Rapport du projet SameGame en java :

Sommaire

Introduction 2

Fonctionalités du programme 2

Analyse de la fonction de detection des groupes 3

Structure et organisation du programme 5

Conclusions 7

Introduction

Le projet SameGame consiste en la réalisation d'un jeu dont le but est de vider une grille contenant des blocs de 3 couleurs différentes : rouge, vert et bleu. La grille est au format 15 blocs à l'horizontale et 10 blocs à la verticale. L'élimination des blocs se fait de la manière suivante : lorsque le joueur clique sur un bloc alors un groupe indiqué en surbrillance se forme puis il disparait. Le groupe comprend les blocs situés en haut, en bas, à droite et à gauche du bloc cliqué, puis de manière récursive, les blocs à proximité des blocs indiqués précédemment se retrouvent ainsi dans le groupe. Les blocs situés en diagonale du bloc en question ne sont pas pris en compte dans le groupe. Ensuite, la grille peut être construite de 2 manières différentes, soit aléatoirement soit à partir d'un fichier texte qui comprend les caractères R, V, B permettant la création de la grille. La partie se termine lorsqu'il n'y a plus du tout de blocs dans la grille ou bien qu'il ne reste plus que des groupes d'un bloc. À la fin le joueur peut alors recommencer en retournant vers le menu principal ou bien quitter le jeu. Dans ce même menu de fin le joueur pourra alors voir le score final de sa partie.

Fonctionnalités du programme

Le programme pour atteindre tous les objectifs demandés précédemment par le jeu possède plusieurs fonctionnalités permettant la gestion d'une partie et de la recommencer une fois celle-ci terminée. La classe grille possède des constructeurs l'un à partir d'un fichier dont on lit les caractères un à un pour ainsi les stocker dans un tableau. Le second quant à lui se fait à partir d'un objet Random qui renvoie un chiffre de 0 à 2 et en fonction de celui-ci on attribue les caractères R, V, B.

La classe Boule a été créée pour caractériser chacune des boules de la grillecoordonnée x et y, une longueur, une largeur et enfin une image qui sera affichée dans la fenêtre en fonction des paramètres

Ensuite, à partir de ces 2 objets la construction du noyau du jeu peut commencer à s'opérer. La classe SameGame gère le cœur du jeu, celle-ci possède 2 constructeurs l'un à partir d'une grille aléatoire et l'autre à partir de la grille construite à partir d'un fichier. Dans ces 2 constructeurs, on initialise les attributs score et état à 0. On charge aussi une première fois l'high score contenu dans le fichier high_score.bin.
Dans la redéfinition de la méthode paintComponent, les différents blocs sont dessinés en fonction du caractère stocker dans le tableau de notre objet grille, les images sont alors dessinées à partir de l'attribut tab_nom_image contenant les noms des images. On incrémente ainsi une double boucle pour ainsi obtenir notre grille d'images 15x10. 2 chaines de caractères sont dessinées en haut du panneau : le score actuel et l'high score pour ainsi observer l'évolution de ceux-ci.

Analyse de la fonction qui s'occupe des groupes :

La méthode compteGroupe permet de renvoyer un entier compteur qui représente le nombre de blocs qu'il y'a dans un groupe. Pour ce faire on incrémente une double boucle qui parcourt le tableau d'état et qui augmente alors le compteur d'un à chaque fois qu'une case du tableau est à true.

La méthode principale du jeu est groupedeboulesurbrillanceetenleve, c'est une méthode récursive qui a 3 rôles : la gestion de la surbrillance du groupe lorsque décision est égale à 1, le fait que l'on enlève les boules lorsque décision est égale à 2, ainsi que de juste compter le nombre de blocs dans le groupe à partir de la méthode compteGroupe si décision est diffèrent de 1 et 2. Cette méthode prend argument la position du bloc dans la grille avec des positions i et j pour situer le bloc en question dans le tableau. Pour tester un bloc on vérifie déjà si celui-ci n'est pas déjà égale à true dans le tableau d'état, si c'est le cas alors ce bloc passe a true puis on appelle 4 fois la fonction récursivement : pour la case de gauche si le bloc n'est pas situé sur la première colonne, la case de droite si le bloc n'est pas situé sur la

dernière colonne, la case du haut si le bloc n'est pas situé sur la première ligne et enfin la case du bas si le bloc n'est pas situé sur la dernière ligne. Pour gérer la surbrillance on fait passer les caractères en minuscule et pour gérer l'enlèvement on remplace le caractère par un espace blanc, ces caractères sont évidemment stockés dans le tableau de la méthode paint component lors du rappelle de la méthode avec repaint pour changer les images grâce à cette méthode.

La méthode remplirTabEtat permet de réinitialiser toutes les valeurs du tableau d'état à false pour ensuite pouvoir réutiliser les méthodes se servant de ce tableau d'état par la suite.

Ensuite, la méthode chuteDeBoule permet la gestion du fait que les blocs chutent si un caractère blanc se situe dans la position en dessous de celui-ci et que ce bloc n'est pas déjà un caractère blanc s'il n'est pas situé sur la dernière ligne d'a grillé. On parcourt alors toute la grille pour ainsi faire en sorte que tous les blocs chutent correctement.

La méthode EtatColonne renvoie faux si une colonne n'est pas vide et vraie si elle est vide. La méthode DecaleBouleGauche permet le décalage des colonnes vers la gauche si une colonne est vide. On teste chaque colonne de la grille puis on appelle la méthode EtatColonne, si elle renvoie vraie alors cette colonne prend les valeurs des caractères de la colonne de droite, et la colonne de droite devient la colonne vide, on répète cela jusqu'à ce qu'elle se situe totalement à droite car la méthode parcourt toute la grille. Pour assurer une sécurité pour les méthodes decaleboulegauche et chutedeboule on incrémente une boucle qui vérifie 10 fois si tous les blocs a bien chuté et si tous les blocs se sont bien décalés.

La méthode score s'occupe de renvoyer le score en fonction de l'évolution de la partie à partir de la formule : nouveau score = score actuel + (nb de boule du groupe – 2)^2. C'est l'attribut Scr qui s'occupe de prendre la nouvelle valeur a chaque évolution. On lui passe en attribut la méthode CompteGroupe après l'appel de la méthode récursive.

La méthode findejeu renvoie un booléen vrai si la partie est terminée et fausse sinon. Cette détermination se fait en parcourant encore une fois toute la grille puis on prend un à un chacun des blocs, si ce n'est pas un caractère blanc alors on détermine s'il fait partie d'un groupe d'un bloc ou plus. Si aucun groupe de bloc n'est supérieur à une taille d'un alors la partie est terminé, si on trouve un groupe d'au moins 2 blocs alors on renvoie false et la partie on continue.

Enfin une classe spécifique a été créer pour gérer les évènements concernant un objet Same Game, il s'agit de la classe EvenementJeu qui hérite des interface MouseMotionListener et de MouseListener. 2 méthodes ont été redéfinies dans cette classe :

-MouseClicked qui gère le fait que des blocs soient remplacés par un caractère blanc lors de la clique de la souris sur l'un d'eux. Pour situer le bloc sur lequel on clique on se sert des méthodes getx et gety que l'on encadre par les coordonne renvoyées par les différents objets boules. Une fois cette condition remplie, alors on met l'attribut décision à 0 pour que la méthode récursive ne s'occupe pas du remplacement des blocs par un espace blanc. Grace a cela on vérifie alors si la méthode compteGroupe renvoie un entier supérieur a un et que le caractère soit différent d'un caractère blanc. Si ces 2 conditions sont remplies alors on réinitialise le tableau état, puis décision passe à 1. On rappelle alors une nouvelle fois la methode recursive pour que le groupe soit remplacer par des caractères blancs, les

méthodes score et highscore permettent l'actualisation de ceux-ci ? On appelle enfin les methode chuteDeBoule et decaleBouleGauche puis enfin on appelle repaint pour que graphiquement le panneau soit mis a jour et que l'utilisateur puisse observer les changements.

La redéfinition de la méthode MouseMoved consiste en la même chose que celle de MouseClicked sauf que décision prend la valeur 2 pour ainsi gérer la surbrillance comme indiqué précédemment

Structure et organisation du programme

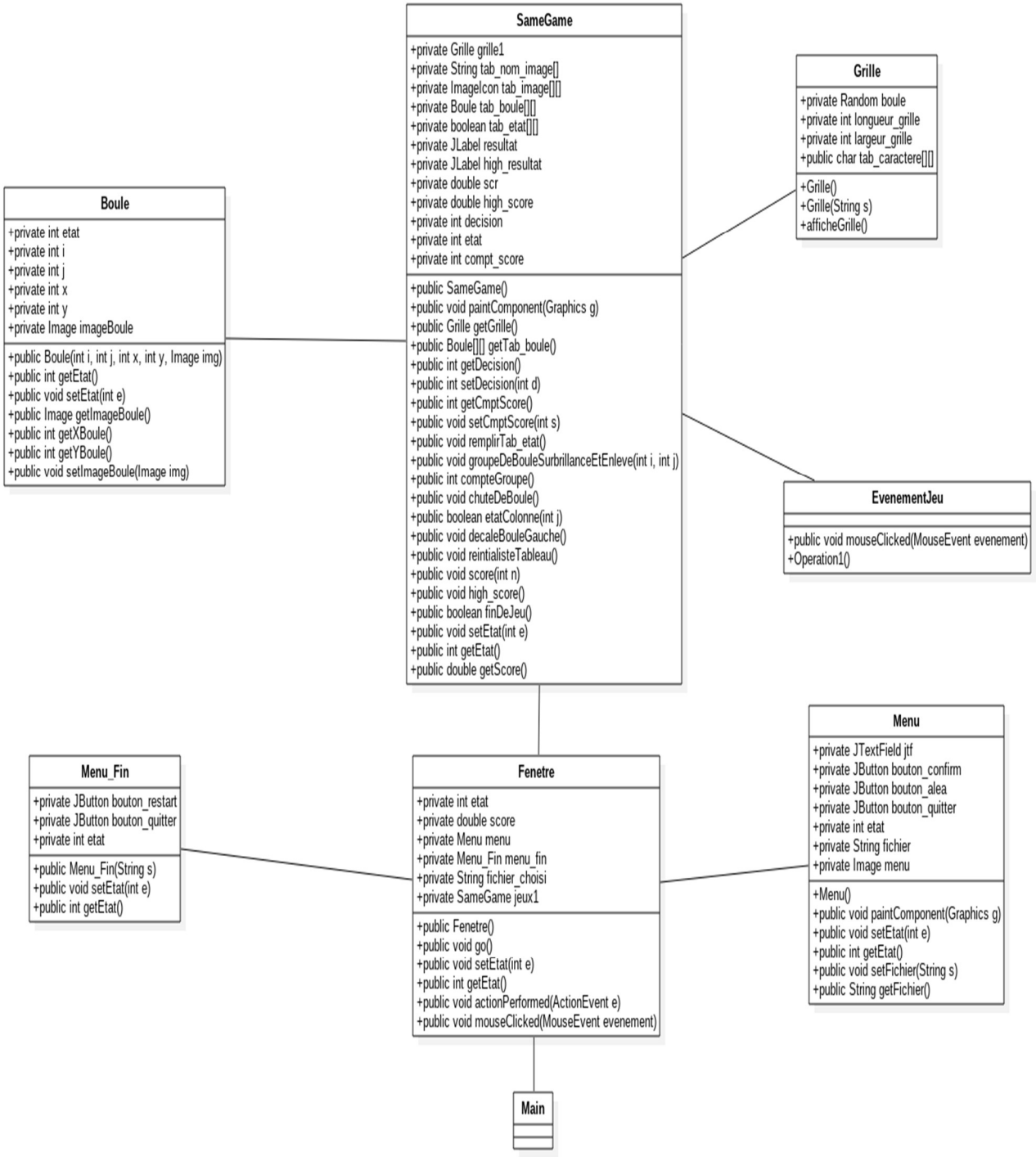
Le programme est divisé en 8 classes différentes ayant chacune un rôle bien particulier pour que le jeu puisse fonctionner correctement selon la bonne organisation. Comme indiqué précédemment un objet SameGame qui est le cœur du programme a besoin de plusieurs objets Boule et d'un objet Grille pour fonctionner. EvenementJeu représente tous les évènements agissant sur un objet SameGame. Ensuite la classe Fenetre a pour rôle la transition entre les 3 différents panneaux : le menu principal, le jeu en lui-même puis le menu de fin.

Nous allons ainsi détailler dans cette partie comment cela a été mis en place. Le menu principal comporte 3 boutons : un bouton pour lancer une partie aléatoire, une autre qui choisis un fichier grâce à JFileChooser, puis un bouton pour quitter qui ferme la fenêtre. Pour le menu de fin nous avons 2 boutons, un pour recommencer une partie ce qui nous rapporte directement au menu principal et un autre pour quitter le jeu. Ces 2 menus héritent de la classe JPanel.

La classe fenêtre qui hérite de JFrame a pour attribut les objets Menu, Menu_fin et SameGame pour chacun des 3 panneaux. L'attribut fichier_choisi sert grâce a JFileChooser après que le nom du fichier soit converti en string de prendre cette valeur pour que le constructeur de SameGame si c'est le bouton fichier choisis prenne cette valeur pour construire l'objet.

C'est dans une redéfinition de la méthode action performed que le changement de Panneau est géré. En utilisant le schéma suivant à chaque fois :

Création d'un tampon pour le nouvel objet menu ou jeux, l'attribut de la fenêtre correspondant a l'objet prend les valeurs du tampon, on lui ajoute le listener lui correspondant puis on utilise les 2 méthodes : getContentPane().remove(<ancien objet>) et geContentPane().add(<nouvel objet>), Puis on valide grâce à la méthode revalidate.


SameGame
+private Grille grille1
+private String tab_nom_image[]
+private ImageIcon tab_image[][]
+private Boule tab_boule[][]
+private boolean tab_etat[][]
+private JLabel resultat
+private JLabel high_resultat
+private double scr
+private double high_score
+private int decision
+private int etat
+private int compt_score
+public SameGame()
+public void paintComponent(Graphics g)
+public Grille getGrille()
+public Boule[][] getTab_boule()
+public int getDecision()
+public int setDecision(int d)
+public int getCmptScore()
+public void setCmptScore(int s)
+public void remplirTab_etat()
+public void groupeDeBouleSurbrillanceEtEnleve(int i, int j)
+public int compteGroupe()
+public void chuteDeBoule()
+public boolean etatColonne(int j)
+public void decaleBouleGauche()
+public void reintialisteTableau()
+public void score(int n)
+public void high_score()
+public boolean finDeJeu()
+public void setEtat(int e)
+public int getEtat()
+public double getScore()
Grille
+private Random boule
+private int longueur_grille
+private int largeur_grille
+public char tab_caractere[][]
+Grille()
+Grille(String s)
+afficheGrille()
Boule
+private int etat
+private int i
+private int j
+private int x
+private int y
+private Image imageBoule
+public Boule(int i, int j, int x, int y, Image img)
+public int getEtat()
+public void setEtat(int e)
+public Image getImageBoule()
+public int getXBoule()
+public int getYBoule()
+public void setImageBoule(Image img)
EvenementJeu
+public void mouseClicked(MouseEvent evenement)
+Operation1()
Menu
+private JTextField jtf
+private JButton bouton_confirm
+private JButton bouton_alea
+private JButton bouton_quitter
+private int etat
+private String fichier
+private Image menu
+Menu()
+public void paintComponent(Graphics g)
+public void setEtat(int e)
+public int getEtat()
+public void setFichier(String s)
+public String getFichier()
Menu_Fin
+private JButton bouton_restart
+private JButton bouton_quitter
+private int etat
+public Menu_Fin(String s)
+public void setEtat(int e)
+public int getEtat()
Fenetre
+private int etat
+private double score
+private Menu menu
+private Menu_Fin menu_fin
+private String fichier_choisi
+private SameGame jeux1
+public Fenetre()
+public void go()
+public void setEtat(int e)
+public int getEtat()
+public void actionPerformed(ActionEvent e)
+public void mouseClicked(MouseEvent evenement)
Main

Conclusions

Conclusion Alexis :

Pour conclure, ce projet a été selon moi une bonne solution pour tester à peu près tous les concepts vus pendant ce module du langage Java. Personnellement j'ai eu plus de facilités a réalisé ce projet que le projet Blocus car je pense avoir progressé quant à la logique à utiliser lors de la création de jeux assez complexes comme ceux-ci. Ensuite, cela m'a permis de confirmer que je préfère le Java au langage C car j'arrive à mieux visualiser la notion d'objet et l'organisation du programme est ainsi plus pratique à mettre en place. Les difficultés intervenues n'ont pas été vraiment nombreuses, juste quelques problèmes pour le compte du nombre de blocs dans chaque groupe ou encore la gestion de la transition entre les différents panneaux de la fenêtre qui n'est pas vraiment la méthode la plus propre à mon sens. Mais à part ces petits bémols le façonnement du jeu s'est bien déroulé et j'ai trouvé cela assez plaisant de le réaliser en travaillant en groupe notamment.

Conclusion Chahine :

Ce projet m'a permis d'approfondir mes connaissances du langage Java. J'avais en effet beaucoup de difficultés à comprendre l'essence de ce langage de programmation mais grâce à ce projet, j'ai pu m'améliorer et comprendre certaines notions que je n'avais pas acquises en cours. Nous avons eu quelques difficultés à la réalisation de la méthode permettant de compter le nombre de blocs dans un groupe afin d'afficher le score mais le problème a été finalement réglé. Le travail en binôme s'est très bien déroulé et nous avons eu recours à un partage des taches efficace et bien organisé.