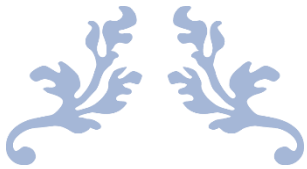


SAMEGAME

Aléxia BERNARD & Florian Cunsolo

06 AVRIL 2021

Groupe 2

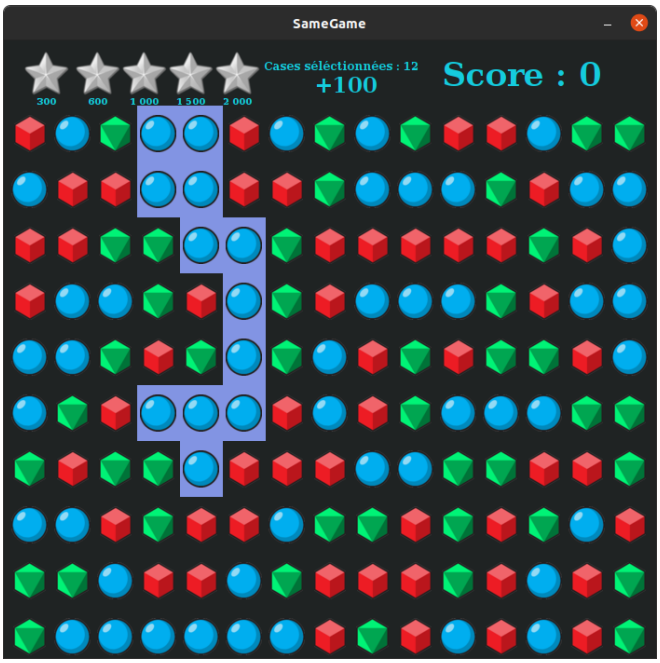
Table des matières

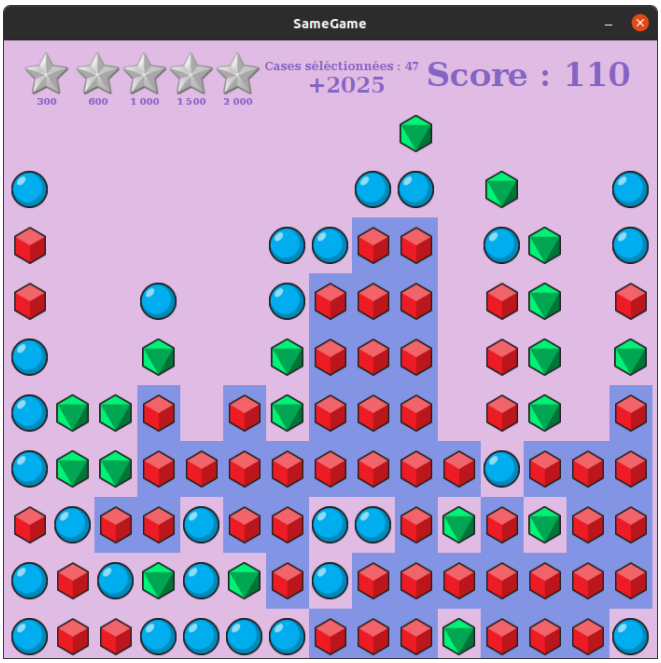
Introduction :

Le projet qui nous a été donné est la réalisation d'un jeu « SameGame », consistant à vider une grille, contenant trois types différents de blocs, en combinant les blocs identiques positionnés les uns à côté des autres afin de marquer le meilleur score possible. Pour cela, nous avons utilisé du Java ainsi que l'API standard Java.

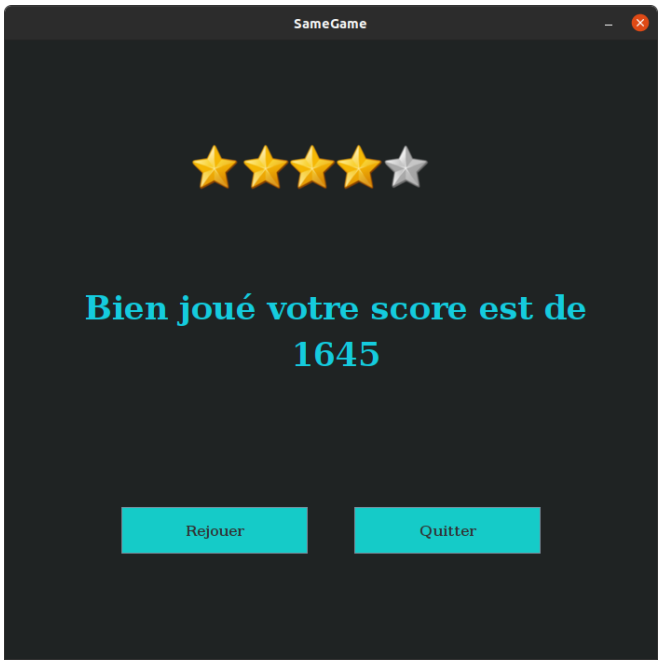
Description :

1 : Le premier écran permet au joueur de choisir comment il veut que la grille se génère : aléatoirement, ou bien en entrant lui-même un fichier contenant 10 lignes et 15 colonnes avec pour seuls caractères R, V ou B. Le joueur peut également choisir le thème qu'il désire (sombre ou clair).





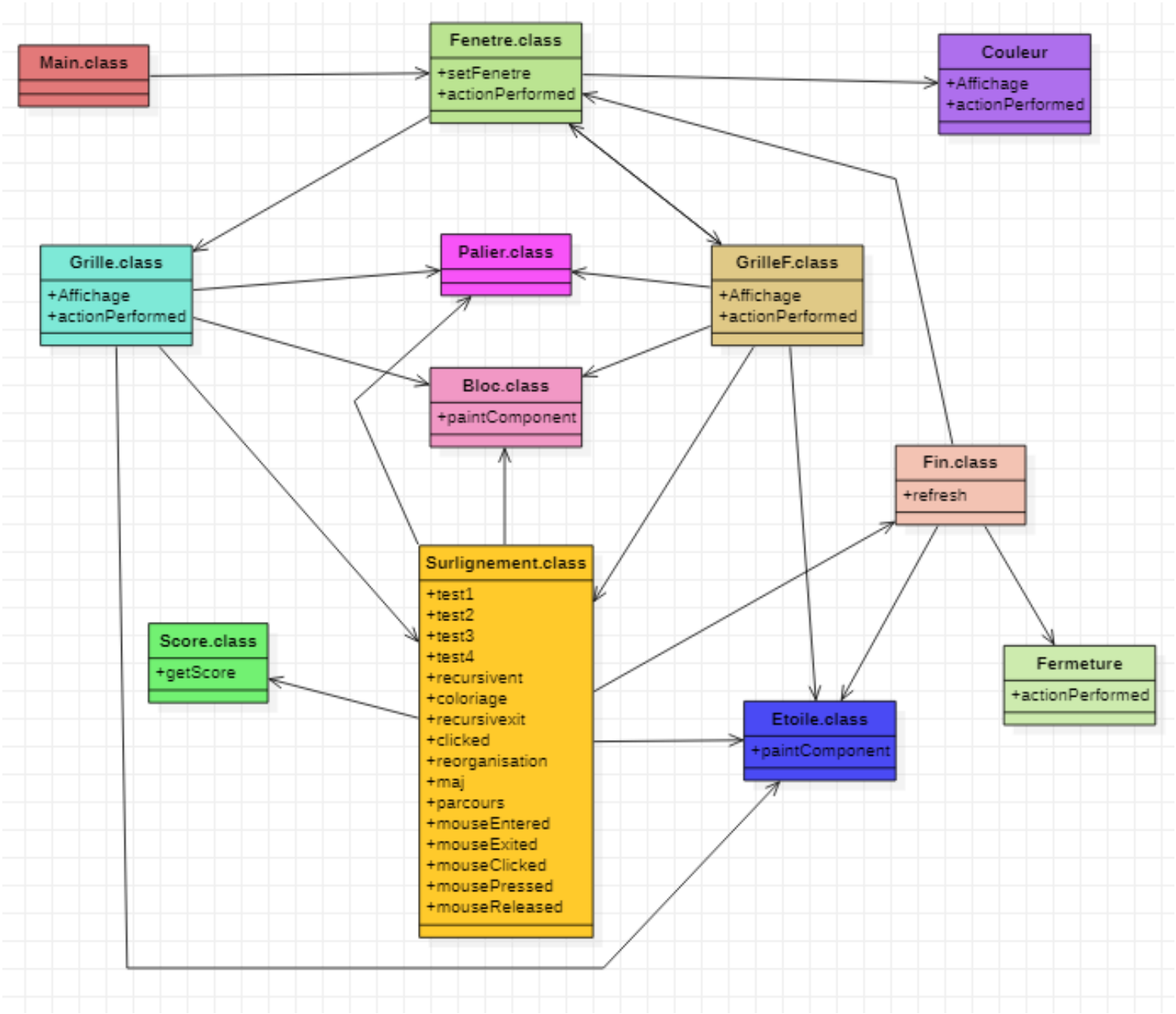
2 : Le deuxième écran affiche la grille créée : soit aléatoirement soit à l'aide du fichier que le joueur a choisi. Nous pouvons également lire le score, le nombre de cases qui sont actuellement sélectionnées par le joueur, le nombre de points que ces cases lui rapporteront s'il clique dessus ainsi que des étoiles avec des paliers en fonction du score. La partie démarre dès que le joueur clique sur un groupe de bloc. A chaque fois que le joueur vide la grille, le fenêtre se réorganise et met à jour la grille, le score et les paliers.



3 : Le troisième écran affiche le score avec le nombre d'étoiles que le joueur a obtenu et permet au joueur de rejouer ou de quitter le jeu. Si le joueur rejoue, le

premier écran apparait de nouveau. Si le joueur décide de quitter le jeu, la fenêtre se ferme.

Présentation :



Exposition :

Nous allons maintenant vous expliquer le fonctionnement global de notre algorithme ainsi que certaines spécificités.

Tout d'abord la classe « Main » crée une nouvelle fenêtre et un composant graphique (JLabel) qui permet de prédéfinir un thème par default à l'ouverture de la fenêtre. Une fois ces deux choses faites le *Main* appelle la méthode **setFenetre**. Dans la classe « *Fenetre* » la méthode **setFenetre** permet d'afficher le menu du jeu avec les règles ainsi que les boutons « fichier » et « jouer » qui permettent respectivement de générer une grille grâce à un fichier soit aléatoirement. Si le bouton fichier est cliqué et qu'un fichier non correct est sélectionné, le menu se réaffiche à sa position actuelle et avec le message d'erreur correspondant à l'erreur généré en dessous des deux boutons. Le jeu possède deux thèmes qui peuvent être choisi grâce à deux boutons (sombre ou clair) et qui change la couleur de tous les composants que nous avons choisi grâce à la classe « *Couleur* ».

Une fois que l'un des deux boutons est cliqué (fichier ou jouer), la classe « *GrilleF* » ou la classe « *Grille* » se lance et génère la grille demandée par le joueur. *GrilleF* lit chaque ligne du fichier donné par l'utilisateur et affiche la grille correspondante tandis que *Grille* utilise un nombre aléatoire pour choisir quel bloc mettre dans chaque case puis affiche également la grille correspondante. Pour afficher un bloc, la classe « *Bloc* » nous permet de dessiner chaque type de blocs possible (rouge, vert ou bleu) avec la redéfinissions de la méthode **paintComponent**. GrilleF et Grille affichent la fenêtre de jeu avec la grille généré mais aussi plusieurs indications qui permettent d'aider le joueur, notamment le score actuel du joueur, le nombre de cases surlignées avec le nombre de points que cela apportera au joueur s'il supprimait les blocs surlignés et enfin des étoiles qui correspondent aux paliers que le joueur atteint en fonction de son score. Pour identifier chaque case de la grille, nous avons créé 3 tableaux : le tableau « test » qui correspond à chaque case du tableau et qui permet de remplir ces cases avec un bloc, le tableau « grille » correspond au numéro du bloc (1,2,3) qui est associé à la case quand la grille est créée et le tableau « occupe » qui correspond aux cases surlignées par le joueur (initialisé à 0). Ces trois tableaux sont donc transmis à la classe « *Surlignement* » qui est un observateur de type MouseListener. Chaque case se voit affecté un observateur quand elle est créée.

La classe principale qui permet à notre jeu de fonctionner est « *Surlignement* ». Cette classe est utilisée quand le joueur entre, sort et clique sur les cases de la grille.

Quand la souris entre dans une case (**mouseEntered**) le programme effectue des tests récursifs (**recursivent**) pour voir si les cases autours sont du même type, c'est-à-dire regarder si la grille de la case actuelle est égale à la grille de la case au-dessus, à droite, à gauche ou en bas, et fait de même pour les cases qui sont pareil autour de la case actuelle. Pour chaque case qui se retrouve dans le groupe, leur valeur dans le tableau occupe est égale à 1. Le programme parcours ensuite toute la grille est dès qu'une case est occupée (occupe[i]=1), la méthode **coloriage** surligne la case de la couleur prédéfinie et met aussi à jour le nombre de cases sélectionnées et le gain associé.

Quand la souris sort d'une case (**mouseExited**), c'est la méthode **recursivexit** qui parcours toute la grille puis applique l'ancienne couleur de fond sur toutes les cases occupées et met le nombre de cases sélectionnées à 0 ainsi que le gain.

Quand la souris est cliquée (**mouseClicked**), le programme parcourt toute la grille et vérifie d'abord qu'il y a au moins un groupe de deux cases ; si c'est le cas, elle remplace le score par l'ancienne valeur plus le nouveau gain (classe « *Score* ») puis redessine toute la grille avec chaque case surlignée remplacée par la couleur de fond de la fenêtre (**clicked**, les autres cases sont redessiner et se voit ajouter un nouvel observateur), ainsi la case n'a plus de bloc et devient vide. Cette nouvelle grille a sûrement créé des trous, c'est pourquoi **reorganisation** vérifie chaque case depuis la dernière jusqu'à la première et si la case est vide (vide[i]==1), on vérifie le nombre de cases vides au-dessus et le bloc de la première case non vide au-dessus est affecté à la case actuelle (grille[i]=grille[(i-(cpt2*15))] etc. jusqu'à ce qu'il n'y ait plus de trou. A la fin de cette boucle, on effectue une mise à jour de la grille avec **maj** qui réaffiche simplement la grille avec les nouvelles valeurs de chaque case. Dans un deuxième temps, la méthode **reorganisation** va vérifier s'il n'y a pas de colonne vide. Elle va parcourir la grille par colonne, si les 10 cases de la colonne sont vides alors elle affecte la valeur grille de chaque case de la colonne suivante (case de droite) à chaque case de la colonne actuelle. Pour vérifier qu'il n'y avait pas plusieurs colonnes vide d'affilées, on effectue à nouveau le test sur la colonne que l'on vient de décaler. Enfin, on attribut les bonnes valeurs aux cases dorénavant vides puis on réaffiche la grille avec **maj**. **Maj** ne fait pas que réafficher la grille avec les bonnes cases, mais elle s'occupe aussi d'afficher le bon nombre d'étoiles avec les paliers correspondant au score (classe « *Etoiles* ». Un réaffichage de la grille signifie soit la création d'une case vide, soit la modification des cases déjà existante, ce qui signifie qu'un observateur à été ajouté à chacune de ces cases pour pouvoir continuer de jouer. Pour terminer, il y a une dernière méthode (**parcours**) qui vérifie soit que la grille est vide et termine le jeu, soit que chaque case n'a plus aucun voisin du même type et termine également le jeu. Dans les deux cas, elle renvoie un booléen qui est égal à *true* sinon elle renvoie la valeur *false*.

A chaque clique, le programme doit calculer le score à l'aide de la classe « *Score* ». Afin de récupérer le score générer par cette classe il suffit d'appeler la méthode **getScore**. Une fois le score récupéré il est transmis à la classe « *Etoiles* » qui permet d'afficher le palier correspondant. Les étoiles en or correspondent aux étoiles obtenues par le joueur.

Quand **parcours** renvoie *true*, **mouseClicked** crée un nouvel objet de type « *Fin* » et lance sa méthode (**refresh**). **Refresh** vide l'ancienne page et affiche la page de fin du jeu avec le nombre d'étoile que le joueur à gagner, son score, ainsi que deux boutons (rejouer et quitter). Le premier bouton (rejouer) lance une nouvelle page avec le menu et le deuxième (quitter) ferme simplement la fenêtre.

La dernière classe « *Couleur »* est très importante, c'est grâce à cette classe que le thème peut être transmis de page en page. Son constructeur reçoit tous les composants qu'il faut modifier quand on passe du thème clair au thème sombre.

Conclusion Florian :

Ce second projet m'a permis d'appliquer les connaissances en Java que j'ai pu apprendre en cours. J'avoue avoir une légère préférence pour la programmation en Java qui permet à mon avis d'avoir plus de liberté sur la création de programme.

Aléxia et moi avons décidé de garder notre binôme pour ce projet puisque nous trouvions qu'il avait bien fonctionné la dernière fois. En effet, à présent nous avons l'habitude de travailler ensemble ce qui nous rend beaucoup plus efficace. A chaque fois que l'un de nous deux était bloqué, l'autre trouvait la solution et cela nous permettait d'avancer assez vite. Il y a eu une légère difficulté pour la réorganisation de la grille, le temps de trouver comment faire d'abord sur papier sinon le reste a été assez fluide.

Pour finir, j'ai pris beaucoup de plaisir à coder ce jeu et je suis fière de notre réalisation.

Conclusion Aléxia :

Au départ j'avais beaucoup de mal à comprendre comment coder en Java et je pense que ce deuxième projet m'a parmi de mettre en pratique mes connaissances, les approfondir et surtout comprendre comment coder en Java. Il m'a également permis de confirmer le fait que j'aime faire des mini-jeux et travailler en binôme. J'ai également pu approfondir mes connaissances sur Git en créant ma propre branche (git branch nom_de_la_branche).

Florian et moi avons décidé de garder notre binôme car nous trouvons que nous avons un bon rythme de travail, une bonne communication, que nous sommes efficaces mais également lorsque l'un n'arrivait pas à faire quelque chose ou était bloqué, l'autre lui donnait des indications ou le débloquait. De plus je trouve que notre travail est propre, autant dans le code que à la vue de l'utilisateur, pratique pour le joueur, avec les points qui peuvent être ajoutés par le joueur (« +100 ») mais également donne un défi à ce dernier grâce aux paliers représentés par les étoiles.

Pour conclure, je pense que nous n'avons pas eu de grosse difficulté à produire ce code, cependant nous avons tout de même eu quelques problèmes, par exemple lorsque nous cliquions sur la case en bas à gauche cela faisait tout disparaitre, cela était dû au fait que nous avions oublié de compter cette case dans une boucle.