UPEC
UNIVERSITÉ PARIS-EST CRÉTEIL

IUT
SENART
FONTAINEBLEAU

Rapport de situation d’apprentissage et d’évaluation

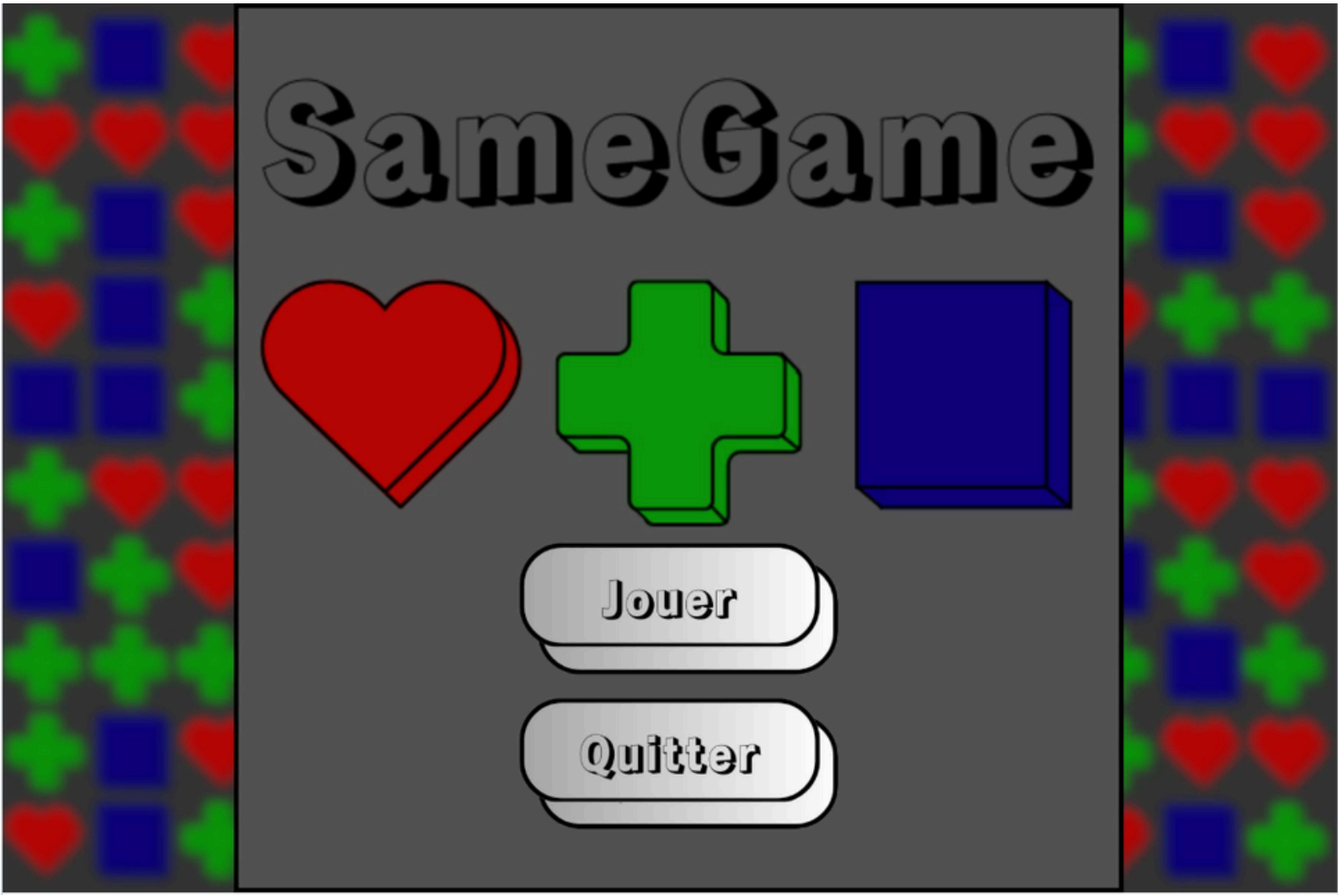


Table des matières

I. Introduction

Dans ce projet, Wael ATIK et moi même Emmanuel SRIVASTAVA-TIAMZON avons développé le jeu SameGame en Java sans emprunt extérieur sauf l'API officielle, pour notre année de BUT1 informatique.

Une grille est remplie de blocs de trois types et votre but est de la vider. Un groupe de blocs adjacents et de même type peut être retiré en cliquant dessus. Les blocs restants se réarrangent afin de boucher les trous, et on recommence jusqu'à avoir vidé la grille ou n'avoir plus que des blocs isolés.

SameGame, à l'origine appelé Chain Shot!, a été créé en 1985 par Kuniaki Moribe. Il a été distribué pour les plateformes FM-8 et FM-7 de Fujitsu dans un magazine mensuel japonais dédié aux ordinateurs personnels, intitulé Gekkan ASCII. En 1992, le jeu a été porté sous le nom de SameGame sur les plateformes Unix par Eiji Fukumoto, sur la série NEC PC-9801 par Wataru Yoshioka, et sur Macintosh sous le nom ChainShot! par Eiichiro Mawatari. En 1993, il a été porté sur Windows 3.1 par Ikuo Hirohata.

Aujourd'hui, au sein de l'iut de Fontainebleau pour un projet de situation d'apprentissage et d'évaluation il est recrée par au moins une trentaine de groupe de deux personnes.

Dans ce livrables vous retrouverez donc notre version. Sous tout ses aspects.

II. Carte mentale

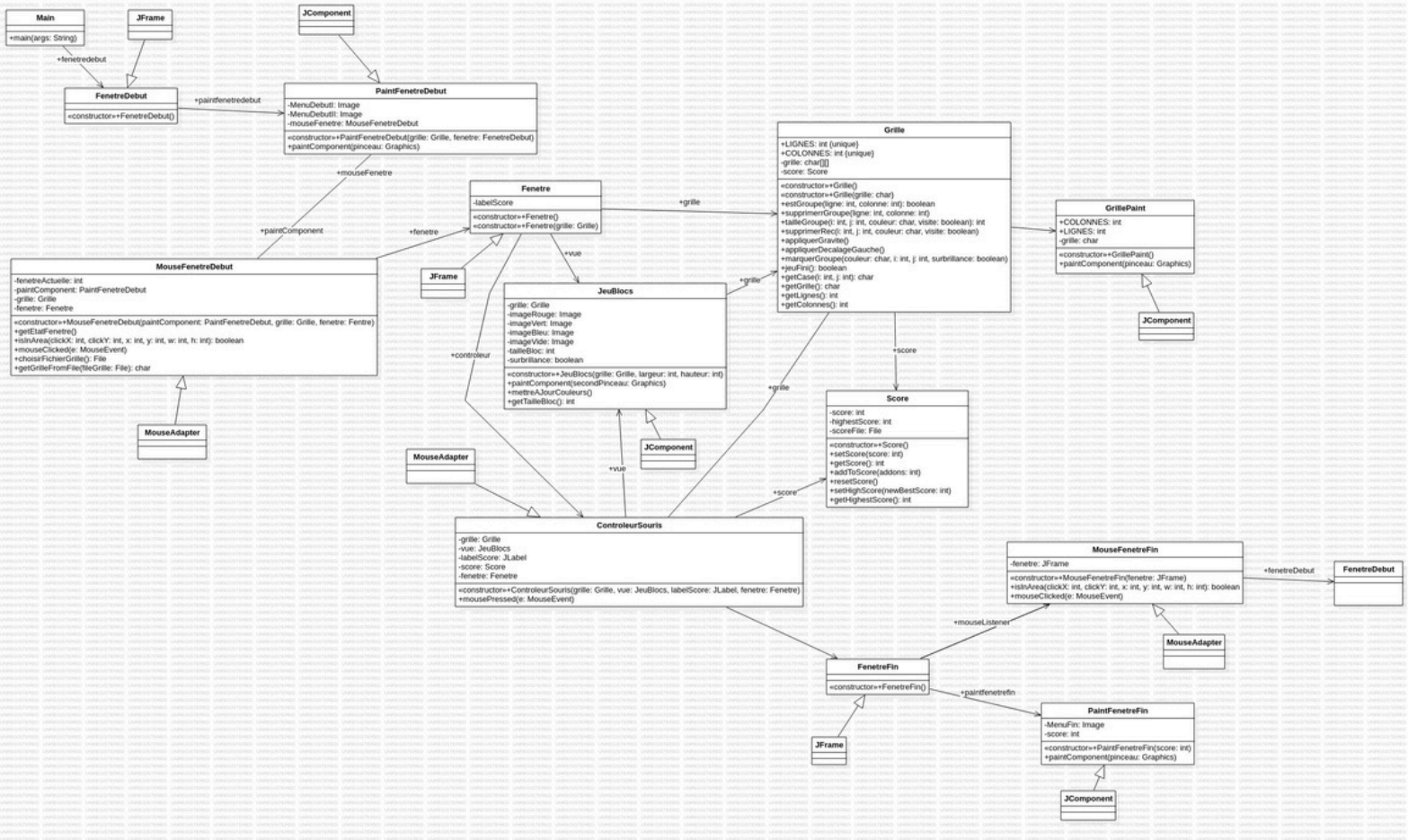
On a décidé de créer une carte mentale pour pouvoir visualiser les différentes étapes du projet et pour qu'on puisse se séparer les tâches plus facilement.

Si vous avez des difficultés à visualiser la carte mentale alors dirigez vous vers ce lien :

https://www.canva.com/design/DAGjy6wcrK0/8oIzC-A06paMjBha5HHpJA/view?utm_content=DAGjy6wcrK0&utm_campaign=designshare&utm_medium=link2&utm_source=uniquelinks&utlId=h8556ff533f

III. Diagramme de classes

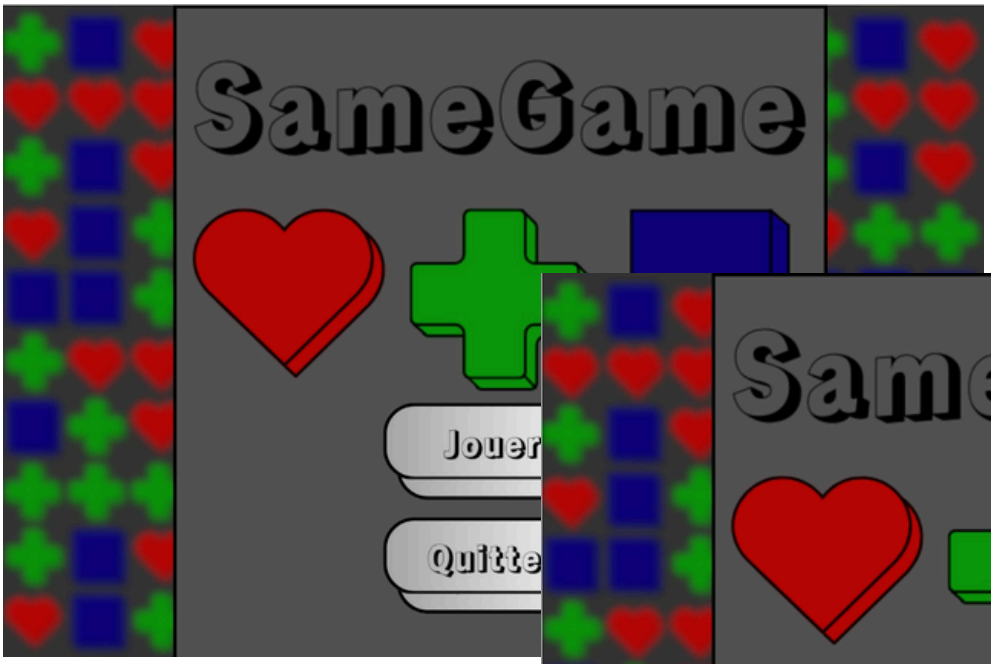
Dans un projet de développement, que ça soit d'application ou web, un diagramme de classes est toujours utile pour le(s) développeur(s) pour la conception en tant que visualisation de différentes classes utilisées. Pour l'utilisateurs, il peut être une aide sur la compréhension du code et de la composition des classes, de leur méthodes et de leur attributs.
Notre diagramme de classe est très détaillé, donc révèle notre code entièrement en UML.



Malheureusement la lisibilité n'est pas au point, mais vous pouvez directement le consulté sur le dépôt git

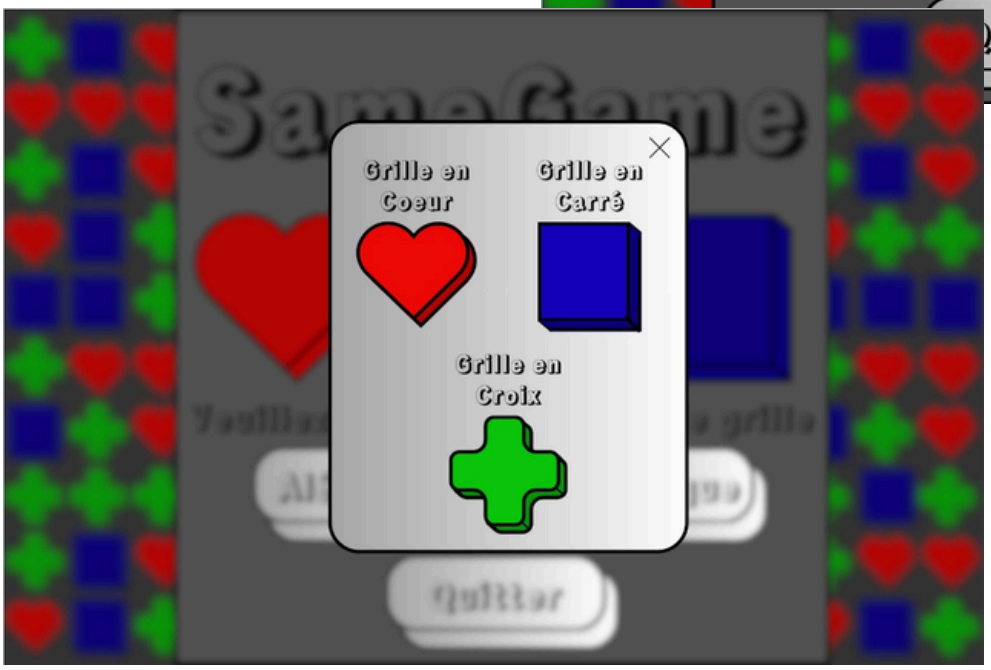
IV. Visualisation du menu de jeu

Dans un jeu, le menu du jeu est une étape essentielle à sa création. C'est cette étape qui attire le jouer/utilisateur à jouer et donc à le découvrir.
C'est aussi là que le joueur passera beaucoup de temps avant de décider de jouer, il est donc important d'avoir un menu beau, esthétique et surtout facile à prendre en main.



Ci dessous, sont des captures d'écran du jeu, trois fenêtres pour les différents choix à faire avant de jouer.

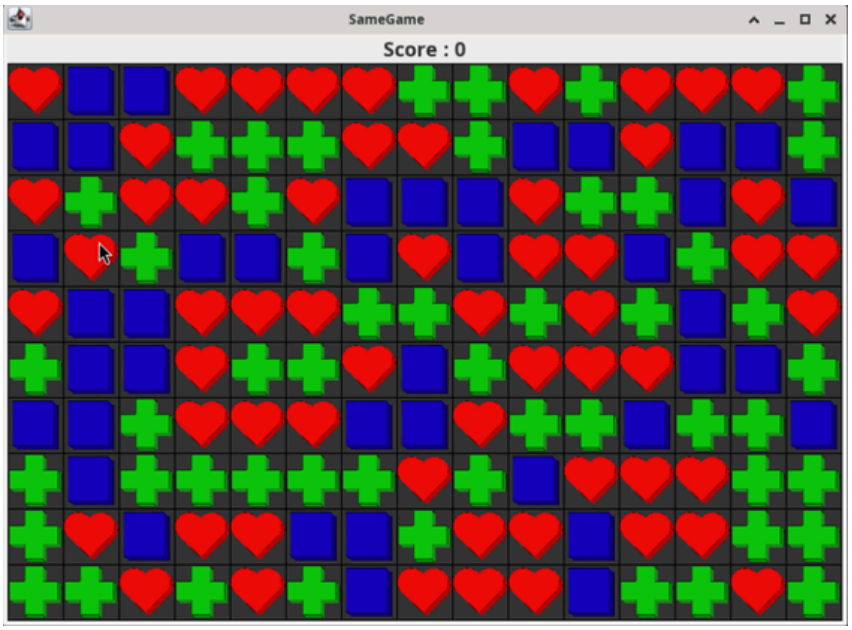
Le premier choix :
si on veut jouer



Le deuxième choix : comment est ce que l'on veut jouer

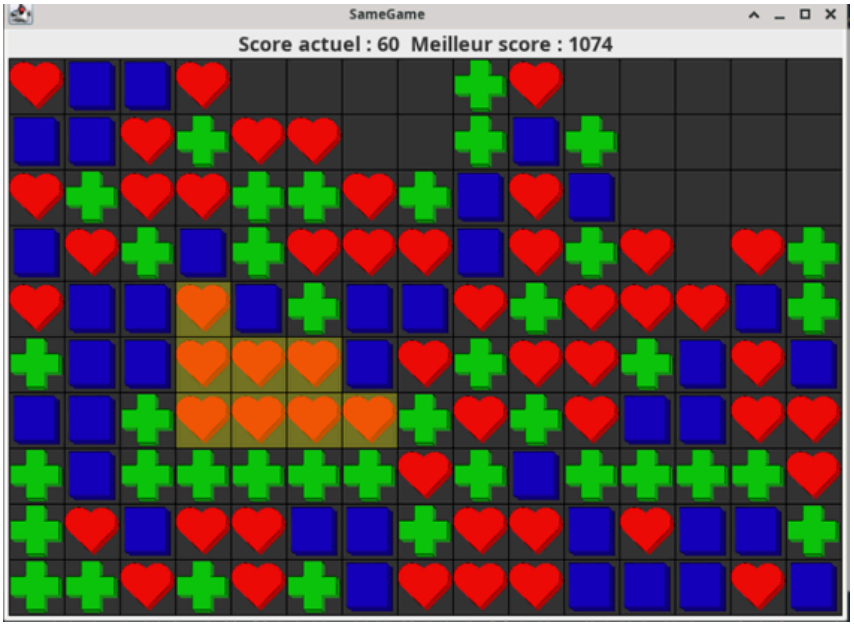
Le troisième choix : sur quel type de grille jouer

V. Grille du jeu



La grille est un tableau 2D représentant les blocs du jeu.
Chaque case contient un bloc de couleur (rouge, vert, bleu) ou est vide. Lors du clic d'un joueur, la grille vérifie si un groupe de blocs adjacents de la même couleur existe, via la méthode estGroupe().

Si un groupe est trouvé, la méthode supprimerGroupe() supprime les blocs et laisse des espaces vides. Après la suppression, un effet de gravité est appliqué grâce à appliquerGravite() puis si une colonne a été entièrement vidée, les colonnes à sa droite sont décalées vers la gauche grace a appliquerDecalageGauche()
La grille est mise à jour après chaque mouvement, et le score est recalculé.



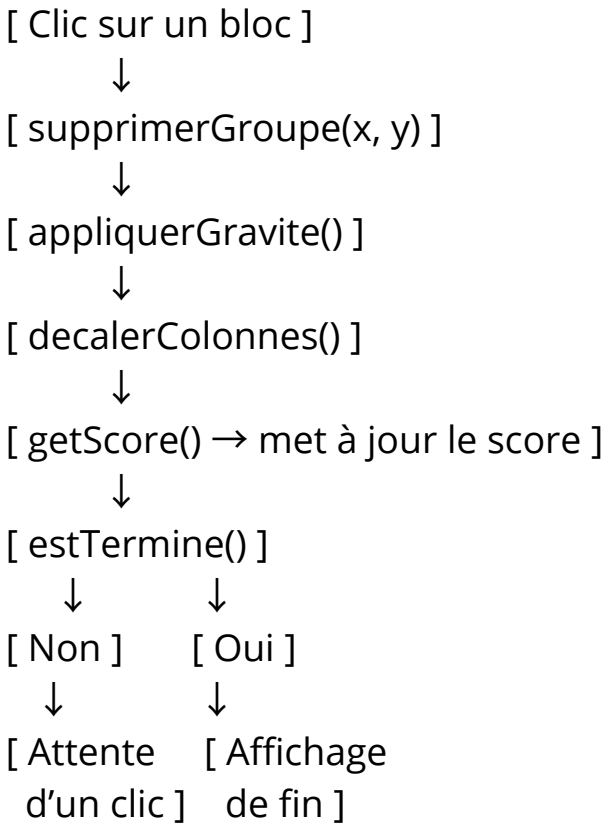
VI. Explication de l’algorithme de recherche de groupe

L’algorithme de recherche de groupe s’appelle estGroupe pour nous.
La méthode estGroupe() vérifie si un groupe de blocs de la même couleur existe à partir d'une position donnée (ligne, colonne).
Elle fonctionne en appelant la méthode tailleGroupe(), qui utilise une approche de recherche en profondeur pour explorer tous les blocs adjacents (horizontaux et verticaux) ayant la même couleur que celui cliqué.
Si un groupe est trouvé et que sa taille est supérieure à 1, cela signifie qu'un groupe valide existe. Si c’est le cas, la méthode retourne true, sinon elle retourne false.

R	**R**	**B**	**B**	**R**
R	**R**	**V**	**V**	**R**
R	**R**	**V**	**B**	**R**
R	**R**	**B**	**B**	**R**
R	**R**	**R**	**R**	**R**

VII. Gestion du jeu

Le jeu est entièrement automatisé pour gérer les interactions du joueur avec la grille. Il repose sur une seule fonction principale dans la classe Grille.
Son objectif : permettre au joueur de supprimer des groupes de blocs de même couleur.
Lorsqu'un joueur clique sur un groupe de blocs, la méthode supprimerGroupe(int x, int y) est appelée pour retirer tous les blocs de même couleur connectés.
Ensuite, la méthode appliquerGravite() fait tomber les blocs restants, et decalerColonnes() décale les colonnes vers la gauche si elles sont vides. Le score est calculé grâce à getScore() selon le nombre de blocs supprimés.
Le jeu détecte la fin automatiquement avec estTermine() : s'il n'y a plus de groupes valides, la partie s'arrête et une fenêtre de fin s'affiche.

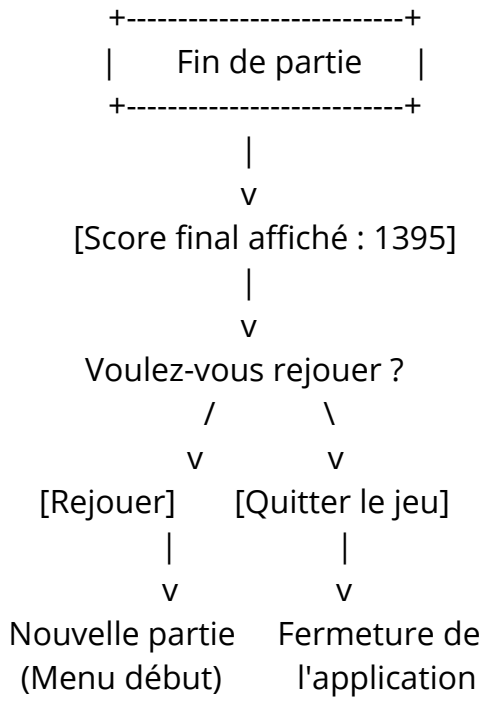


VIII. Menu de fin

Le menu de fin est un élément essentiel pour clôturer la partie de manière claire et intuitive. Il offre un retour visuel sur le score final du joueur et propose à l'utilisateur des actions post-partie.
Ce menu a deux objectifs principaux :

- Informer du résultat de la partie : Afficher un message indiquant que la grille est vide ou que plus aucun coup n'est possible, tout en montrant le score final obtenu.
- Permettre une interaction simple : Proposer deux boutons interactifs pour rejouer une nouvelle partie ou quitter le jeu.

La fonction associée à cette interface est MenuFin, qui reçoit un entier score en paramètre afin d'afficher le résultat obtenu par le joueur :
MenuFin(int score)
Le score est déterminé par le nombre de blocs supprimés au cours de la partie. Plus le joueur a réussi à éliminer de gros groupes de blocs, plus son score sera élevé. Cela permet une évaluation finale claire de la performance du joueur.



IX. Données de l’état d’une partie

Dans SameGame, l'état de la partie est géré par plusieurs données qui suivent l'évolution du jeu.

1. La Grille de Jeu
La grille est une matrice 2D char grille[15][10], où chaque case peut contenir :

- 'R' pour un bloc rouge.
- 'V' pour un bloc vert.
- 'B' pour un bloc bleu.

' ' pour une case vide (quand un groupe de blocs est supprimé).
La grille a une taille de 15 colonnes et 10 lignes.

2. L'etat de la partie
L'état de la partie est un indicateur qui change en fonction des actions :

0 : Jeu initialisé (grille générée).
1 : Un groupe de blocs est détecté et supprimé.
2 : Application de la gravité (les blocs tombent).
3 : Fin du jeu (toutes les cases sont vides).

3. Le score
Le score est comptabilisé chaque fois qu'un groupe de blocs est supprimé. Il est stocké dans int score.

4. Les déplacements et interactions
Quand le joueur clique sur la grille, la position de la souris est utilisée pour déterminer quel groupe de blocs supprimer. Des fonctions comme detecterGroupe(x, y) et supprimerGroupe(groupe) gèrent cette interaction.

5. Le tour de jeu
Le jeu fonctionne avec un système de tours. Le joueur doit sélectionner un groupe de blocs à supprimer à chaque tour. Le int tour détermine quel joueur joue (ou si c’est la fin de la partie).
Interaction des Données
Lorsque le joueur clique sur un groupe de blocs, la grille est mise à jour. Cela affecte l'affichage à l'écran, le score et l'état du jeu. Le but est de vider la grille en supprimant les groupes de blocs adjacents.

X. Structuration des fichiers

Pour la réalisation du Makefile il fallait avoir une idée claire de la structuration du jeu. Par quel classe est ce que l'on commence ou par quel classe est ce que l'on finit.
Le diagramme de classe joue déjà un grand rôle pour la réalisation du Makefile, il nous montre chaque étapes, chaques classes que le code passe pour parvenir au but.

Le code source est réparti dans différents fichiers .java, chacun correspondant à une classe spécifique du programme. Cette séparation s'appuie sur les principes de la programmation orientée objet, où chaque classe a une responsabilité bien définie.

Le point d'entrée du programme est la classe Main, qui instancie l'interface principale du jeu via la classe FenetreDebut qui ensuite va petit à petit instancier la classe Fenetre. Cette dernière constitue le cœur de l'interface graphique pendant la partie. Elle interagit notamment avec la classe Grille, qui représente la matrice de blocs du jeu, et avec JeuBlocs, qui encapsule les règles : suppression de blocs, gestion du score, détection de fin de partie, etc.

Afin de rendre l'expérience utilisateur plus complète, le projet comprend également des fenêtres dédiées à l'accueil (FenetreDebut) et à la fin de partie (FenetreFin). Chacune de ces interfaces repose sur des composants graphiques et des contrôleurs distincts (tels que PaintFenetreDebut, MouseFenetreDebut, etc.), ce qui permet de maintenir une cohérence tout en isolant les comportements spécifiques à chaque étape du jeu.

Le projet repose aussi sur un Makefile qui illustre clairement les dépendances entre les différentes classes. Par exemple, la classe Fenetre dépend de Grille, JeuBlocs et ControleurSouris, tandis que JeuBlocs repose sur Grille et Score. Cette organisation rend la compilation plus efficace : seules les classes modifiées ou dépendantes sont recompilées. Cela reflète également l'architecture logique du projet.
Enfin, cette séparation permet non seulement de faciliter la maintenance et les évolutions futures du code, mais aussi de garantir une meilleure lisibilité et une meilleure testabilité des composants.

En cas d'ajout de nouvelles fonctionnalités (niveaux, effets sonores, etc.), cette structure modulaire s'adapte aisément, ce qui constitue un atout important dans une perspective de développement à long terme.

XI. Conclusion

ATIK Wael :
Durant ce projet, j'ai eu l'opportunité de travailler sur la logique du jeu, l'interface graphique, le débogage , la structure du code. J'avais déjà travaillé avec Emmanuel sur un projet en NSI donc la séparation du travail a été plutôt simple , ayant en plus fait le projet Blokus ensemble, notre cohesion est à son paroxysme.
En tant que gérant de la logique du jeu , j'ai eu à faire le EstGroupé et AppliquerGravite notamment qui furent très compliquées, car au début par exemple, il y avait des erreurs où certains groupes n'étaient pas détectés, ou bien des blocs isolés étaient confondus avec des groupes.

J'ai donc rencontré de nombreuses difficultés à essayer tant bien que mal les erreurs durant 2 jours et 2 nuits.
Il y a eu des moments ou je me disais si réellement un coefficient 40 était important mais au final j'ai réussi a faire un projet qui tient la route et qui marche et j'en suis très fier.

Emmanuel TIAMZON :
Durant ce projet, j'ai travaillé sur plusieurs aspects du jeu SameGame, notamment la logique du jeu, l'interface graphique, ainsi que la structuration du code. J'ai contribué à la mise en place des menus (début et fin), à l'esthétique générale, à la gestion du score, et à l'intégration du JFileChooser pour permettre l'ouverture de fichiers. Une attention particulière a été portée à l'organisation du projet, en structurant les fichiers de manière claire et modulaire grâce à un Makefile efficace.
Ce projet m'a permis de consolider mes compétences en Java, de mieux comprendre l'architecture d'un programme orienté objet.

J'ai appris à mieux organiser mon travail en équipe, à structurer un programme plus complexe, et à relever les défis liés à la logique du jeu, notamment la vérification des conditions de fin de partie ou des blocs adjacents.
Cette situation d'apprentissage m'a donné l'opportunité de renforcer mes bases en développement tout en découvrant l'importance d'une structure de projet claire, surtout lorsqu'on travaille en groupe sur un projet conséquent.