

Rapport de Projet : Démineur en Java (SAE21_2025)

Implémentation du Jeu du Démineur en langage Java

Auteurs : Youssef Amrouchi et Baptiste Lecointre

12 avril 2026

Table des matières

Rapport de Projet : Démineur en Java (SAE21_2025)	1
1. Introduction	3
2. Description des fonctionnalités	4
3. Présentation de la structure du programme	7
3.1. Fichiers présents dans le model.....	7
3.2. Fichiers liés aux Vues	7
3.3. Fichiers liés aux Contrôleurs.....	8
3.4. Fichier de compilation.....	8
4. Explication du mécanisme de sauvegarde d'une partie	9
5. Exposition de l'algorithme permettant de révéler plusieurs cases.....	10
6. Conclusion personnelle	12

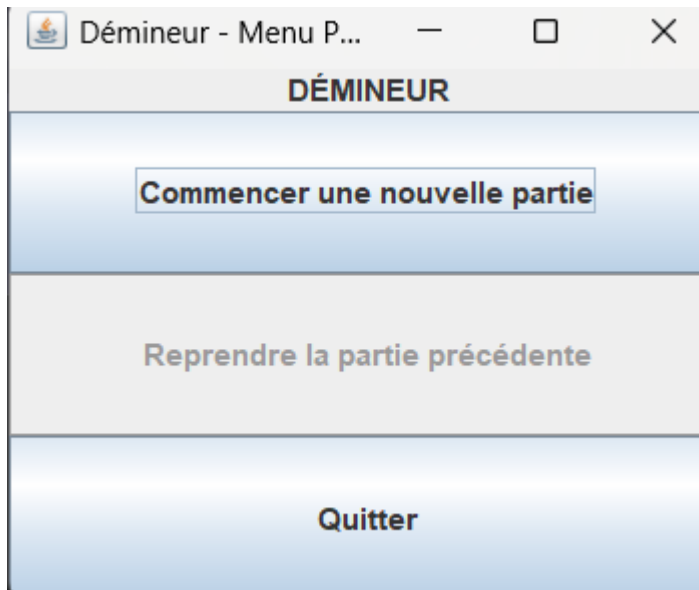
1. Introduction

Dans le cadre de notre projet de programmation, on a été chargés de concevoir et développer un jeu de Démineur classique en langage Java. Le but de ce projet est de consolider nos acquis en programmation orientée objet, en conception d'interfaces graphiques (avec la bibliothèque Swing) et en algorithmique. Ce rapport présente les fonctionnalités de notre jeux, l'architecture de ce dernier que l'on a choisie pour structurer nos fichiers, ainsi que le code permettant le bon fonctionnement du jeu.

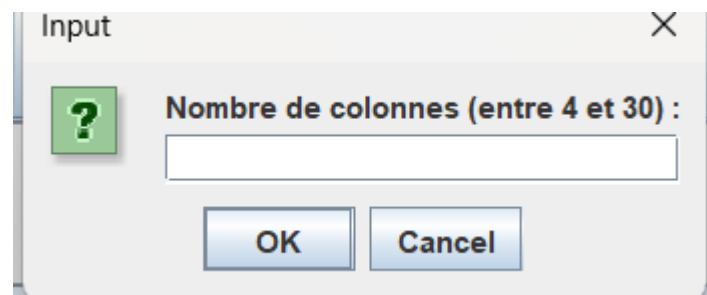
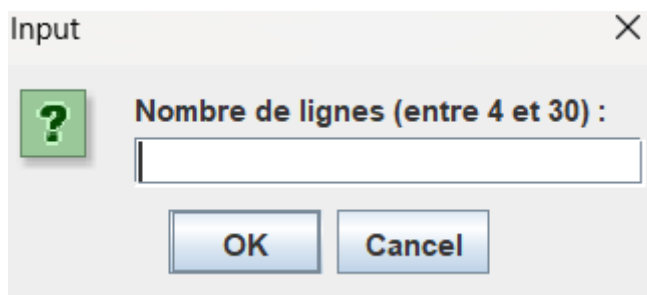
2. Description des fonctionnalités

Notre programme respecte l'ensemble des règles classiques du Démineur et intègre les fonctionnalités demandées suivantes :

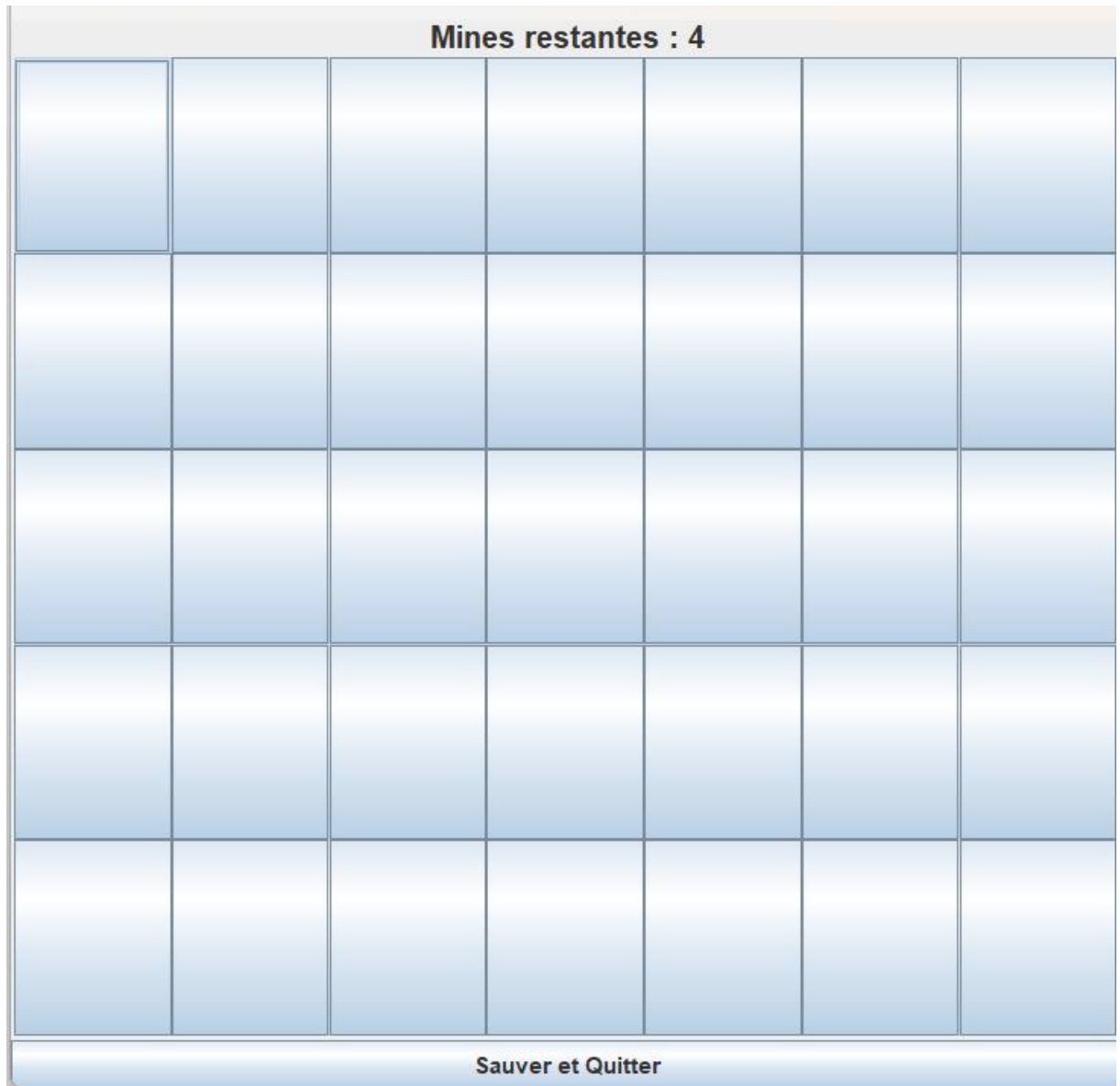
- **Menu d'accueil** : Au lancement, on accède à un menu permettant de commencer une nouvelle partie, de reprendre une partie sauvegardée ou de quitter. L'option de reprise est grisée si aucune sauvegarde n'est trouvée.



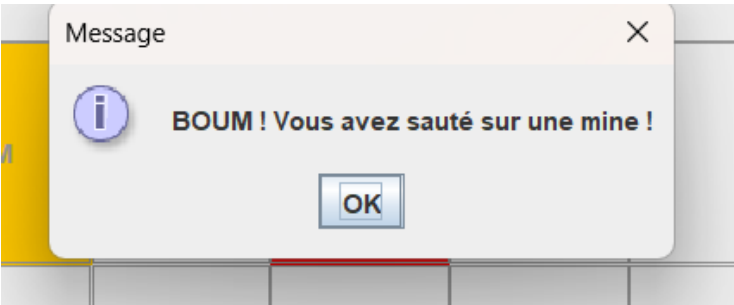
- **Paramétrage de la partie** : Lorsqu'on lance une nouvelle partie, on peut saisir le nombre de lignes, de colonnes et le nombre total de mines souhaitées via des petites fenêtres.



- **Interface de jeu :** L'écran de jeu affiche la grille générée aléatoirement, ainsi qu'un compteur dynamique indiquant le nombre de mines restantes à trouver (calculé en soustrayant le nombre de marqueurs placés au nombre total de mines).



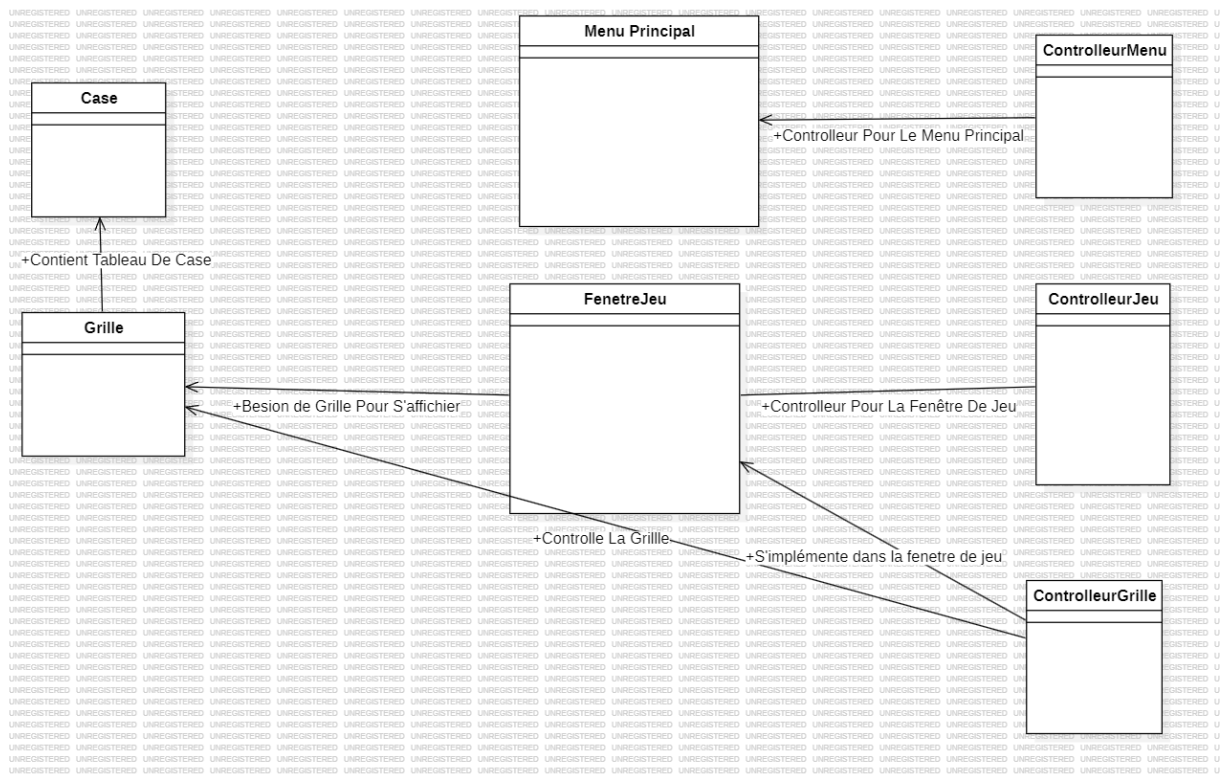
- **Interactions :** On peut faire un clic gauche pour révéler une case, et un clic droit pour alterner entre les marqueurs (rien, ★ pour une mine certaine, et ? pour un doute).
- **Fin de partie :** En cas de victoire ou de défaite, des messages s'affichent. Lors d'une défaite, on révèle clairement la mine qui a explosé, les mines non trouvées, et les erreurs de marquage. Le joueur peut ensuite retourner au menu via un bouton dédié.



M	3	2	1	1	
M	M		BOUM	1	
2	2	2	1	1	

3. Présentation de la structure du programme

Pour structurer notre application, on a suivi une logique de séparation des responsabilités L'ensemble des classes a été commenté pour permettre la génération de la documentation Javadoc. Voici le rôle de chaque fichier composant notre projet :



3.1. Fichiers présents dans le model

- **Case.java** : Bien que non détaillée dans ce rapport, cette classe modélise une case individuelle. Elle stocke l'état logique de la cellule (minée ou non, révélée ou cachée, type de marqueur posé, et nombre de mines voisines).
- **Grille.java** : C'est le cœur logique du jeu. Ce fichier gère le tableau à deux dimensions regroupant toutes les cases. C'est ici que l'on place aléatoirement les mines lors de l'initialisation, que l'on précalcule le nombre de mines voisines pour chaque case, et que l'on vérifie si les conditions de victoire sont remplies. Cette classe est sérialisable pour permettre la sauvegarde.

3.2. Fichiers liés aux Vues

- **MenuPrincipal.java** : Gère la fenêtre d'accueil de l'application. On y définit la mise en page (BorderLayout et GridLayout) et l'apparence des boutons de navigation (Nouvelle partie, Reprendre, Quitter).

- **FenetreJeu.java** : Gère l'affichage de la partie en cours. Elle dessine la grille sous forme de boutons, affiche le compteur de mines restantes et le bouton de sauvegarde. Elle contient également les méthodes permettant de rafraîchir l'aspect visuel d'une case (texte, couleur) en fonction de son état logique, notamment lors d'une défaite pour afficher les erreurs de marquage.

3.3. Fichiers liés aux Contrôleurs

- **ControleurMenu.java** : Implémente l'interface ActionListener. Il écoute les clics sur les boutons du menu principal. C'est lui qui demande à l'utilisateur les dimensions de la grille et qui tente de lire le fichier de sauvegarde si le joueur choisit de reprendre sa partie.
- **ControleurJeu.java** : Gère les actions globales pendant la partie, principalement le bouton situé en bas de l'écran. Selon l'état du jeu, il permet soit de sauvegarder et quitter l'application, soit de retourner au menu principal si la partie est déjà terminée.
- **ControleurGrille.java** : Implémente l'interface MouseListener pour détecter de manière précise les clics de souris sur chaque bouton de la grille. Il différencie le clic gauche pour révéler une case du clic droit pour marquer une case et demande à la Vue de se rafraîchir en conséquence. Il gère aussi l'apparition des boîtes de dialogue de victoire ou de défaite.

3.4. Fichier de compilation

- **Makefile** : Ce fichier définit les règles de compilation de notre projet. Il établit les dépendances entre les différents fichiers Java pour s'assurer que tout soit compilé dans le bon ordre à l'aide de la commande make, et permet de lancer le jeu via la commande make run.

4. Explication du mécanisme de sauvegarde d'une partie

- Pour permettre à un joueur de mettre sa partie en pause et de la reprendre plus tard, on a mis en place un système de sauvegarde reposant sur la sérialisation des objets en Java.
- Au lieu d'enregistrer manuellement l'état de chaque case dans un fichier texte. On a mis en place un système qui lorsqu'on clique sur le bouton "Sauver et Quitter", le programme ouvre un flux d'écriture vers un fichier binaire local (nommé sauvegarde.dat). On y injecte directement l'objet grille dans sa globalité, ce qui capture instantanément l'emplacement des mines, les cases révélées et les drapeaux posés, avant de fermer l'application.
- À l'inverse, si le joueur clique sur "Reprendre la partie précédente" au menu d'accueil, le programme tente d'ouvrir un flux de lecture sur ce même fichier. S'il existe et n'est pas corrompu, le programme reconstitue l'objet grille en mémoire, l'associe à une nouvelle fenêtre de jeu, et force l'interface visuelle à se synchroniser avec cet état logique récupéré.

5. Exposition de l'algorithme permettant de révéler plusieurs cases

L'une des mécaniques fondamentales du Démineur est lorsqu'on révèle une case qui n'est adjacente à aucune mine, le jeu doit automatiquement révéler toutes ses cases voisines, et répéter l'opération si ces voisines sont elles-mêmes vides.

Pour implémenter ce comportement au sein de la classe Grille, on a opté pour un algorithme récursif. Cet algorithme prend en paramètres les coordonnées (ligne et colonne) de la case à traiter. Voici son fonctionnement étape par étape :

```
public boolean revelerCase(int l, int c) {
    Case caseCliquee = this.plateau[l][c];
    if (caseCliquee.isRevelee() || caseCliquee.getMarqueur() == 1) return false;
    caseCliquee.setRevelee(true);
    if (caseCliquee.isMinee()) return true;

    if (caseCliquee.getNbMinesVoisines() == 0) {
        for (int di = -1; di <= 1; di++) {
            for (int dj = -1; dj <= 1; dj++) {
                int voisinL = l + di;
                int voisinC = c + dj;
                if (voisinL >= 0 && voisinL < this.lignes && voisinC >= 0 && voisinC < this.colonnes) {
                    revelerCase(voisinL, voisinC);
                }
            }
        }
    }
    return false;
}
```

Conditions d'arrêt (Cas de base) : Au début de la méthode, on vérifie si la case ciblée est déjà révélée ou si elle est protégée par un marqueur « étoile » (★). Si c'est le cas, l'algorithme s'arrête immédiatement pour éviter de tourner en boucle infinie ou de dévoiler une case que le joueur a verrouillée.

Révélation : Si les conditions sont bonnes, on change le statut de la case pour la marquer comme étant « révélée ».

Vérification de la défaite : Si la case que l'on vient de révéler contient une mine, la méthode renvoie un signal positif pour indiquer l'explosion et la fin de la partie.

Appel récursif (La propagation) : C'est ici que la réaction en chaîne se produit. On vérifie si le nombre de mines voisines de la case actuelle est strictement égal à zéro. Si c'est le cas, on utilise deux boucles imbriquées (variant de -1 à +1) pour parcourir virtuellement les 8 cases adjacentes (diagonales incluses).

Contrôle des limites : Avant de traiter un voisin, on s'assure par sécurité que ses coordonnées calculées se trouvent bien à l'intérieur des limites de la grille (supérieures ou égales à zéro, et inférieures au nombre maximum de lignes et de colonnes).

Récursivité : Pour chaque voisin valide, la méthode s'appelle elle-même en se passant les coordonnées de ce voisin.

Grâce à ce mécanisme d'auto-appel, l'algorithme se propage de proche en proche comme une tache d'huile. Il parcourt la grille et ne s'arrête que lorsqu'il rencontre les "bords" de la zone vide, c'est-à-dire des cases possédant au moins une mine dans leur voisinage immédiat (affichant un chiffre de 1 à 8).

[Insérer ici une capture d'écran montrant le résultat d'un grand clic révélant une vaste zone vide bordée de chiffres, ou un schéma logique de l'algorithme récursif]

6. Conclusion personnelle

- **Youssef Amrouchi** : Ce projet a été très formateur dans mon processus d'apprentissage de java et de l'organisation des classes d'un gros projet, ça n'a pas été facile d'arriver un résultat pareil mais si c'était à refaire je le referai.
- **Baptiste Lecointre** : Le développement du Démineur en java m'a permis de mieux comprendre le système d'actionneur et la complexité de ce dernier, je me suis grandement amélioré dans la création de classe sur Star UML, ce projet m'a donné du fil à retordre mais il fut une très bonne expérience.