Projet Same Game

Projet réalisé par Mélissa Da Costa et Chloé Trugeon

SOMMAIRE:

Introduction

Le projet est un jeu de type SameGame, Le jeu ce présente sous une forme de grille composée de 15 colonnes et 10 lignes remplie par des blocs de différentes couleurs: rouge, vert et bleu. Le but de ce jeu est d'avoir le meilleur score possible en éliminant des groupes de blocs.
Un groupe de blocs se définit par un ensemble d'au moins deux blocs de la même couleur qui sont adjacents entre eux.
En éliminant des groupes de blocs, les blocs restants se déplacent horizontalement et verticalement de façon à boucher les trous formés.
La partie est considérée comme finie lorsqu'il reste seulement des blocs sans voisin de même couleur ou alors lorsqu'il ne reste plus de bloc du tout. Le jeu est entièrement jouable à la souris.

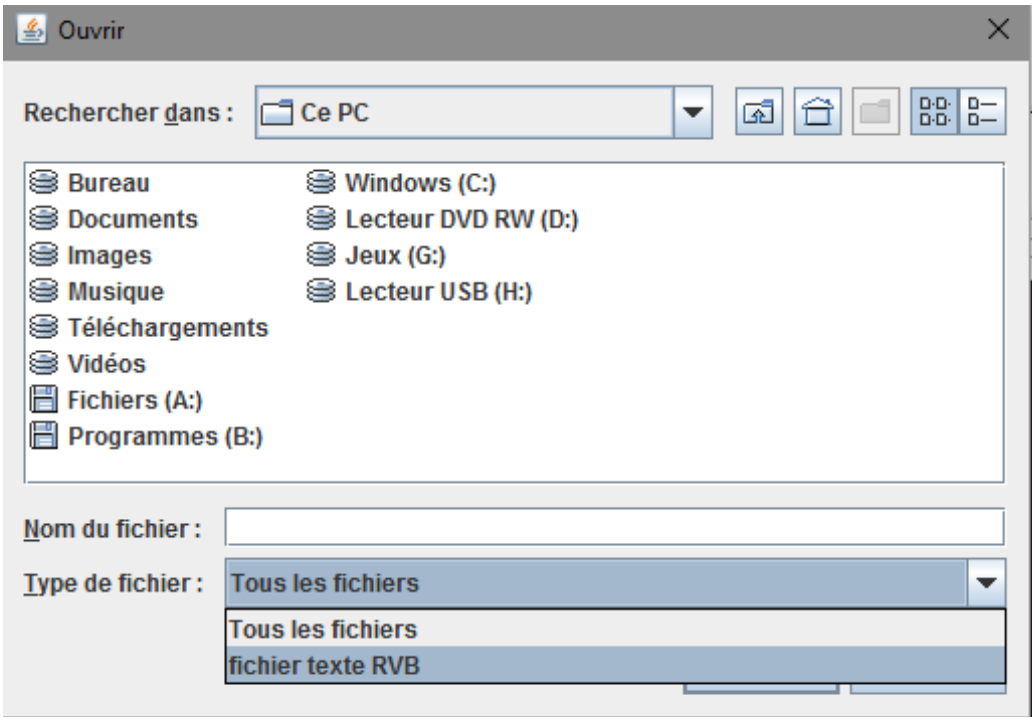
Description

Le Menu:

Premièrement notre projet s'ouvre sur un menu grâce à la fonction principale de notre classe *Main* qui appelle la méthode *creerMenu* de la classe *Menu* qui se charge de l'afficher. Ce menu contient deux boutons qui laissent deux possibilités de jeu au joueur.

Notre classe *BoutonMenu* permet de distinguer lequel de ces deux boutons à été cliqué.
Si le joueur clique sur «Jouer avec une grille aléatoire», alors le programme fermera cette fenêtre puis immédiatement après en ouvrira une autre (avec la méthode *créerNouvelleFenetre* de la classe *NouvelleFenetre)* contenant une grille de blocs rouge, vert et bleu. Cette grille à été créée de manière aléatoire, dans notre classe *Hasard*, grâce à la classe préexistante *Random*. Le principe est de sélectionnée une lettre au hasard dans la suite de lettre «RVB». La lettre sélectionnée permet alors de remplir un tableau de 10 lignes et 15 colonnes.

La deuxième possibilité est que le joueur clique sur «Jouer à partir d'un fichier». Dans ce cas, le même principe est suivi, le menu se ferme puis une fenêtre s'ouvre, laissant le choix au joueur de sélectionner le fichier voulue. Cette fenêtre à été créée grâce à la classe *JfileChooser* déjà existante.

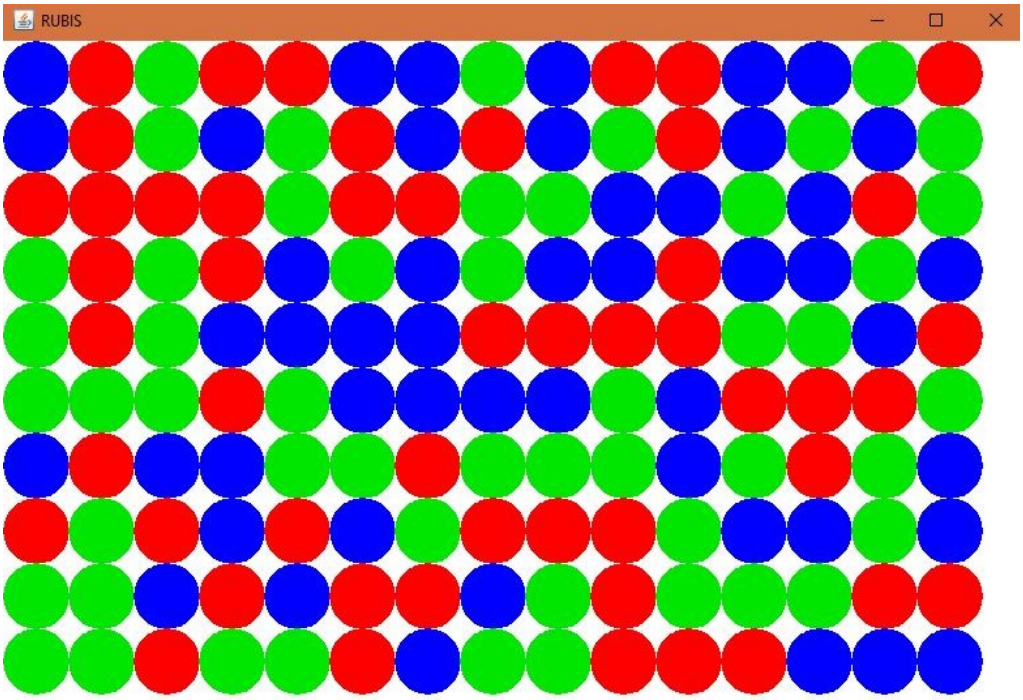


Grâce à cette classe nous avons pu ajouter un paramètre filtrant le choix des fichier. Celui-ci se nomme «fichier texte RVB», il n'est pas sélectionner par défaut et permet de filtrer seulement les fichiers *.txt*.
En effet, si le joueur souhaite jouer une partie avec une grille de blocs prédéfinie, il peut alors fournir au jeu un fichier contenant les lettres R, V ou B désignant respectivement un bloc de couleur rouge, vert ou bleu. Ces lettres doivent former 10 lignes et 15 colonnes.

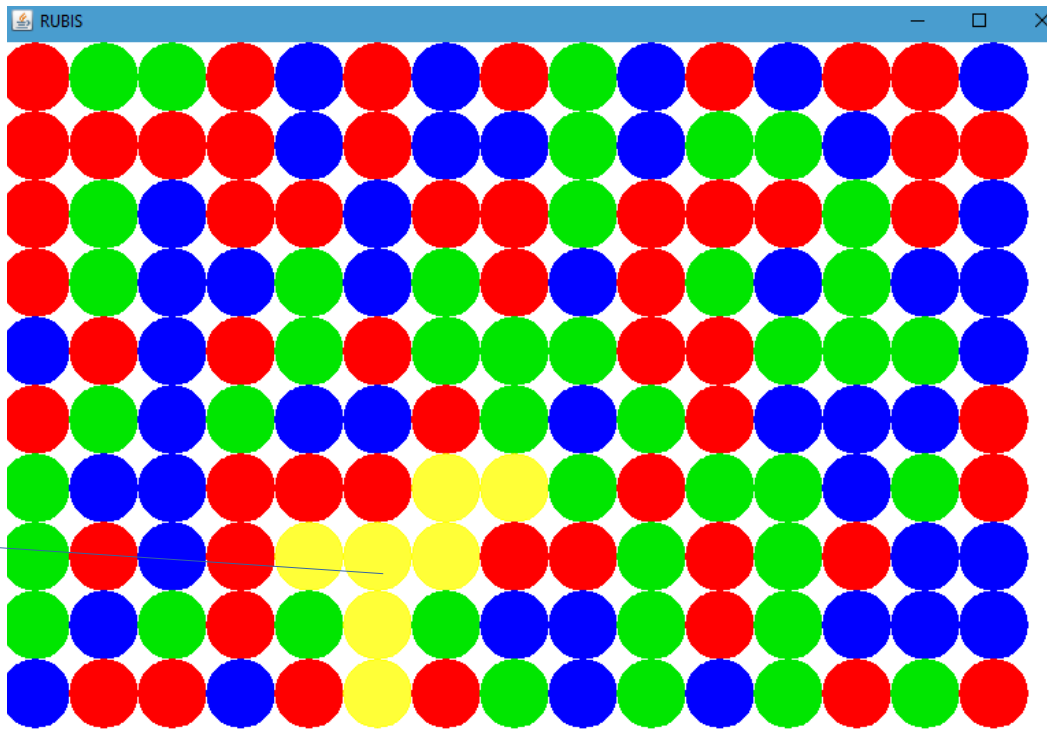
```
RVVRRVRVBBBBRBV
BVVVVRBVVBRRVRB
VBBRVRBVRRBRRRR
BRBVBRBBVVBRVRV
RVBVBBBRRBRRRBV
RVVVBBRBVVBVVRB
BRBRBBBRBVRVRRV
VRRVVBBVVBBRVVV
BVRRVVBRVRRRBVV
BBRBBBBRVVRRVRB
```

Le survol de la grille:

Pour chacune de ces possibilités, la classe *GrilleAleatoire* est appelé. Celle-ci permet grâce à ça méthode *paintComponent* de lire le tableau rempli de RVB fourni et d'afficher la grille correspondante.
Lorsque le joueur à fait son choix, apparaît alors la fenêtre de jeu.



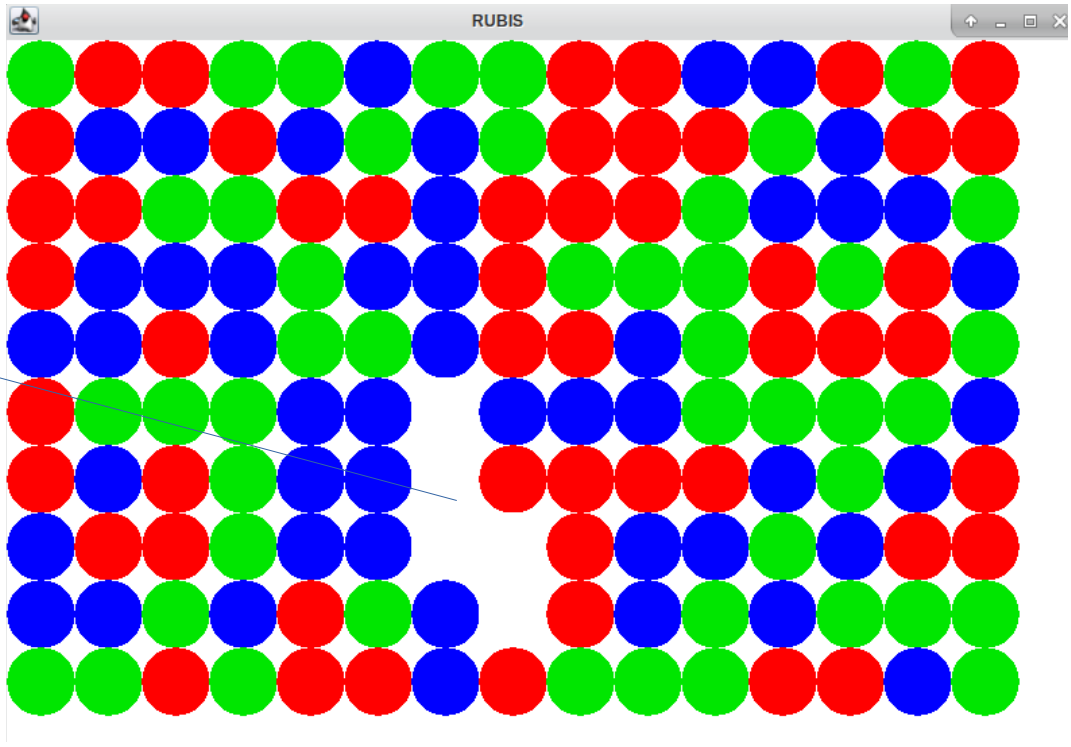
Pour représenter nos blocs nous avons fait le choix de dessiner des cercles avec la méthode *drawOval*. Ensuite le joueur peut survoler à l'aide de sa souris la grille. Lorsqu'il survol un groupe de blocs, celui est visuellement accentué en se coloriant en jaune.



Le déroulement de la partie se fait principalement grâce à notre classe *Survol*. En premier lieu elle nous permet de faire le lien entre les coordonnées de la souris dans la fenêtre et les cases du tableau grâce à un calcul. Suite à ce calcul, on continue la partie si les coordonnées de la souris sont dans un intervalle entre le minimum et le maximum de la taille d'un bloc.

Ensuite vient la détermination des groupes de blocs. Cette détermination se fait dans la classe *Bloc*. On donne à cette classe la case du tableau où la souris est et la méthode *determinerBloc* vérifie les cases adjacentes à celle-ci. Si une des cases adjacente est de la même couleur que notre case, alors on place à cette endroit dans le tableau un 'S' pour «Survol» et on incrémente l'attribut *blocValide* de 1. Si celui-ci est supérieur ou égale à 2, l'ensemble des blocs survolé est considéré comme un groupe de bloc.
Puis par récursivité, la méthode est appelée encore une fois avec les coordonnée de la case du bloc adjacent jusqu'à ce que tout le bloc survolé soit identifié par un 'S'.
Une fois ceci fait, la méthode *paintComponent* de la classe *GrilleAleatoire* permet de ré afficher la grille avec des cercles jaunes lorsque le tableau contient des 'S'. Ce ré affichage est possible grâce la méthode *repaint*. *À* chaque mouvement de souris géré par l'interface *MouseMosionListener*, une réactualisation de la fenêtre est effectué avec cette méthode.
Enfin, lorsque le joueur clic sur un groupe de bloc survolé, celui-ci disparaît.

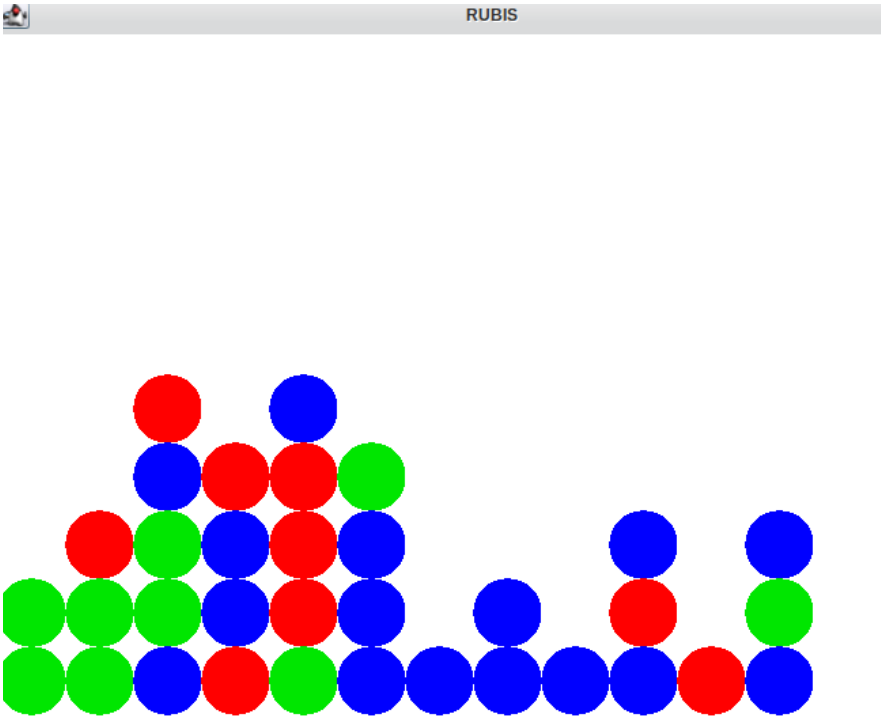
La disparition des blocs et leurs déplacements:



Pour ce faire, nous appelons la classe *EffacerBlocs*, qui grace à la méthode *mouseClicked* de l'interface *MouseListener* remplace les 'S' de notre tableau par des 'E' signifiant «Effacer». Puis par le même principe que précédement la fenêtre s'actualise pour ré afficher la grille en remplacant les cercles jaunes survolés par des cercles blancs, ce qui les fond dans la fenêtre car elle est de la même couleur.

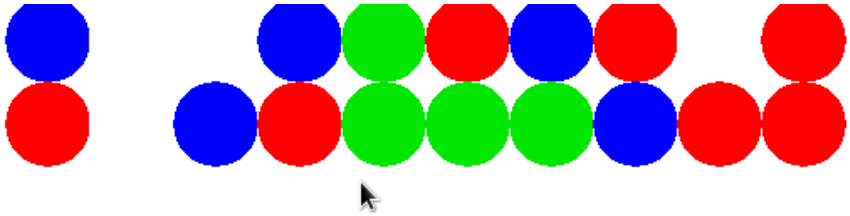
La méthode *mouseClicked* de la classe *EffacerBlocs* va nous être utile pour d'autres action qui doivent être mis en place après le clic. En effet, nous allons l'utilisé pour appeler le déplacement de notre grille ainsi que l'affichage du score.

Lorsque les blocs disparaissent, ils doivent se déplacer horizontalement et verticalement de façon a combler les trous. Pour ce faire nous avons appelé la méthode *deplacer* de la classe *Deplacement* dans *EffacerBlocs* pour qu'elle soit appelé à chaque clic. La methode *déplacer* est composée de deux partie. La première pour le déplacement verticale. La méthode parcourt le tableau et si elle tombe sur un bloc préalablement effacé, elle vérifie si les blocs au-dessus de lui existe. S'il n'y a rien, alors on sauvegarde ce bloc et on inverse les blocs du dessus avec celui-ci de façon à le faire decendre et on le déplace autant de fois qu'il le faut pour qu'il se retrouve le plus en bas possible.
Le même principe est suivi pour le déplacement horizontale.



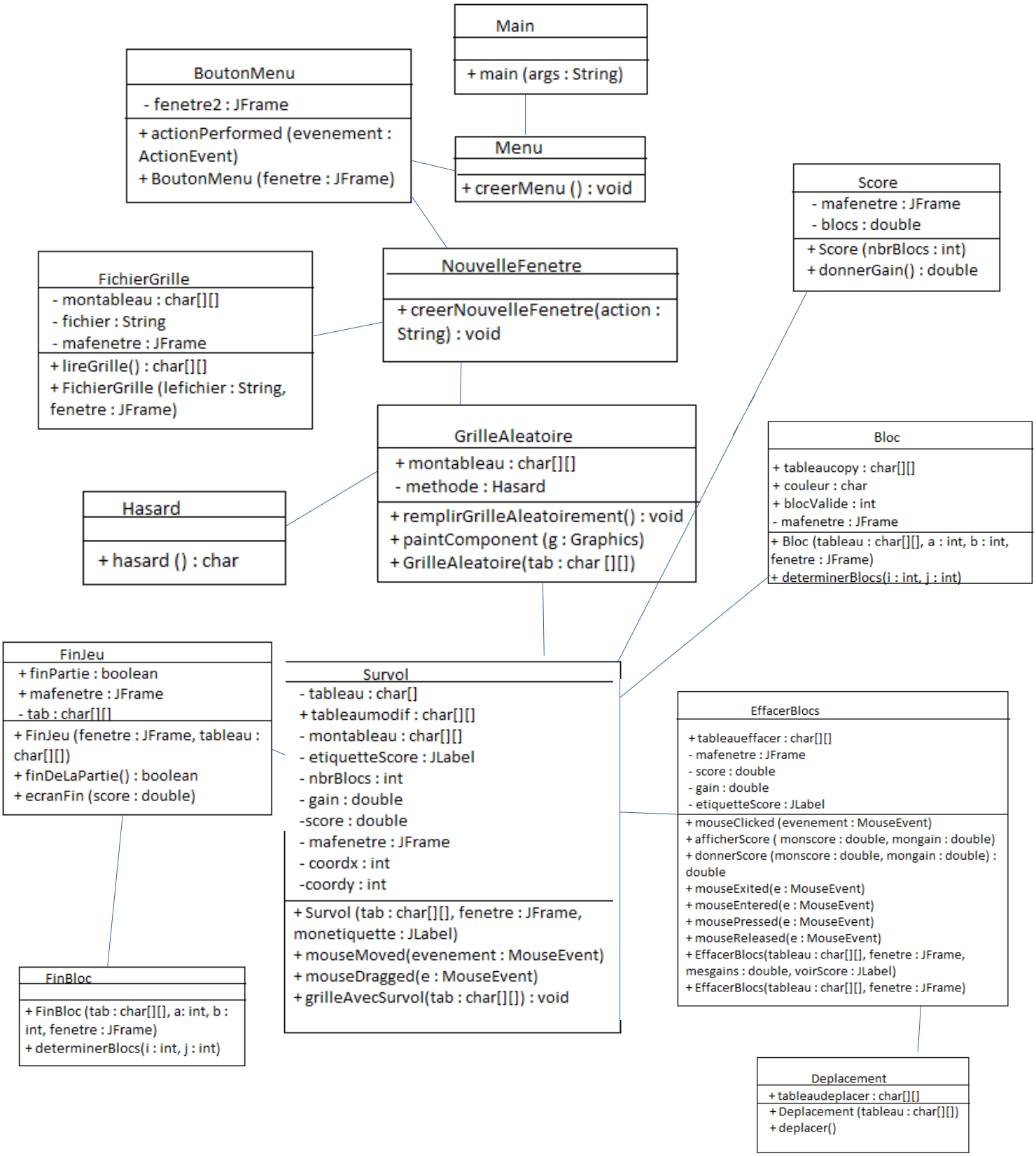
Le score du joueur et la fin de la partie:

La gestion du score se fait en deux partie, premièrement il faut déterminer le gain obtenu pour le groupe de bloc effacé. Pour ce faire nous utilisons l'attribut *blocValide* qui nous donne le nombre de bloc contenu dans le groupe. En donnant cet attribut à la méthode *donnerGain* de classe *Score*, celle-ci renvoie le gain obtenu grâce à un calcul.
Ensuite, la méthode *donnerScore* de la classe *EffacerBlocs* permet de mettre a jour le score en ajoutant à chaque clic le gain obtenu.
Pour finir, l'affichage du score se fait dans la méthode *afficherScore*, toujours dans la classe *EffacerBlocs*. Elle met à jour l'étiquette crée en début de partie (dans la classe *Survol*) en lui donnant une chaîne de caractère qui contient le score ansi que le gain de l'action précédente, pour que le joueur est une idée plus précise du nombre de points qu'il obtient.



Finalement, nous devons détecter la fin de la partie. Pour ce faire, lorsque nous survolons un groupe de bloc qui n'est pas valide, c'est-à-dire qu'il n'a pas de voisins de la même couleur. Alors, on appelle la méthode *ecranFin* de la classe *FinJeu*. Celle-ci appelle la méthode *finDeLaPartie* qui renvoie true lorsque la partie est terminer. On détecte une partie finie en parcourant le tableau et lorsqu'un bloc n'est pas effacé (quand la case du tableau est différente de 'E') , on appelle la méthode *determinerBlocs* de la classe *FinBloc*.
La classe *FinBloc* hérite de la classe *Bloc* et le principe de la méthode *determinerBlocs* est le même sauf que là nous ne remplissons pas le tableau de 'S'.
Si la méthode *finDeLaPartie* renvoie true, alors la méthode *ecranFin* prend le relais et ferme la fenêtre du jeu puis en ouvre une autre dans laquelle un écran de fin apparaît. Cet écran nous donne notre score et nous laisse le choix de rejouer ou de quitter.
La gestion de ces deux boutons se fait grâce à le classe *BoutonMenu* qui gère aussi les boutons du menu. Si on clique sur «Rejouer», alors la fenêtre se ferme et la méthode *creerMenu* de la classe *Menu* est appelé puis le programme se poursuit. Si le bouton «Quitter» est appelé, alors le programme s'arrête.

Diagramme de classe:



Conclusion

Mélissa Da Costa

Pour ce second projet j'ai essayé d'éviter les erreurs faites lors du premier projet. J'ai fait en sorte de mieux gérer mon code en essayant de produire un code non redondant, en utilisant le plus possible les mêmes morceaux de code comme les classes.
Ce projet m'a permis de mieux comprendre les spécificités du Java. Comme l'héritage ou les interfaces. J'ai aussi entièrement compris la gestion des événements, malgré le fait que nous avons perdu du temps par rapport à la ré actualisation de notre fenêtre et aux nombreux passages de tableau. En effet dès le début du projet nous nous sommes compliqué le travail en utilisant deux tableaux de différentes taille. Celà nous à causé pas mal d'erreurs que nous avons pu rectifierpar la suite.

Chloé Trugeon

J'ai trouvé ce second projet un plus difficile que le précédent, mais contrairement au premier projet, j'ai essayé de créer au plus des classes différentes pour executer chaque morceau de code.. En effet, je ne savais pas, de prime abord, comment m'y prendre. De plus nous avons rencontré plusieurs problèmes, du au chois de la taille de nos tableau, en partie, que nous avons du refaire. Cependant je pense que cette erreur et les autres que nous avons pu faire peuvent être considérée comme une bonne chose. En effet, c'est grâce à elles que j'ai pu me rendre de compte de ce qui n'allait pas et surtout, c'est en les corrigeant que j'ai mieux compris les spécificitées du Java.