une grille 6x6 et placer les billes sur les intersections. Je me suis servi de mon erreur comme tremplin car j'avais déjà accès à l'emplacement final des



billes, il fallait juste que je trace la grille en passant par les centres des billes, soit par les centres des rectangles de la première grille qui est en pointillés sur ce schéma :

1.10.2 buttonChoix()

Création des boutons "OUI" "NON" pour pouvoir rejouer ou non.

1.10.3 createButtons()

Création des flèches directionnelles et des listeners. Les flèches sont placées autour d'un rectangle invisble, afin d'avoir une certaine régularité dans les positions des flèches.

1.10.4 Coords()

Méthode pour transformer les positions x et y sur la fenêtre en coordonnées entre 0 et 6. Cela a été fait de manière assez empirique en détectant les positions x et y qui correspondaient à la première bille, la deuxième... Les coordonnées retournées sont stockées dans les champs **i et j** qui peuvent être obtenues avec **getCoords()**.

1.10.5 areaText()

Une zone de texte afin d'afficher les messages au cours de la partie qui est dans un rectangle blanc sous le plateau.

1.10.6 win()

Une méthode pour lancer la fenêtre de fin dans le mode **false** avec une méthode **getFin()** qui va appeler la méthode **getFin()** du type **FenetrePop** afin de savoir si l'utilisateur a appuyé sur le bouton ou non.

1.11 FenetrePop

Type de fenêtre pour le début et fin de programme. Ces deux temps sont spécifiés dans le constructeur de la fenêtre par un boolean pour déterminer le mode. True pour afficher la fenêtre de début, False pour afficher la fenêtre de fin. Pour la fenêtre de début, on affiche 4 boutons :

- **BOT** : Pour jouer contre le Bot
- **Joueur** : Pour jouer contre un Joueur
- **Bille Blanche** : Avec une bille blanche
- **Bille Noire** : Avec une bille noire

Chaque bouton est au centre de la fenêtre divisé par 4.
Pour la fenêtre de fin, on affiche un message avec un bouton pour fermer d'abord cette fenêtre, puis la fenêtre principale

1.11.1 getMode()

Méthode pour obtenir le choix de jouer contre un Bot ou un Joueur.

1.11.2 getCamp()

Méthode pour obtenir le choix de jouer avec une bille blanche ou noire.

1.11.3 END()

Permet d'être lancé par le bouton **OK** via le **listener** afin de changer la valeur du champ **FIN** en true, on peut récupérer cette valeur avec **getFin()** qui signera donc la fin du programme.

1.12 Jeu.java

Classe pour lancer le jeu en lui même.

1.12.1 Jouer()

Permet d'afficher la fenêtre du début afin de savoir si le joueur souhaite jouer contre un Bot ou contre un autre Joueur, ainsi que le camp choisi. On crée ainsi les objets correspondant aux choix.

1.12.2 jeu()

On affiche la fenêtre principale et une boucle **while** qui tourne tant qu'un joueur n'est pas déclaré gagnant. On lance le coup de chaque joueur.
En sortant de la boucle, un joueur a donc gagné, on vérifie qui a gagné est on affiche un message ainsi qu'une fenêtre pour permettre au joueur de quitter le programme en cliquant sur le bouton **OK**.

1.13 Main.java

Simple main pour créer un Plateau et lancé la méthode static **Jouer()** pour lancer le jeu.

2 Présentation du programme

2.1 Début

Ce programme est donc une implémentation du jeu de Kuba. Il permet de jouer contre un Bot ou contre un Joueur (ou faire jouer un Bot contre un Bot si on modifie le nom du Joueur 1 par "Bot" du constructeur Joueur() dans la classe Jeu.java). Le programme débute avec une fenêtre qui demande si on veut Jouer contre un Bot ou un Joueur et avec quelle bille. Cette fenêtre sera active tant que l'on clique pas sur la croix, ou sur un choix de chaque ligne.

2.2 Fonctionnalités du jeu

2.2.1 Bouger une bille

Ensuite, le plateau s'affiche avec des flèches directionnelle afin de bouger les billes. On choisit la bille a bougé par le clique de la souris. On peut cliquer dans l'ordre que l'on veut, soit d'abord la bille puis la direction, soit la direction puis la bille, il faut bien sur que le déplacement soit réglementaire. Une zone de texte plus bas permet d'avoir des informations quant à l'état du jeu.

2.2.2 Éjection

Les éjections provoquent l'apparition de boutons "OUI" "NON" en demandant donc au Joueur si il veut rejouer ou non.

2.3 Fin

La fin est détectée si un Joueur capture 7 billes rouges ou toutes les billes adverses.

3 Discussion

Ce projet a été réalisé en monôme en une semaine et demi / 2 semaines. Le code est évidemment améliorable et certaines parties ne sont probablement pas optimales comme par exemple la gestion des fenêtre de début, de fin et la fenêtre principale dans la classe Jeu.java, quand il s'agit de les fermer, il y a peut-être une façon plus élégante de le faire. L'esthétique des fenêtre de début et de fin qui reste assez simple.
Le placement des billes, certaines billes sont placées manuellement, il y a aussi une amélioration possible, tout comme la gestion de la règle sur les configuration où il y a 5 plateaux mis en jeu pour pouvoir vérifier cette règle.

3.1 Conception du projet

Le projet Tablut a facilité la conception de ce projet Kuba, car certains aspects étaient similaires comme la gestion des billes ou du plateau. La partie graphique était assez plaisante à faire car elle demandait un peu de créativité dans la conception des images via Photoshop, le placement des objets graphiques et voir que tout fonctionne et que tout se met en place comme on le souhaitait est satisfaisant même si ce n'est pas parfait.
Les moments difficiles de ce projet étaient certains aspects qui me semblaient pas clairs dans l'énoncé, j'ai donc mis du temps pour d'abord les comprendre parfaitement, et ensuite les implémenter. Je pense à la règle où une bille ne peut être poussée si son point de poussé n'est pas libre. C'est cette nouvelle formulation de la règle qui m'a débloqué pour ce point.

3.2 Apports

Je trouve ce projet beaucoup plus complet que le projet Tablut et plus formateur. J'ai beaucoup appris sur le langage Java notamment sur l'aspect interface graphique. En soit, chaque projet nous apprend des choses sur le langage en question, mais l'interface graphique me semblait pas simple avant de commencer le projet avec seulement le TP Swing comme référence. Finalement, les briques s'emboitent plutôt bien lorsque l'on commence l'interface graphique. Au final, c'était un peu fastidieux par moment, mais pas si difficile que ça.

Merci à vous de m'avoir aider pour certains aspects du projet et d'avoir répondu à mes mails.