Runser Laure
n° 21955060
L2 info groupe 3

Bee Rescue Saga

Projet de POO – 2020/2021

I / Description des classes

a) Généralités

J'ai réalisé un jeu qui ressemble beaucoup au Pet Rescue Saga de King. Il est écrit en java, avec Swing pour l'interface, et selon le schéma model-view-controller.

Pour ce faire, j'ai pris le parti de compartimentaliser le code avec des packages. Au plus haut niveau, on en retrouve 5 :
- main qui contient les deux classes Install et Main pour lancer le jeu
- utils qui devait contenir le parser de niveaux avant que je n'abandonne cette idée
- model qui contient tout le backend du jeu
- view qui contient les interfaces graphiques
- controller qui fait la liaison entre le modèle et la vue choisie.

b) Modèle

Ce package est divisé en trois parties : board, bonus et level.

Board contient les différentes classes qui forment les pièces, toutes héritées d'une classe abstraite Piece. Cela permet de définir pour chacune des méthodes de déletion différentes facilement. Ce package contient aussi Board qui comme son nom l'indique est le plateau : c'est ici que j'ai implémenté les méthodes d'actualisation après la déletion des pièces.
J'ai choisi de représenter le plateau sous la forme d'un tableau de tableaux de Piece. C'est avantageux pour pouvoir accéder à chaque case facilement grace aux index. Au final, je me demande si utiliser une List de tableaux de Piece, où chaque tableau représente une colonne n'aurait pas été plus efficace, notamment pour faire tomber les pièces, ou décaler les colonnes (ce que je n'ai pas réussi à faire avec le tableau de tableaux).
Ce package contient aussi NewPieceBuilder qui a pour seul but de créer et renvoyer une nouvelle Piece selon les conditions données par le niveau. En effet, les niveaux où le nombre de coups est limité ont une quantité de pièces infinie : il en tombe toujours de nouvelles quand on en détruit. Cette classe permet d'éviter d'avoir des pièces vertes dans un niveau uniquement composé de pièces rouge et bleues par exemple.

Bonus contient les différentes classes des bonus. De la même façon que pour les pièces du plateau, je les ai fait hériter d'une classe abstraite Bonus. Je n'ai pas eu le temps de bien implémenter les bonus, notamment pour qu'ils se débloquent au fur et à mesure que l'utilisateur avance dans les niveaux.

Level contient les différents types de niveaux. J'ai fait une classe mère Level qui contient des méthodes générales et j'en ai fait hériter LevelLimitedMoves où il y a un nombre de coups limités, LevelLimitedPieces où il faut détruire toutes les pièces du plateau pour gagner, et LevelLimitedTime où il y aurait une limite de temps (je n'ai pas implémenté cette fonctionnalité). La classe LevelMap contient un tableau de niveaux classés dans l'ordre, ainsi que les scores obtenus à chaque niveau.

Enfin, le package board contient aussi Player, qui représente le joueur. Il contient les méthodes qui permettent de sauvegarder sur le disque les progès du joueur.

c) Controller

Ce package contient les classes qui font la liaison entre le modèle et l'affichage.

LevelMapController est chargé de lancer les niveaux, de déterminer si on peut les jouer, et de la navigation entre les vues.
LevelController permet d'interpréter les coups joués par l'utilisateur et de les répercuter sur le modèle.

Enfin, un sous-package contient les listeners. J'ai choisi d'utiliser une construction que j'ai vue dans le projet Andor's Trail (https://github.com/AndorsTrailRelease) (mais je suis sûre qu'elle existe ailleurs). Il s'agit de faire des classes qui regroupent un grand nombre de listeners qui doivent observer le même objet en même temps. On utilise pour cela la classe ListOfListeners (qui j'ai mise dans le package utils mais qui pourrait être refactorisée ici) pour définir un type général de listener. J'ai trouvé que cette construction permet d'avoir un code plus propre et plus compréhensible.

d) View

View contient tout d'abord les interfaces LevelView, MapView et WelcomeView qui sont ensuite implémentées dans cli et gui. L'interface Vue dont héritent MapView et LevelView permet de naviguer entre les deux plus facilement dans LevelMapController.

Le package cli contient l'implémentation des 3 interfaces pour afficher le programme à l'utilisateur en mode texte. C'est plutôt très moche et peu ergonomique, j'aurais pu essayer de trouver une syntae plus agréable pour jouer, mais j'ai pensé que ce n'était pas la priorité.

La package gui contient l'implémentation de l'affichage en mode graphique. Pour rendre le code plus lisible, j'ai ajouté une classe Block qui étend JButton : cela permet de regrouper en un seul endroit la récupération des icônes pour les pièces du plateau et les bonus. Le code est assez moche, notamment la méthode initIcon(Piece piece) qui récupère les images. J'avais essayé de faire des getter dans chacune des classes Piece et Bonus (je n'ai pas eu le temps de nettoyer le code) mais je n'arrivais pas à récupérer la chaîne de caractères du chemin.

e) utils

Utils contient la classe ListOfListeners qui permet de regrouper les listeners (voir c) Controller). Elle devrait être refactorisée dans le package controller.listeners.

Utils contient aussi un package parser qui devait être responsable de lire les niveaux depuis le disque. Puis je me suis intéressée à la sérialisation et j'ai décidé de l'utiliser pour faire un script d'install qui sérialise les niveaux sur le disque. Je pensais avoir le temps de faire un éditeur de niveaux et je me suis dit qu'il serait plus efficace de sérailiser plutôt que de devoir faire d'autres méthodes pour écrire la même syntaxe sur le disque.
Ce package est maintenant abandonné, je n'ai pas voulu le supprimer pour ne pas perdre la trace de mon travail.

f) main

Le package main contient Main et Install. J'ai du faire ce package pour pouvoir utiliser gradle pour lancer l'une ou l'autre des classes selon les arguments de l'utilisateur.

Install permet de lancer la sérialisation des niveaux sur le disque. Ce n'est pas forcément une bonne idée, je pense que cela pourrait prendre beaucoup de temps si il y avait une quantité de niveaux plus importante.

Main récupère le nom de l'utilisateur, et soit lance une nouvelle partie, soit désérialise la partie commencée sous ce même nom. C'est aussi la classe qui analyse si le joueur a décidé de jouer en mode graphique ou textuel.

J'ai décidé d'utiliser gradle pour gérer l'exécution du jeu. Cela me permet de lancer des tests automatiquement lors de la compilation. Cela m'a sauvé la vie plusieurs fois quand j'ai modifié les méthodes de délétion de façon inapropriée.
Je n'ai pas eu le temps de faire des tests pour les bonus, seulement pour le plateau et les méthodes de suppression des pièces (blocks de couleur, bombes, abeilles, décor).

II / Problèmes rencontrés

J'ai rencontré un certain nombre de problèmes lors de la création du jeu. Tout d'abord, j'ai eu beaucoup de mal à gérer mon temps. Entre le retour du confinement, les cours à distance et les problèmes dans le projet de POO, j'ai eu du mal à boucler le projet.

a) Problème de Scanner

Au moment d'écrire les méthodes pour l'affichage dans la vue textuelle, j'ai eu beaucoup de problèmes avec les Scanners. J'ai appris que fermer un Scanner qui écoutait sur System.in ferme aussi le flux System.in, qui ne peut pas être rouvert après. Cela veut dire que à chaque fois que je rouvrais un Scanner, il n'avait pas accès à l'information que l'utilisateur tapait.
En cherchant la solution à ce problème sur Internet, j'ai aussi appris qu'il n'était pas recommandé d'avoir plusieurs Scanners ouverts en même temps : il aurait fallut n'avoir qu'un seul Scanner et le passer en paramètre des méthodes qui le demandent. Je n'ai pas eu le temps de changer ça.

b) Problèmes de sérialisation

Mon principal problème avec la sérialisation était au moment des tests. J'ai perdu une dizaine d'heures à me demander pourquoi mes changements dans le code n'étaient pas répercutés dans l'interface graphique. Le problème était que je cliquais sur reprendre une partie sauvegardée, et que cette partie avait été sérialisée AVANT les changements. Elle ne les possédait donc pas. J'ai fini par abandonner et par écrire le code rempli de instanceof dans la méthode initIcon() de Block, et je ne me suis rendue compte que trop tard que le problème venait de là.

c) Mauvaise compréhension du sujet

J'avais compris qu'il était possible d'utiliser javaFX pour faire l'interface graphique. Plusieurs professeurs ont dit des choses contraires, mais il s'avère au final que swing été obligatoire. J'ai donc du mettre à la poubelle 15 heures de code et recommencer.

d) Problème d'affichage mineur

Le plateau s'affiche à l'envers : de droite à gauche au lieu de gauche à droite. Cela donne une impression un peu bizarre lorsqu'on joue puisqu'il est contre-intuitif que la « gravité améliorée » pousse les blocks vers la droite.

e) Problème de décalage de colonnes.

Le principal problème du jeu en l'état est le décalage des colonnes. Lorsqu'une colonne est vide, les autres sont sensées se décaler pour combler les trous. En l'occurrence, je me suis rendue compte

que ce n’était pas le cas, et que le programme entre dans une boucle infinie et plante. Je n’ai pas réussi à résoudre ce bug.

III / Améliorations possibles

Il faudrait en premier lieu résoudre le bug des colonnes, puisque le jeu n’est pas vraiment jouable en l’état.

Il faudrait tester de façon plus approfondie les Bonus et la réorganisation du plateau qui suit leur suppression : si je l’avais fait plus tôt, j’aurais peut-être eu le temps de trouver une solution.

Je n’ai pas eu le temps d’implémenter un bot qui puisse jouer le niveau, ni une « IA » qui pourrait donner des conseils à l’utilisateur.

On pourrait ajouter des bonus supplémentaires, et implémenter correcter le ChangeBlockColor (qui transforme tous les blocks d’une couleur en une autre) : pour l’instant l’utilisateur n’a aucun moyen de lui donner la couleur de destination, et il est donc inutilisable.

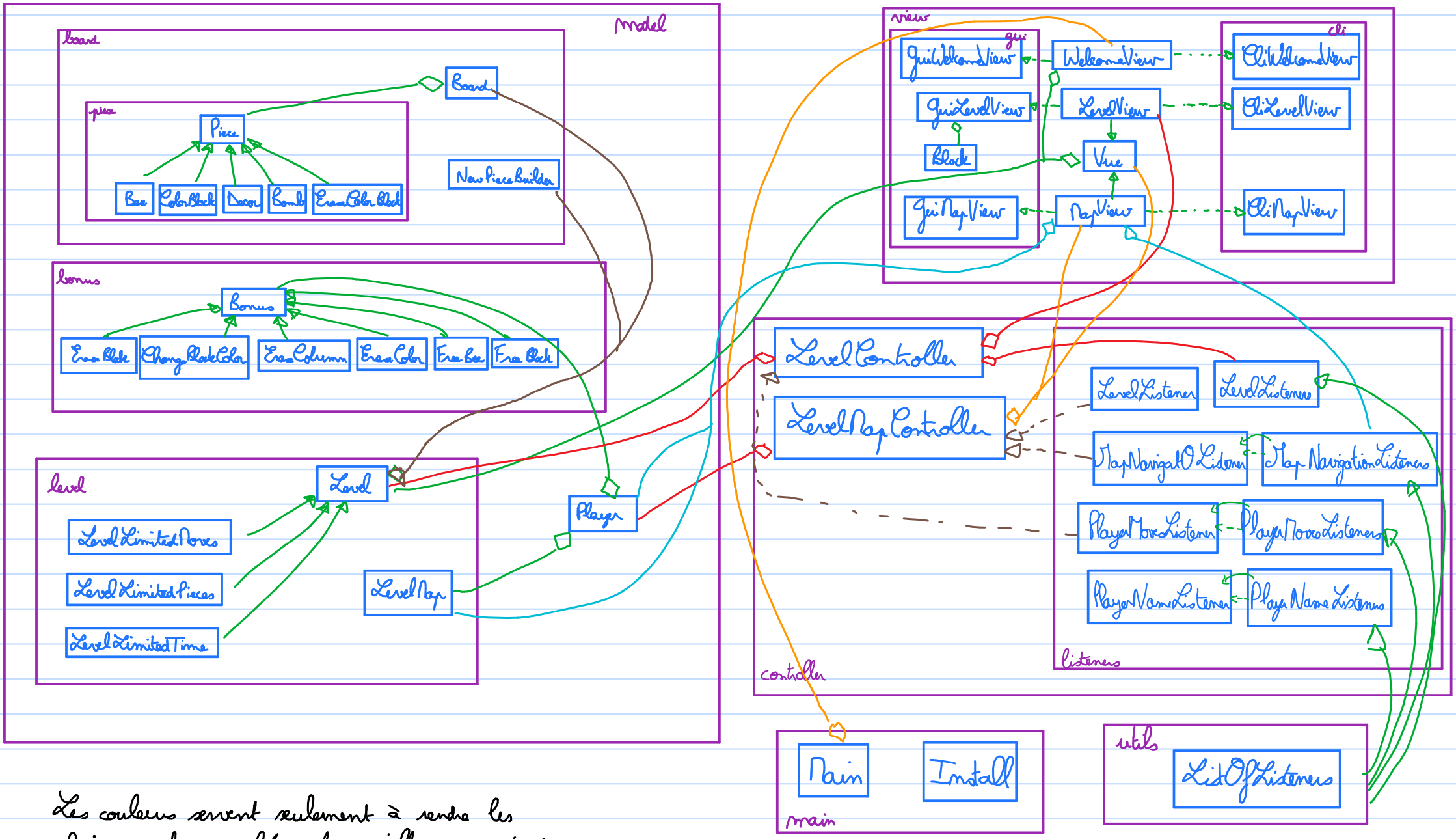
On pourrait implémenter LevelLimitedTime : un niveau où il y a une limite de temps pour libérer toutes les abeilles.

J’avais pour projet d’implémenter une boutique où le joueur pourrait acheter des bonus et des vies, et une lotterie où le joueur pouvait en gagner.

On pourrait empêcher un joueur de lancer une nouvelle partie si le nom qu’il ou elle a choisi est déjà utilisé pour savegarder une partie. En ce moment, si le nouveau joueur décide de sauvegarder, l’autre partie est écrasée.

On pourrait améliorer le design du jeu : les illustrations sont faites à la main avec Gimp et ne sont pas très jolies.

Diagramme des classes - Projet Bee Rescue Saga



Les couleurs servent seulement à rendre les liaisons plus visibles lorsqu'elles se croisent.