

BAUDRIER Nathan  
MURALITHARAN Lakshman  
Groupe 3

# **Compte-rendu SAE21\_2025 DEV2.1**

## **TABLES DES MATIÈRES:**

### **1) INTRODUCTION**

- 1.1 Règles du jeu
- 1.2 Répartition des tâches

### **2) FONCTIONNALITÉS**

### **3) STRUCTURE DU PROGRAMME**

- 3.1 Diagramme de Classes
- 3.2 Détails des choix

### **4) ALGORITHMES**

### **5) CONCLUSION**

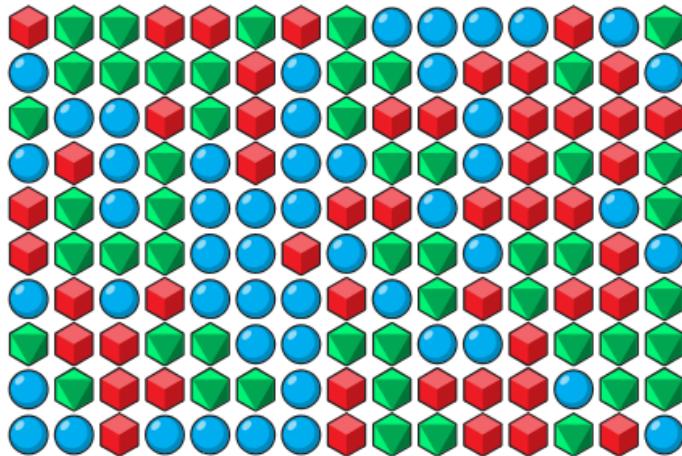
### **6) SOURCES**

## **INTRODUCTION :**

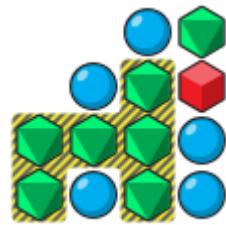
Dans le cadre de la SAé 2.1, nous avions pour consigne de réaliser le jeu **SameGame** en langage JAVA.

### **Règle du jeu :**

- **Objectif** : Vider une grille de blocs colorés (rouge, vert, bleu).



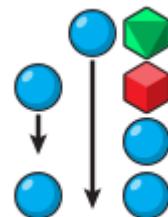
- **Groupe** : Ensemble d'au moins deux blocs adjacents (horizontalement ou verticalement) de même couleur.
- **Interaction** :
  - Survol d'un bloc : le groupe auquel il appartient est mis en évidence.



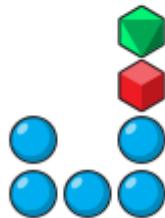
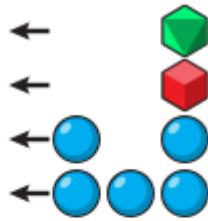
- Clic sur un groupe : les blocs du groupe sont supprimés.

- **Réarrangement :**

- Les blocs au-dessus des espaces vides tombent pour combler les trous.



- Les colonnes vides sont supprimées, décalant les colonnes restantes vers la gauche.



- **Score :**

- Calculé selon la formule :  $(n - 2)^2$ , où n est le nombre de blocs dans le groupe.
- Exemples :
  - 2 blocs : 0 point
  - 3 blocs : 1 point
  - 4 blocs : 4 points
  - 5 blocs : 9 points
  - ...

- **Fin de partie** : Lorsqu'aucun groupe ne peut être supprimé (des blocs isolés peuvent rester).

### **Répartitions des tâches :**

**Nathan** : Il s'occupe de concevoir la grille du jeu SameGame ainsi que son fonctionnement. Cela comprend la mise en place de la structure de données permettant de représenter chaque case de la grille, ainsi que le développement des fonctions nécessaires pour gérer les interactions du joueur avec cette dernière. Il veille également à ce que l'affichage de la grille soit fidèle à l'état réel du jeu, en mettant à jour les éléments visuels en fonction des actions effectuées (sélection de groupes de couleurs, suppression de cases, etc.). Par ailleurs, il organise le code de manière modulaire, en séparant clairement les responsabilités entre les différentes classes (grille, logique de jeu, mise à jour de l'affichage), ce qui facilite la lisibilité, la maintenance et les évolutions futures du projet. Son travail est essentiel pour assurer la cohérence entre les actions du joueur et ce qui est affiché à l'écran, garantissant ainsi une expérience de jeu fluide et intuitive et réalise le diagramme de classes du projet.

**Lakshman** : S'occupe de la création de l'interface utilisateur du projet SameGame, comprenant l'écran d'accueil et l'écran de fin. Il conçoit le menu principal avec un fond personnalisé, un logo, et des boutons permettant d'importer une grille depuis un fichier ou d'en générer une aléatoirement. Il développe également l'écran de fin, avec des boutons pour relancer une partie ou quitter le jeu. Il s'assure que les transitions entre les différentes fenêtres soient fluides, et améliore l'esthétique de l'application en intégrant des icônes graphiques aux boutons. Il met aussi en place des composants réutilisables (comme les boutons personnalisés) pour

une meilleure organisation du code, qu'il structure dans des packages clairs. En plus du développement, Lakshman s'occupe de la rédaction complète du compte rendu du projet. Il y détaille les fonctionnalités, la structure du code, ainsi que la répartition des tâches au sein du groupe. Son travail contribue à offrir une expérience utilisateur intuitive et une documentation claire pour valoriser le projet SameGame.

## **FONCTIONNALITÉS**

Notre jeu dispose des fonctionnalités suivantes :

Menu d'accueil :

Au lancement du programme, on accède au menu suivant :

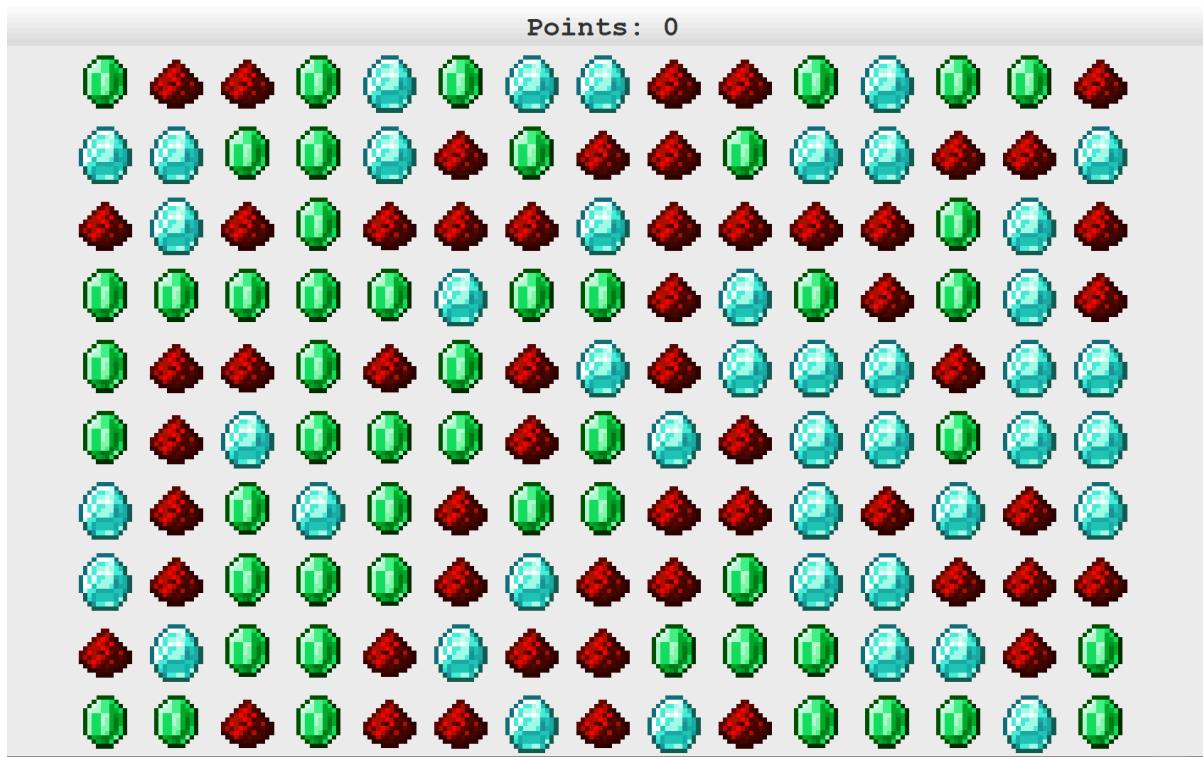


Dès le lancement du jeu, un écran d'accueil s'affiche, offrant au joueur deux options :

- Charger une grille depuis un fichier .txt
- Générer une grille de manière aléatoire

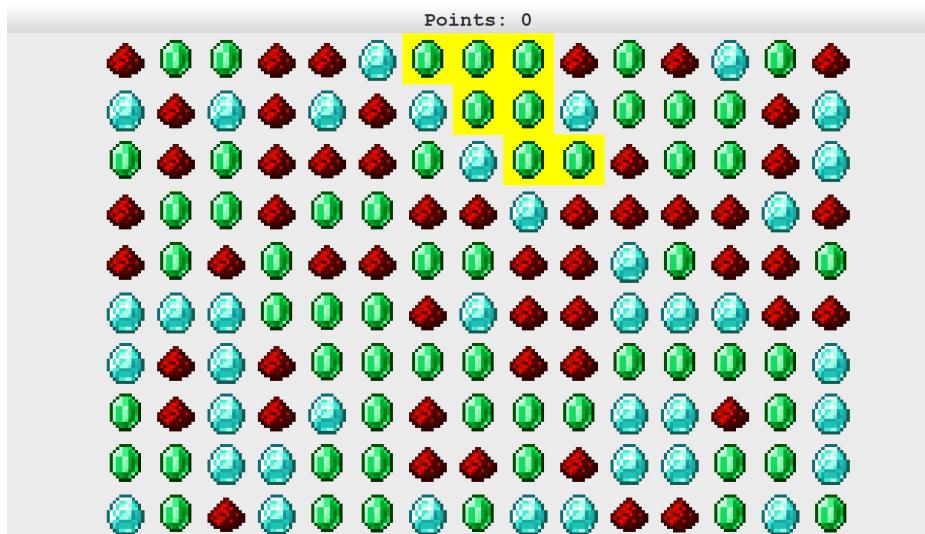
Cet écran inclut également un logo, un fond graphique attrayant, et des boutons interactifs décorés avec des icônes.

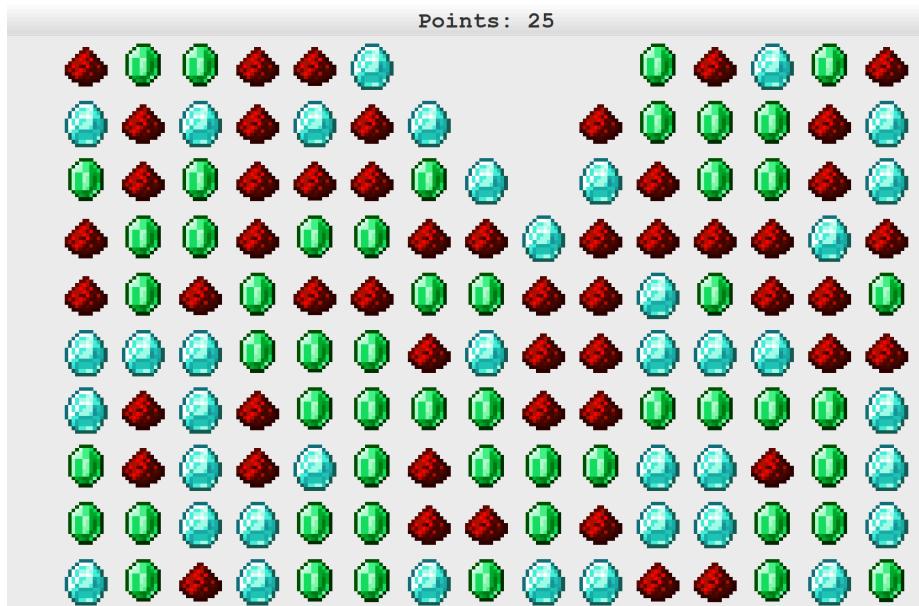
#### Affichage de la Grille :



Le jeu affiche une grille rectangulaire (*15 colonnes et 10 lignes*) composée de cases colorées (minérais). Celle-ci peut être générée de manière aléatoire ou bien chargée à partir d'un fichier texte (.txt).

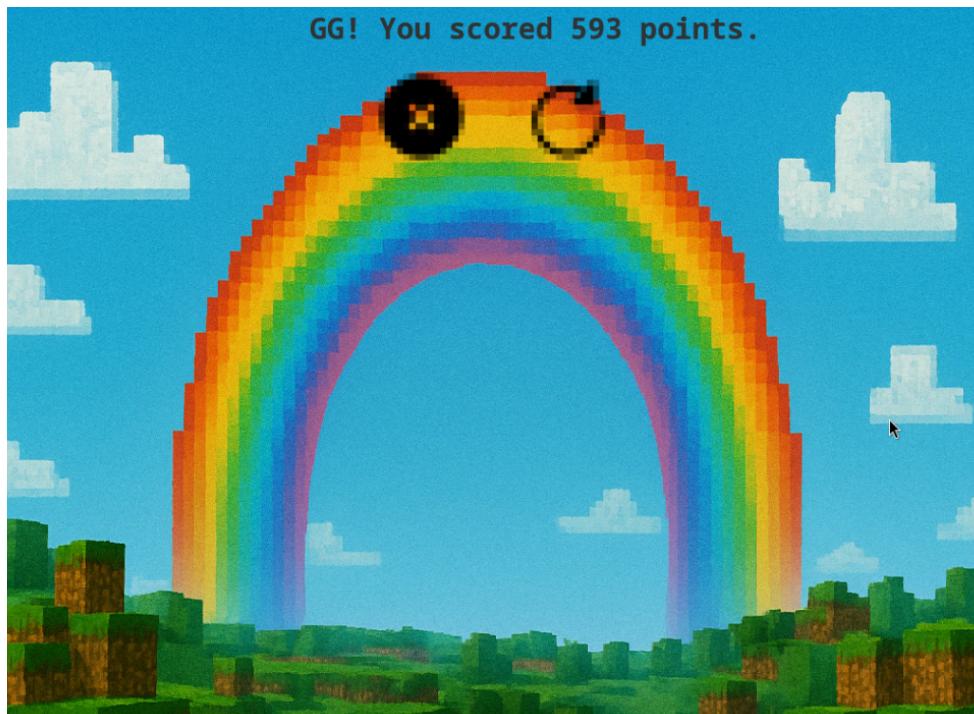
#### Interactions Joueur :





Le joueur peut interagir avec la grille en cliquant sur un groupe de cases adjacentes de même couleur. Quand un groupe est sélectionné, celui-ci est surligné en jaune pour mieux les distinguer. Lorsqu'un tel groupe est sélectionné, toutes les cases concernées disparaissent. Ensuite, un effet de gravité est appliqué : les cases situées au-dessus tombent pour combler les vides laissés. Si des colonnes deviennent entièrement vides, elles sont supprimées, et les colonnes restantes se recentrent automatiquement pour maintenir la grille compacte.

## Écran de Fin :



Une fois qu'aucun coup n'est plus possible, un écran de fin apparaît. Celui-ci propose deux actions au joueur :

- Recommencer une nouvelle partie (Bouton Restart).
- Quitter le jeu (Bouton Exit).

# **STRUCTURE DU PROGRAMME**

Voici le Diagramme de Classes de notre programme :

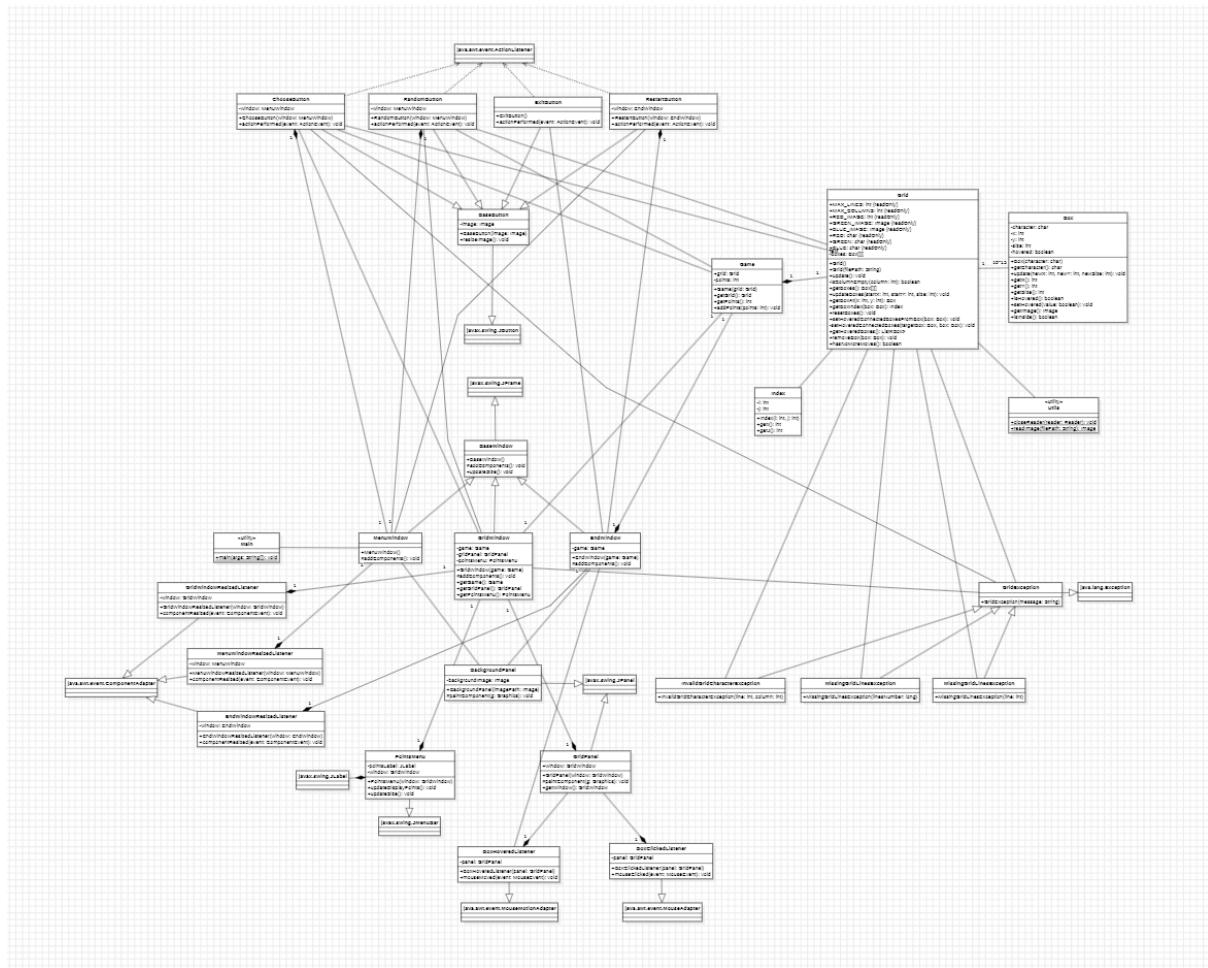


Diagramme disponible sur GIT pour une meilleure lisibilité du diagramme

## Détails des choix :

Le projet a été structuré avec rigueur afin de garantir lisibilité, modularité et maintenabilité du code. L'organisation repose sur une séparation claire des responsabilités grâce à l'utilisation de **packages** spécifiques, ce qui permet de mieux organiser le grand nombre de classes nécessaires au fonctionnement du jeu.

## main

- **Main.java** : Point d'entrée du programme. Lance l'interface principale (**MenuWindow**).

## components.buttons

- **ChooseButton.java** : Bouton pour importer une grille à partir d'un fichier **.txt**.
- **ExitButton.java** : Bouton pour quitter l'application.
- **RandomButton.java** : Bouton pour générer une grille aléatoire.
- **RestartButton.java** : Bouton qui relance le jeu en retournant au menu principal.
- **BaseButton.java** : classe mère commune pour créer des boutons personnalisés avec une image. Elle permet d'afficher un bouton avec fond transparent, sans bordure, et redimensionnable dynamiquement selon la taille de la fenêtre.

## components.events

- **BoxClickedListener.java** : Gère les clics de l'utilisateur sur les cases.
- **BoxHoverListener.java** : Gère le survol des cases.
- **GridWindowResizedListener.java** : Gère le redimensionnement de la fenêtre de jeu.
- **MenuWindowResizedListener.java** : écouteur spécifique pour la fenêtre du menu principal, permettant d'ajuster dynamiquement les composants (boutons, logo, etc.).
- **EndWindowResizedListener.java** : écouteur de redimensionnement pour la fenêtre de fin de partie. Permet de garder les boutons bien centrés même lorsque la fenêtre est redimensionnée.

## components.panels

- **BackgroundPanel.java** : Panel personnalisé avec une image de fond.
- **GridPanel.java** : Conteneur visuel qui affiche la grille du jeu.
- **PointsMenu.java** : Affiche les points du joueur pendant la partie.

## components.windows

- **BaseWindow.java** : Classe de base pour toutes les fenêtres (extends JFrame).
- **MenuWindow.java** : Fenêtre principale avec les boutons "aléatoire" et "importer".
- **GridWindow.java** : Fenêtre de jeu avec grille interactive.
- **EndWindow.java** : Fenêtre affichée à la fin d'une partie (avec boutons restart/exit).

## exceptions.grid

- **GridException.java** : Exception générique pour les erreurs liées à la grille.
- **InvalidGridCharacterException.java**,  
**MissingGridCharactersException.java**,  
**MissingGridLinesException.java** : Exceptions spécifiques au format du fichier de grille.

## game

- **Game.java** : Gère la logique de jeu (points, état, grille...).
- **Grid.java** : Représente la grille du jeu, gère l'affichage et les suppressions de groupes.
- **Box.java** : Représente une case (bloc) de la grille (couleur, position, état).

## utils

- **Index.java** : Structure pour représenter une position (ligne, colonne) dans la grille.
- **Utils.java** : Fonctions utilitaires (génération aléatoire, etc.).

L'ensemble du code source est écrit en **anglais**, aussi bien pour les noms de classes, méthodes, que pour la documentation. Ce choix s'inscrit dans une démarche professionnelle : il facilite la compréhension du code dans un contexte international, et prépare à la lecture et à l'écriture de code en entreprise.

Pour les aspects visuels, une attention particulière a été portée à **l'alignement et au centrage graphique** : tous les éléments (*boutons, logos, composants graphiques*) sont **toujours centrés automatiquement**, quelle que soit la taille de la fenêtre. Cela contribue à une interface utilisateur fluide, cohérente et esthétiquement stable.

Un **Makefile** a été mis en place afin d'automatiser la compilation, l'exécution et la génération de documentation du projet **SameGame**. Cela permet de simplifier l'utilisation du projet au quotidien, notamment pendant les phases de test ou de développement.

Le fichier définit plusieurs variables pour organiser le projet, comme les chemins vers les sources (**src**), les fichiers compilés (**bin**) ou encore la documentation (**doc**). La classe principale du programme est également définie (**Main**), ce qui permet de lancer le jeu sans avoir à spécifier manuellement le chemin complet.

### **Différentes commandes sont disponibles :**

- **make compile** permet de compiler tous les fichiers **.java**.

- `make run` compile puis exécute automatiquement le programme.
- `make doc` génère la documentation JavaDoc.
- `make clean` supprime les fichiers .class et la documentation.
- `make fclean` supprime entièrement les dossiers bin et doc.
- `make re` fait un nettoyage complet suivi d'une recompilation.
- `make` ou `make all` enchaîne compilation et exécution automatiquement.

L'utilisation du Makefile nous a permis de gagner en efficacité tout au long du développement. Il permet également de standardiser l'utilisation du projet, ce qui est particulièrement utile dans un travail de groupe ou lors d'une soutenance. Grâce à lui, le projet peut être compilé et lancé facilement, sans configuration supplémentaire, à l'aide d'une seule ligne de commande.

Enfin, le choix d'utiliser une récursivité pour l'algorithme de détection des groupes de blocs (recherche de composantes connexes) a été motivé par des raisons de lisibilité, de clarté et de performance. La solution récursive est plus concise, naturelle à lire pour ce type de problème, et plus simple à maintenir qu'une version itérative avec pile manuelle.

## **ALGORITHMES :**

L'algorithme utilisé pour identifier les groupes de blocs dans SameGame repose sur une **approche récursive**. Lorsqu'un joueur survole ou clique sur une case de la grille, le programme cherche à détecter tous les blocs **adjacents** (haut, bas, gauche, droite) qui ont **la même couleur** que la case d'origine.

L'algorithme commence à partir de la case sélectionnée et explore ses voisins immédiats. Si un voisin possède la même couleur et n'a pas encore été visité, il est ajouté au groupe, puis ses propres voisins sont explorés de la même manière. Ce processus se répète récursivement jusqu'à ce que tous les blocs connectés soient identifiés.

Chaque bloc détecté est marqué (par exemple avec un indicateur visuel lors du survol), ce qui permet soit de **mettre en évidence le groupe**, soit de **le supprimer** lorsque l'utilisateur clique dessus. Si le groupe est suffisamment grand (au moins deux blocs), il est supprimé, et les blocs restants tombent pour combler les vides, selon les règles du jeu.

Le choix d'utiliser une fonction récursive plutôt qu'un algorithme itératif a été motivé par des raisons de **lisibilité**, de **clarté** du code, et de **facilité de réutilisation**. Cette approche s'adapte parfaitement à la nature du problème, qui est une recherche de composantes connexes dans une grille bidimensionnelle.

## **CONCLUSION**

**Nathan BAUDRIER** : Ce projet a été une expérience particulièrement enrichissante et formatrice. Il m'a permis de mettre en pratique toutes les notions apprises en cours, tout en approfondissant mes connaissances en programmation. Le développement du jeu a également été l'occasion d'expérimenter le travail en équipe, un aspect essentiel dans ce type de projet. Grâce à l'utilisation de Git, j'ai considérablement amélioré mon organisation et appris à gérer efficacement les versions de code, ce qui a facilité la collaboration et le suivi des progrès. En somme, ce projet m'a permis de renforcer mes compétences techniques et humaines, tout en découvrant les défis et la satisfaction liés à la création d'une application ludique et interactive.

**Lakshman MURALITHARAN** :

Ce projet a été très enrichissant pour moi. Cela m'a permis de mettre en pratique ce que j'ai appris en cours en Java. En travaillant sur ce projet, j'ai pu approfondir ma compréhension des différents aspects de la programmation en Java, notamment les notions d'exceptions, d'évènements que je trouvais difficiles. Grâce à l'aide de mon camarade, j'ai pu surmonter certaines difficultés et mieux comprendre ces concepts qui n'était pas très clair pour moi. Sur le plan organisationnel, le travail en équipe a été une expérience positive. Nous avons bien structuré notre travail en utilisant GIT qui nous a permis de travailler efficacement et fluidement.

Cette expérience n'est que bénéfique car elle m'a permis de progresser et de travailler en équipe.

## **Sources :**

Icon :

[https://www.streamlinehq.com/?ref=Awwwards&utm\\_source=awwwards-honors&utm\\_medium=partner&utm\\_campaign=ultimate-icon-library&tab=all&search=import+file](https://www.streamlinehq.com/?ref=Awwwards&utm_source=awwwards-honors&utm_medium=partner&utm_campaign=ultimate-icon-library&tab=all&search=import+file)

Logo & Background : <https://chatgpt.com>