SameGame

DUT INFORMATIQUE 2021
IUT FONTAINEBLEAU (77)

Shana LEFEVRE &
Arthur DECORBEZ
Groupe 3 et 4

Table des matières

I] Introduction

Pour ce deuxième semestre, nous avons comme projet tutoré le « SameGame » à coder en java.

Le but de ce jeu est de vider la grille de ses blocs. Il y a trois blocs possibles : bleu, rouge et vert. Les blocs s'éliminent par groupe. Un groupe est composé d'au minimum de deux blocs identiques adjacents. Le système de points dépend du nombre de blocs par groupe. On utilisera la formule suivante :

$(n - 2)^2$

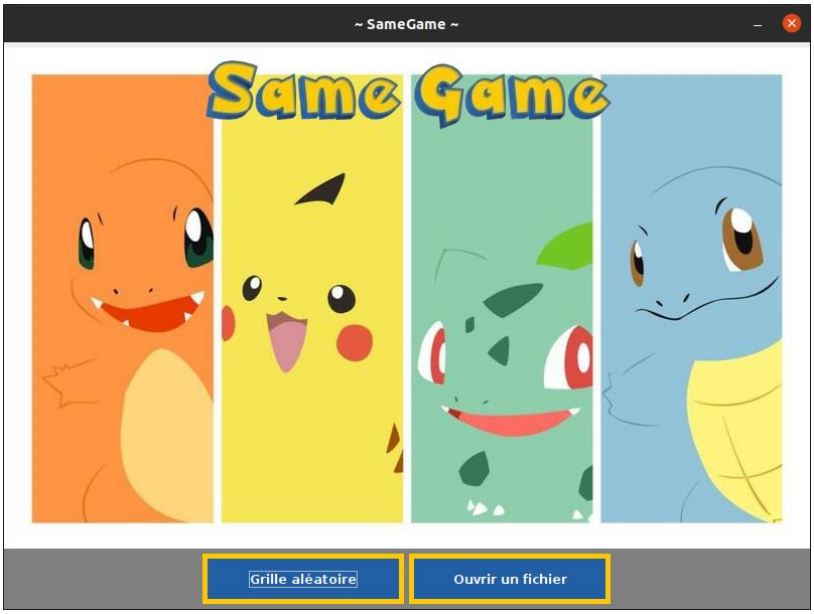
Exemple : Groupe de 5 → $(5 – 2)^2 = 9$ points

Dès qu'un groupe est éliminé, les autres blocs se déplacent vers la gauche ou bien vers la droite afin de supprimer les « trous ».

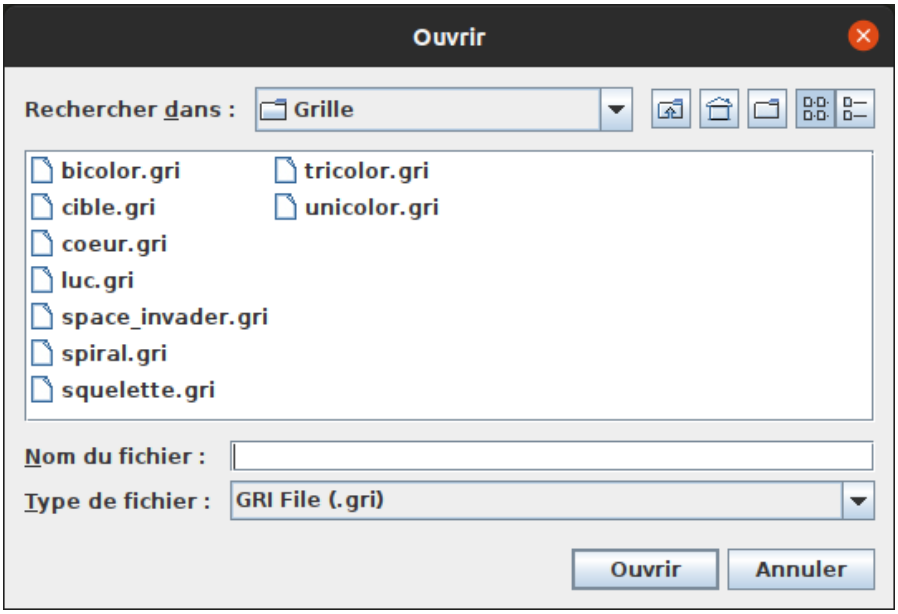
II] Description et fonctionnalités

Durant toute l'exécution de notre programme, nous y avons ajouté des mentions dans le terminal qui nous ne servirons comme points de repères et qui nous assurerons le bon fonctionnement de notre jeu.

Notre jeu accueille le joueur sur notre page menu. Delà, l'utilisateur peut choisir comment générer sa grille : à partir d'un fichier ou bien aléatoirement. Les deux modes ont la même finalité mais c'est la génération de la grille qui changera.



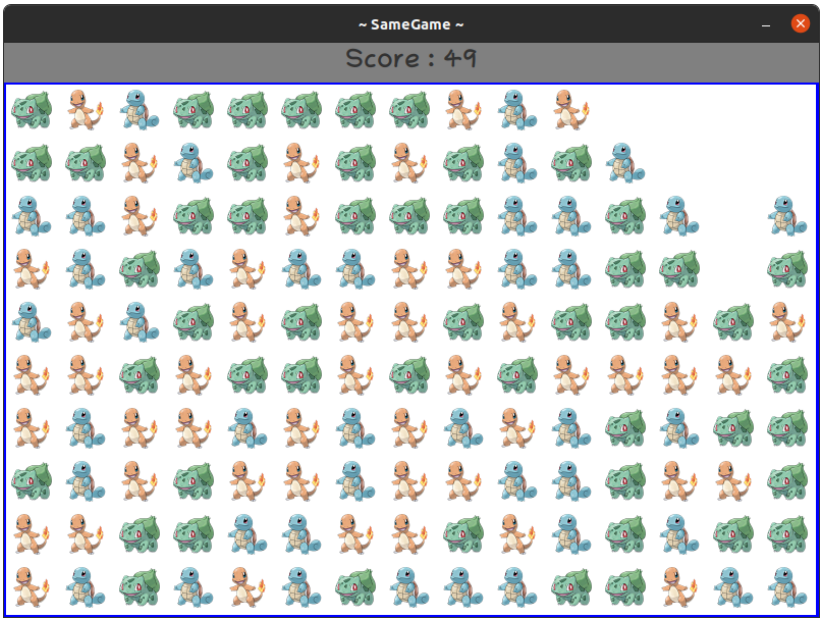
Si l'utilisateur choisit d'ouvrir un fichier, cette fenêtre ci-dessous s'ouvrira. Le joueur ne pourra ouvrir qu'un fichier ayant comme extension « **.gri** ».



Après la génération d'une grille aléatoire, ou après l'ouverture d'un fichier. La grille apparaît sur la fenêtre de jeu.

La détection d'un groupe de même bloc se fait automatiquement. Dès que le joueur mettra sa souris sur un des blocs de la grille, son groupe apparaîtra en fond jaune.

Lorsque le joueur clique sur le groupe, alors la grille change si besoin (déplacement lignes et colonnes). Le score se met ensuite à jour :



Quand il n'y a plus la possibilité de faire de groupe, cela veut dire que la partie est finie.

Le jeu renverra sur le menu de fin et ainsi, l'utilisateur aura le choix de rejouer ou bien de quitter le programme.

III] Présentation de la structure du programme

Notre programme est composé de plusieurs fichiers répartis en quatre grandes parties.

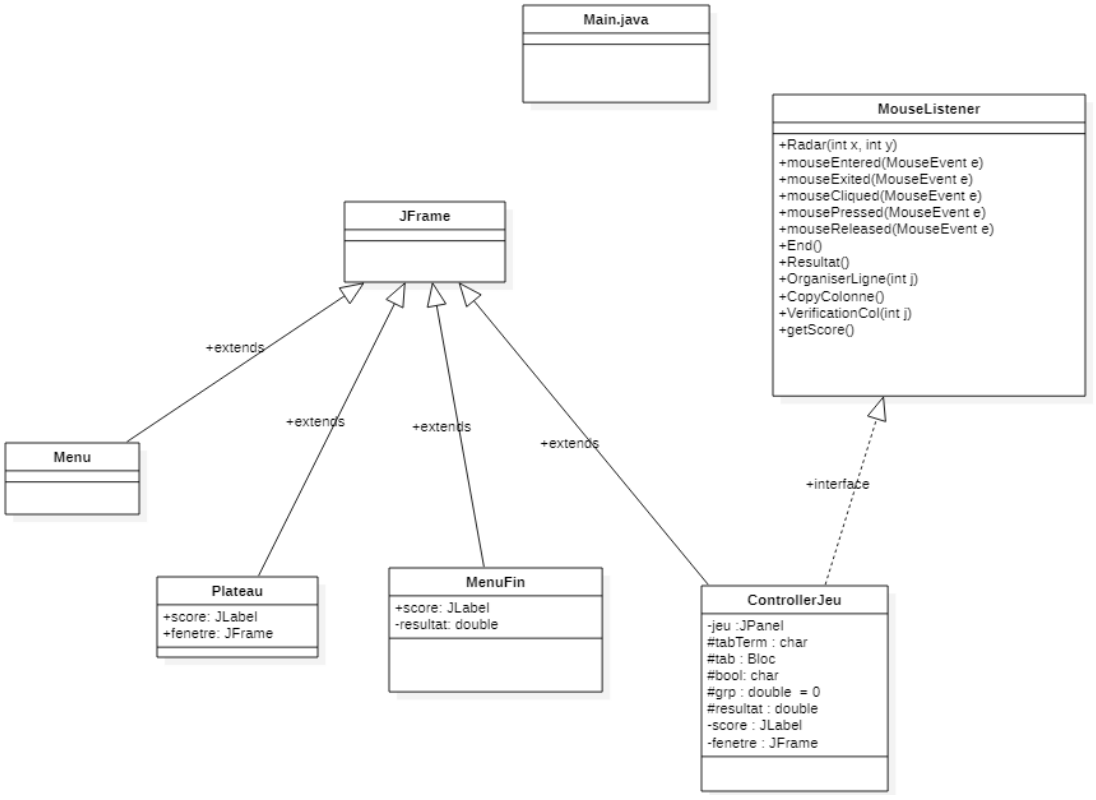
- La première partie : l'accueil du joueur avec le fichier Menu.java ;

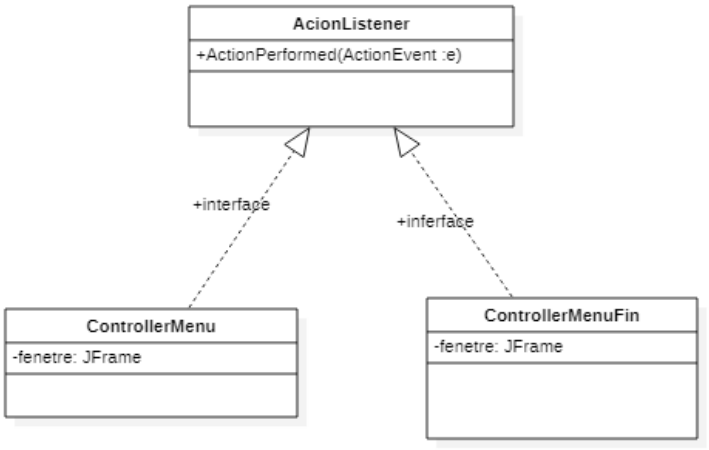
- La seconde partie : la création de la grille (aléatoire ou fichier) ;

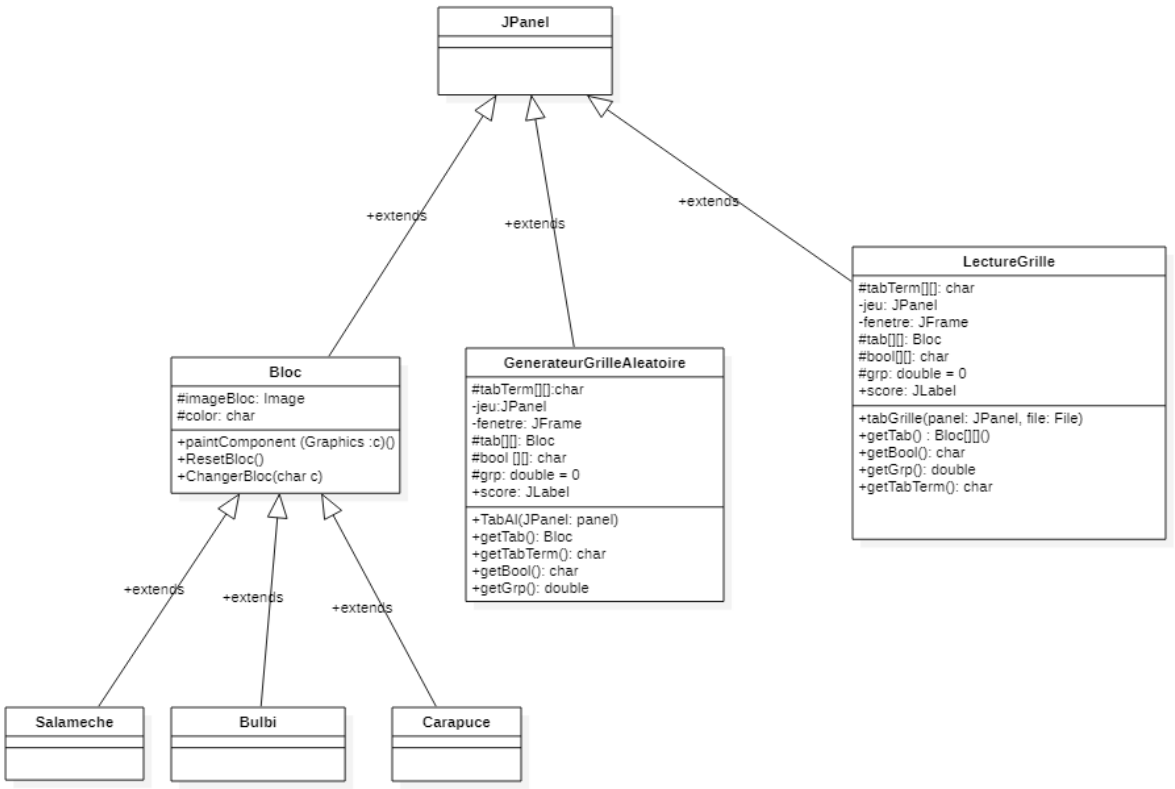
- La troisième partie : le jeu ;

- La quatrième partie : la fin du jeu avec le fichier MenuFin.java et la possibilité de relancer le jeu.

Voici ci-dessous le diagramme de classes correspondant à notre projet :




AcionListener
+ActionPerformed(ActionEvent :e)
+interface
+inferface
ControllerMenu
-fenetre: JFrame
ControllerMenuFin
-fenetre: JFrame


JPanel
+extends
+extends
+extends
LectureGrille
#tabTerm[][]: char
-jeu: JPanel
-fenetre: JFrame
#tab[][]: Bloc
#bool[][]: char
#grp: double = 0
+score: JLabel
+tabGrille(panel: JPanel, file: File)
+getTab() : Bloc[][]()
+getBool(): char
+getGrp(): double
+getTabTerm(): char
Bloc
#imageBloc: Image
#color: char
+paintComponent (Graphics :c)()
+ResetBloc()
+ChangerBloc(char c)
GenerateurGrilleAleatoire
#tabTerm[][]:char
-jeu:JPanel
-fenetre: JFrame
#tab[][]: Bloc
#bool [][]: char
#grp: double = 0
+score: JLabel
+TabAl(JPanel: panel)
+getTab(): Bloc
+getTabTerm(): char
+getBool(): char
+getGrp(): double
+extends
+extends
+extends
Salameche
Bulbi
Carapuce

IV] Les algorithmes

Pour nos algorithmes, nous utilisons deux tableaux de type *char* pour référencer les coordonnées d'un bloc :

- **tabTerm[][]** : correspond à la valeur d'un bloc. Il peut prendre comme valeur soit : '**R'** (rouge), '**V'** (vert), '**B'** (bleu) et '**X'** (case n'ayant pas de bloc)

- **bool[][]** : correspond au statut d'un bloc. Il peut prendre comme valeur soit : '**T'** (true), '**F'** (false) et '**.'** (pas de bloc donc pas de statut particulier).

1er algorithme : Détection des groupes au survole de la souris :

Nous utilisons la méthode **Radar** qui prend en paramètre un x et un y de type int qui correspondent respectivement à une ligne et à une colonne que le joueur aura survolés avec sa souris.

La méthode va commencer par vérifier si tabTerm[x][y] a la même valeur que les tabTerm[][] du haut, du bas, de la droite et de la gauche soit R, V ou B.

Si le tabTerm[][] découvert est de la même valeur, alors à ses coordonnées , on y place un 'T' au tableau bool[][]. Le tableau bool[][] permet de dire que tel bloc appartient ou non au groupe donc, si bool[][] == 'F' alors le tabTerm[][] correspondant n'appartient pas au groupe contrairement à bool [][] = 'T'. Donc, pour lancer une vérification dans les 4 sens, il faut que bool[][] == 'F'.

Après chaque appel de la méthode Radar, on réinitialise le tableau bool[][] = 'F'. Cependant si bool[][] = '.', alors cela signifie que tabTerm[][] n'a plus de bloc correspondant (R, V, B), il prend donc la valeur 'X'.

2ème algorithme : Détection des lignes vides

Afin de faire déplacer le ou les bloc(s) dès qu'une « case » est libre, il faut détecter une case libre. On commence par parcourir le tableau ligne par colonne avec deux boucles imbriquées. Si on trouve tabTerm[][] == 'X', alors on appelle la méthode **OrganiserLignes** qui récupère en argument la ou les colonnes où il y a une case libre.

Dans notre méthode **OrganiserLignes**, on regarde pour chaque ligne que la ou les case(s) soient libres. On « copie » en mettant toutes les valeurs qui nous intéresse dans la variable *color* de type char. Enfin, nous modifions la valeur de tabTerm[][] en case libre en ligne - 1 pour après rajouter notre variable *color* en argument en appelant notre méthode **ChangerBloc**.

3ème algorithme : Détection des colonnes vides

Puis, après d'avoir détecté des lignes vides, il faut détecter s'il y a des colonnes vides. Pour cela, on commence par parcourir toutes les colonnes une par une. Pour chaque colonne, on vérifie si notre méthode **VerificationCol** est vraie. Si cette méthode retourne *true*, alors on fait appel à la méthode **CopyColonne**.

La méthode **VerificationCol** prend en paramètre la n colonne et elle permet de voir si toute la colonne est vide (tabTerm[][] == 'X'). Si oui, alors elle retourne *true*, sinon *false*.

La méthode **CopyColonne** permet de copier les colonnes concernées et de les décaler vers la gauche. Pour se faire, nous créons trois boucles imbriquées. Les deux premières concernent le nombre total de colonnes que l'on peut décaler. Dans ce cas, 14/15 colonnes peuvent être décalées. Puis on vérifie que la colonne est vide avec la méthode **VerificationCol**. Enfin, pour chaque ligne, on décale de colonne + 1.

4ème algorithme : Détection de fin de jeu

A chaque clique, on vérifie s'il y a encore la possibilité de faire des groupes avec les blocs restants.

On appelle la méthode **End**. Si la méthode **End** est *false*, alors le jeu est fini.

La méthode **End** parcours la totalité du tableau. Si les cases ne sont pas vides (tabTerm[][] != 'X') et si elles n'appartiennent pas à un groupe (bool[][] = 'F'), alors, on appelle la méthode **Radar** (*décrite précédemment)*. Si le groupe est supérieur ou égal à 1, alors la méthode nous retourne *true*. Dans le cas contraire, *false*, il n'y a plus de groupe possible, c'est la fin du jeu.

V] Conclusions personnelles

Shana :

« Le projet a été encore une très bonne expérience pour moi. En effet, il m'a permis de découvrir mes capacités à réaliser un projet en équipe. J'ai pu réutiliser ce que j'avais pu faire et apprendre avec notre précédent projet. J'ai rencontré quelques difficultés avec l'organisation des fichiers. Je pense ne pas avoir assez anticipé avec des schémas. J'ai beaucoup appris en réalisant ce projet : écrire une Javadoc, réaliser un diagramme de classes à partir de notre programme en java, mieux structurer un programme. En ce qui concerne le travail de groupe, il s'est bien passé. Nous avons réussi à nous répartir les différentes tâches, mettre en commun et s'entraider. Pour conclure, ce projet tutoré a été une très bonne expérience. »

Arthur :

« Comme le projet C (Blokus), j'ai beaucoup plus apprécié travailler sur ce projet que sur le Sudoku de l'année dernière. Le SameGame a pour avantage d'être beaucoup plus flexible dans la personnalisation qu'un simple Sudoku qui est bien plus traditionnel.

Avec ce projet, j'ai mieux assimilé le Java que l'année précédente, notamment l'incrémentation du modèle MVC, le fait d'avoir un peu d'expérience avec le Java aide.

Je suis plutôt fière de ce que nous avons produit et je compte bien continuer de développer ce projet et ainsi continuer d'acquérir de nouvelle connaissance.
Je remercie mon binôme, Shana LEFEVRE, qui m'a supporté durant l'entièreté du projet et qui a également travaillé d'arrache-pied pour pouvoir rendre ce travail. »