Projet Tutoré: SameGame

Introduction

Ce projet tutoré consiste à réaliser le jeu du SameGame avec le langage orienté objet java**.** Ce jeu est composé d'une grille remplie de bloc (des pierres ici). L'objectif est de la vider mais en marquant le plus grand nombre de point possible. Et pour cela il faut donc détruire le plus grand groupe de pierre même couleur réalisable (le minimum étant de deux pierres). Le jeu est terminé lorsque la grille est vide ou qu'il n'y a plus que des pierres isolées.

Sommaire

Juliette Maumené & Arnaud Jorandon

Menu

Le menu est la fenêtre d'accueil du jeu, elle comporte trois boutons.
-Le premier bouton sert pour choisir un grille prédéfinie.
-Le second pour une génération de grille aléatoire.
-Et le bouton quitter. Tous sont gérés par le fichier Check.java.

Gestion des fenêtres (Fenetre.java)

Ce fichier a été le dernier à être écrit, nous chargeons nos fenêtres dans le main et nous avons décidé de simplifier la classe Main et créer un fichier spécialisé dans la création des fenêtres. Nous avons essayé de faire un fenêtre de Règles, mais nous n'avons pas réussi à la remplir de la façon voulue, c'est à dire soit une image soit une zone de texte. Nous avions une fenêtre indépendante qui ne fermait pas le programme.

Constructeur

Le constructeur créer juste une fenêtre qui servira pour les méthodes.

menu(): Méthode qui créer la fenêtre de menu(taille, position, nom..), y ajoute les boutons et les contrôleurs.

fenetre(): Méthode qui créer la fenêtre du jeu, y ajoute la grille en fonction du bouton sélectionné ainsi que le score en bas. Nous ne souhaitions pas que le joueur puisse redimensionner la fenêtre, alors nous l'en avons empêché avec la méthode setResizable().

end(): Dernière méthode à intervenir dans une partie, elle s'occupe de redimensionner la fenêtre, d'afficher le score et de proposer au joueur de revenir au menu ou de quitter à l'aide de deux boutons. Le message de fin est différent selon certaines tranches de score.

Sélection d'un niveau
Le sujet nous demandait d'utiliser un JFileChooser pour choisir une grille prédéfinie.
Le JFileChooser s'ouvre dans une fenêtre à part, et permet de naviguer parmi les dossiers et les fichiers . Nous avons limité le choix du fichier dans le JFileChooser à un seul sous dossier où l'on a mit tout les niveaux. Nous avons fait plusieurs niveaux pour laisser un choix aux joueurs. En cliquant sur le bouton "Open" de la fenêtre, le fichier est stocké dans une variable qui est envoyé au constructeur de la grille. La lecture du fichier se fait dans ce constructeur.
Au début, cliquer sur le bouton cancel nous exposait à une erreur de type "NullPointerException" que nous avons géré en utilisant un try/catch.

Affichage (Grid.java)

Constructeur

Pour éviter la répétition de code on a décidé de charger les images dans un tableau et de les nommer 0,1 et 2 qui correspondent aux items et 4,5,6 pour les même images mais avec un fond gris qui nous servent pour leur “surbrillance”. 3 et 7 correspondent aux images blanches qui effacent les items lors de leur suppression. Plus tard on s’est rendu compte que ne rien mettre était plus esthétique et correspondait à la couleur de fond de la fenêtre.

Les getteurs et les setteurs

Afin de faciliter l’utilisation de nos données, nous avons décidé de créer des méthodes set et get nous permettant d’obtenir et d’échanger de façon claire les différentes données. Nous nous en servons pour le score, la valeur d’une case ou simplement savoir si une case est sélectionnée ou non.

La méthode countSelected()

Cette méthode nous sert à compter à l’aide d’un compteur les cases sélectionnées. Elle nous permet de tester si oui ou non la souris survole un groupe ou une case seule. Elle nous est utile pour ne pas supprimer les cases seules.

La méthode enleverSelection()

Cette méthode sert à réinitialiser la sélection dans la grille, elle est utilisée dans la classe OnClick.

la méthode paintComponent()

Cette méthode sert à afficher graphiquement la grille. Nous l’avons faite en deux partie. La première est l’initialisation de la grille en fonction du mode choisi, car le tableau utilisé ne sera pas le même en fonction du mode. Pour les fichiers, nous utilisons un tableau de caractères. Par la suite nous avons créé une variable inGame, si elle vaut 0, le jeu s’initialise, et si elle vaut 1, le jeu est en cours. Lorsque le jeu est en cours, nous exploitons les tableaux qui changeront grâce aux décalages pour repeindre la grille à chaque clic et survol de la souris en fonction de la valeur comprise dans chaque case.

Algorithme d’identification des groupes

C’est la fonction check_Stone qui réalise cette algorithme. On a décidé d’utiliser la récursivité. On commence par récupérer la position actuelle de la souris dans la classe OnClick.java, et on y applique le premier appel de la fonction dans MouseMoved().

Une fois les positions de la souris donné, on vérifie à à gauche, à droite, en haut et en bas si ces cases ont la même valeur ou non. Et si les conditions sont respectées on rappelle la fonction pour qu'elle vérifie sur cette nouvelle case si il n'y a pas des groupes autour d'elles.

Pour éviter que la récursive soit infinie on utilise un tableau de booléen d'images sélectionnées. On passe la case sélectionnée par le curseur à true, et la fonction récursive passera toutes les cases de même valeur qu'elle a trouvé à true. Ainsi pour chaque rappel de la fonction on doit vérifier si la case est à false dans le tableau de booléen, et dans le tableau de la grille si elle a même valeur que la case initiale ou précédente.

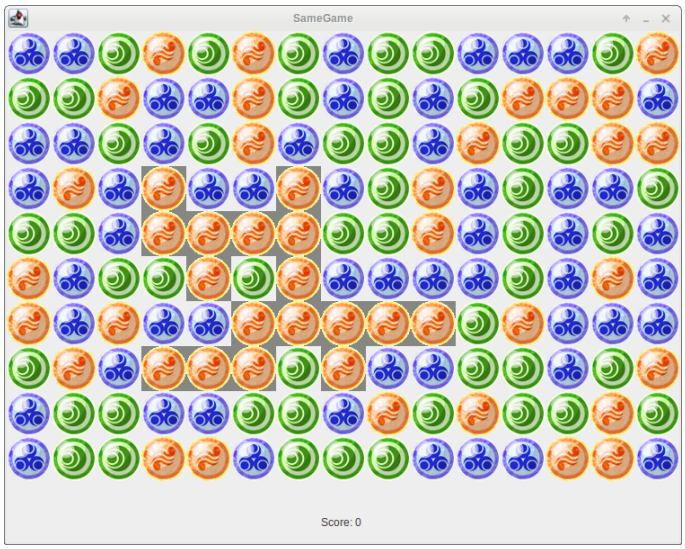
Observateur de la Souris (fichier Onclick.java, Decalage.java)

Ce fichier est un contrôleur des différents mouvement de la souris possible:

- Le passage de la souris sur les images (=surbrillance)
- Le clic

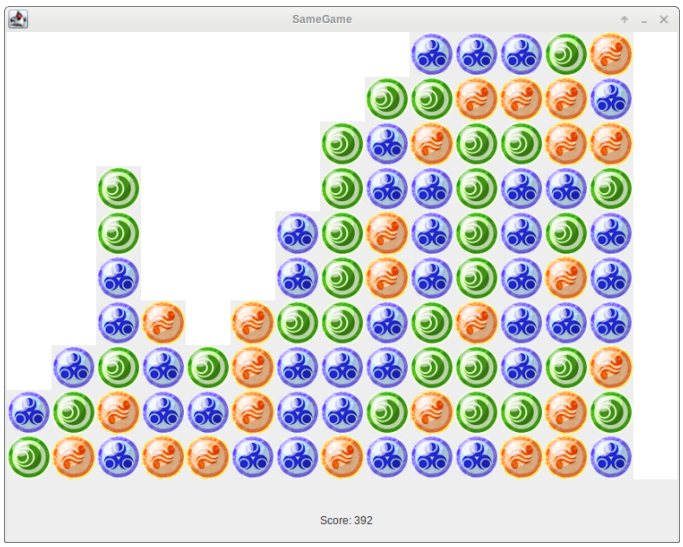
La surbrillance

Pour réaliser la surbrillance, on utilise l'image des items en leur mettant un fond gris. Puis on se sert du tableau initial qui n'a que pour valeur 0(orange), 1(vert) et 2(bleu). On crée une fonction récursive **check_Stone** placée dans Grid.java, qui vérifie si par rapport à la position actuelle de la souris (donnée grâce à MouseMoved) si les pierre située en haut, en bas, à gauche et à droite sont de la même couleur. Si c'est le cas on rappelle la fonction en donnant les nouvelles positions de la pierre de même couleur. ensuite Et grâce à la fonction repaint on affiche la surbrillance. Cependant les valeurs de la souris ne correspondait pas au numéro de l'item dans la tableau pour cela on a décidé de diviser par 50 (la taille de items) la position X et Y de la souris (données par getX et getY). Ainsi la position de la pierre serait la même que celle du tableau, pour la première X=0 et Y=0 et ainsi de suite. Les pierres en surbrillances ont pour valeur dans le tableau 0+4, 1+4 ou 2+4 actionné par la récursive, et change donc l'image, qui ont pour nom 4.png, 5.png et 6.png.



Le clic

On utilise la fonction MouseClicked provenant de l'interface MouseListener. On lui ajoute des fonctions de suppressions d'items, de décalage vers la gauche et vers le bas qui proviennent du fichier Decalage.java ainsi que la fonction de vérification de fin de partie. (soit deletingItem(), decalageGauche(), decalageBottom() et gameOver()).



Pour supprimer un item on utilise une fonction qui compte le nombre d'items sélectionnés. On utilise un tableau de booléen qui forme le groupe d'item dont la valeur est true. Cela arrive lorsque la fonction récursive détecte un item similaire à celui sélectionné par la souris. Dans ce cas la fonction countSelected compte les cases true, et cela forme un groupe. Lorsque que l'on clique sur ce groupe dans le tableau de la grille, cesvaleur sont passées à 3 ou à 7 (à cause de la récursive qui monte de +4 la valeur du tableau, et dont la valeur sera celle de la même image mais en surbrillance.) Dans ce cas là on n'affiche rien à cet endroit.

Juliette Maumené & Arnaud Jorandon

Réorganisation de la grille (Décalage.java)

Ce fichier est composé des fonctions :
-deletingItem();
-decalageGauche();
-decalageBottom();
-gameOver();
-detectedNearColor();

Constructeur

Le constructeur prend en argument un objet de type Grid pour utiliser le tableau de la grille ainsi que sa copie en utilisant la fonction getcaseValue();. On récupère aussi le tableau de booléen qui sélectionne les groupes d'items au passage de la souris.

Suppression d'items

Cette fonction parcourt une copie du tableau de la grille originale qui sert initialement à ré-afficher la grille normalement lorsque que le curseur quitte un groupe d'items, et le tableau de booléen qui passe regroupe les items sélectionnés en true. Si en parcourant ce tableau de booléen il rencontre une case true, alors on passe les cases rencontrés à 3, et la case deviendra vide. On l'utilise dans le fichier OnClick.java pour que la fonction de suppression se fasse seulement au clique de l'utilisateur.

Décalage vers la gauche

On parcourt à nouveau le tableau de la grille, mais seulement les colonnes et si on rencontre une case vide à la position [i][9] (soit la dernière ligne de i-ème colonne) cela signifie qu'une colonne est vide et donc on peut parcourir les lignes. Si i n'est pas la dernière colonne ni la première alors on peut décaler vers la gauche. On enregistre alors la valeur de l'image actuelle, et on lui attribue la valeur de celle de droite. Son ancienne valeur est attribuée à la valeur de celle de gauche. On attribue ces valeurs à la copie du tableau de la grille et celle du tableau de grille initial.

Ensuite pour qu'il re-parcourt chaque colonne on met i à -1.

Décalage vers le bas

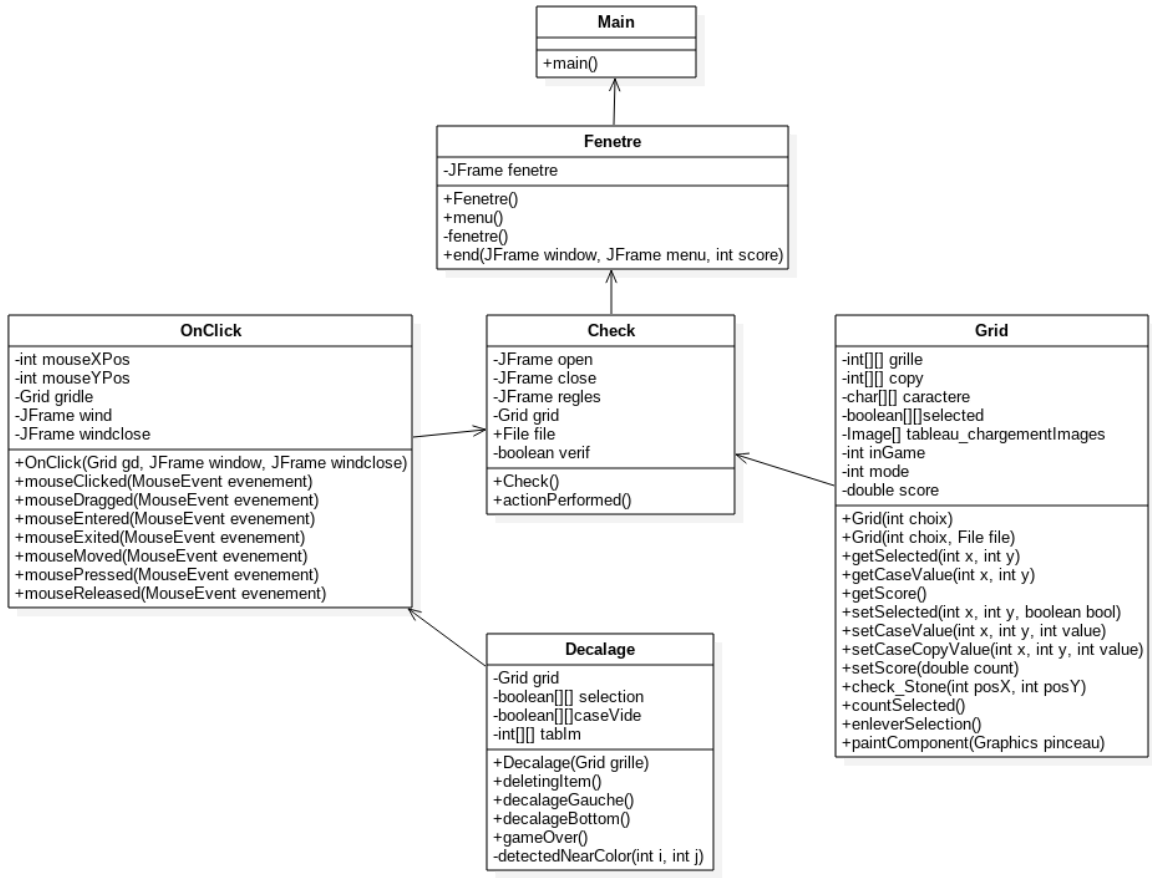
Cette fonction repose sur le même concept que le décalage à gauche, on parcourt le tableau de grille initial et sa copie, si on rencontre une case vide et que la case d'au dessus ne l'est pas alors on attribue à la case vide la valeur de la case au dessus, et ainsi de suite. Pour que l'on puisse refaire ça pour chaque case de la colonne, on remet la valeur de j à -1.

Condition de fin

La fonction gameOver parcourt le tableau de la grille, et si la fonction detectedNearColor(); renvoie true, dans ce cas le jeu continue, sinon le jeu est terminé.

La fonction detectedNearColor(int i; int j) prends en argument des coordonnées d'un tableau, et regarde les cases autour seulement si elle n'est pas vide (en haut, en bas, à gauche et à droite). Si elles ont la même valeur c'est qu'il y a encore la possibilité de continuer le jeu dans ce cas là la fonction renvoie true. Mais si la case est isolée alors on renvoie false. Et si c'est le cas pour toutes les cases restantes du tableau alors le jeu est terminé.

Structure du programme



(Ce diagramme est disponible dans les sources en format PDF)
Le programme commence par la classe Main. Cette classe ne contient que la méthode main() qui elle même ne contient qu'un appel de méthode. Cette classe sert juste pour débuter le programme.
Elle appelle la méthode menu() de la classe Fenêtre. Cette méthode permet de créer une fenêtre de menu comportant trois boutons. Le contrôleur des boutons se trouve dans la

classe Check. Nous avons décidés de séparer les deux contrôleurs, celui des boutons et celui de la souris lors de la phase de jeu, pour plus de clarté dans le programme.
la méthode menu() appelle la méthode fenetre() de la classe Fenetre pour créer une fenêtre de jeu.
Notre classe “moteur” du programme est la classe Grid. C’est elle qui stocke toute les données du plateau de jeu: la grille de valeur, la grille de sélection, la fonction d’affichage, le score etc. Un objet Grid est généré dans la classe Check au moment du choix du mode de jeu. Un objet Grid peut être construit de deux manières différentes, avec un fichier ou sans, en fonction du mode.
La classe OnClick gère toutes les fonctions de clic et utilisation de la souris lors du jeu.
Un objet est aussi généré lors de la création de la grille dans la classe Check.
La classe Decalage est appelée lors des clics de la souris. elle s’occupe de gérer les mouvements de perles tout au long de la partie, ainsi que de vérifier le cas où la partie serait terminée.

Conclusion

(Juliette Maumené)

Ce projet m’a beaucoup plus apporté que le premier en C, peut-être que d’avoir des bases en C et redémarrer à zéro sur du java a été beaucoup plus enrichissante et facile à comprendre. Cependant la difficulté à été d’organiser le code de la bonne manière, à cause de l’orientation objet c’est plus subtil.

(Arnaud Jorandon)

Ce projet était plus plaisant à réaliser de part la plus grande facilité à utiliser des objets graphiques et tous les événements liés. Le travaille était plus facile et surtout plus évident que lors du projet en C. L’organisation nécessaire au Java nous a aussi permit de nous répartir plus facilement les tâches, en se disant qui faisait quelle classe par exemple. Apprendre à utiliser GIT m’a aussi beaucoup servi et rend le projet plus simple à faire en groupe, à quelques exceptions près, n’étant pas très à l’aise avec les branches, nous faisions tout sur la branche “master”. Je pense que nous avons fait un travail relativement clair dans l’ensemble.