Rapport Projet 2.1 : Démineur

Nom des membres du binôme :
Les membres de notre binôme sont :
Aubin TOURAIS et Romain BESSON.

Introduction avec une brève description du sujet :
Le sujet de cette SAE2.1 est un « Démineur » qui est un jeu dont le but est de trouver où se situent des mines cachées dans une grille représentant un champ de mines virtuel, avec pour seule indication le nombre de mines dans les zones adjacentes.

Le nombre de lignes et le nombre de colonnes sont compris entre 4 et 30, mais la grille n'est pas nécessairement carrée. Chaque case peut être révélée ou cachée. Les cases cachées peuvent être minées ou pas. Initialement, toutes les cases de la grille sont cachées.

Pour cela, le joueur devra placer un drapeau/étoile selon là où il pense qu'une mine se trouve.
Le joueur gagne la partie quand il a révélé toutes les cases non minées.

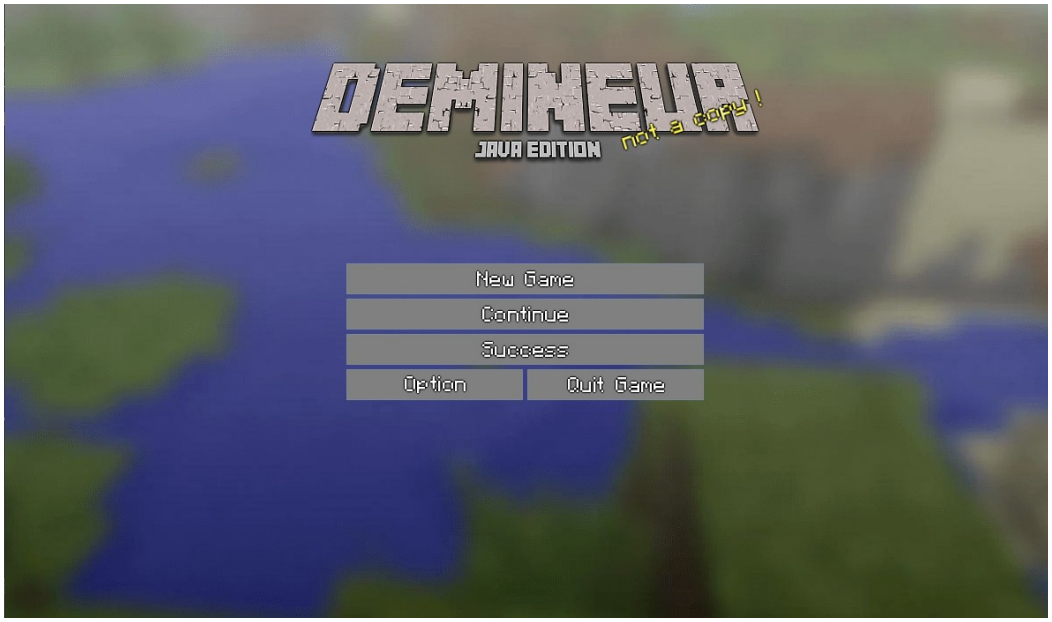
L'objectif ?

Ne pas tomber sur une seule mine !

Description des fonctionnalités de notre programme, avec des captures d'écran :

I. Interface menu de notre Demineur :

Vous pourrez découvrir au lancement du jeu, un menu personnalisé sous la forme de « minecraft ».
Avec un menu fonctionnel, tout en gardant sa fonctionnalité principale.
Cliquez sur « New Game » pour démarrer une partie, ou bien alors sur « Continue » si vous souhaitez reprendre depuis votre dernière sauvegarde.



II. Génération de la grille :

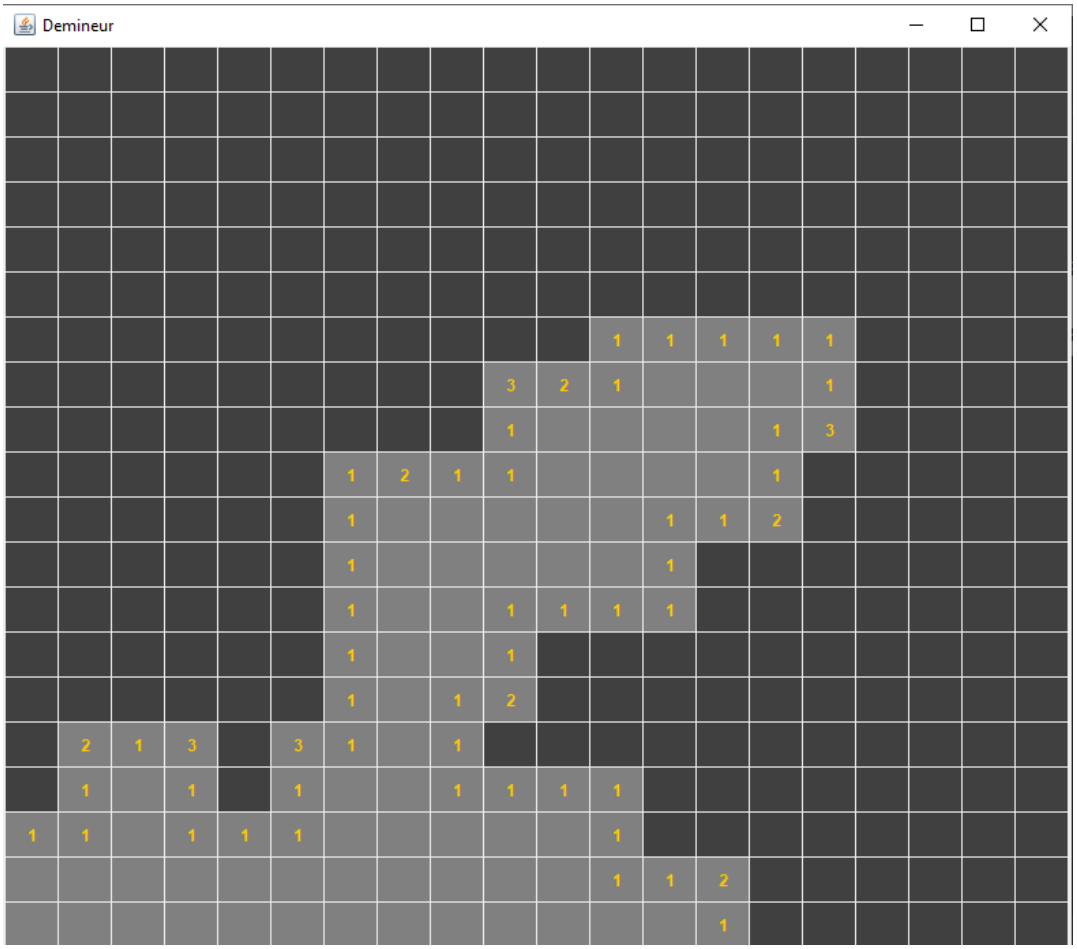
Une fois le lancement d’une partie, il faudra avant tout sélectionner le nombre de cases souhaitées en hauteur ainsi qu’en largeur, accompagnée du nombre de mines désirées.
Maintenant, jouez en cliquant sur « Create New Grid ».

III. Lancement de la Partie :

Vous voilà désormais tout à fait capable de jouer une partie de démineur, clique GAUCHE pour dévoiler une case, et clique DROIT pour placer un drapeau/étoile.

Mais attention, si vous tombez sur ne serait-ce qu'une mine, gare à vous...

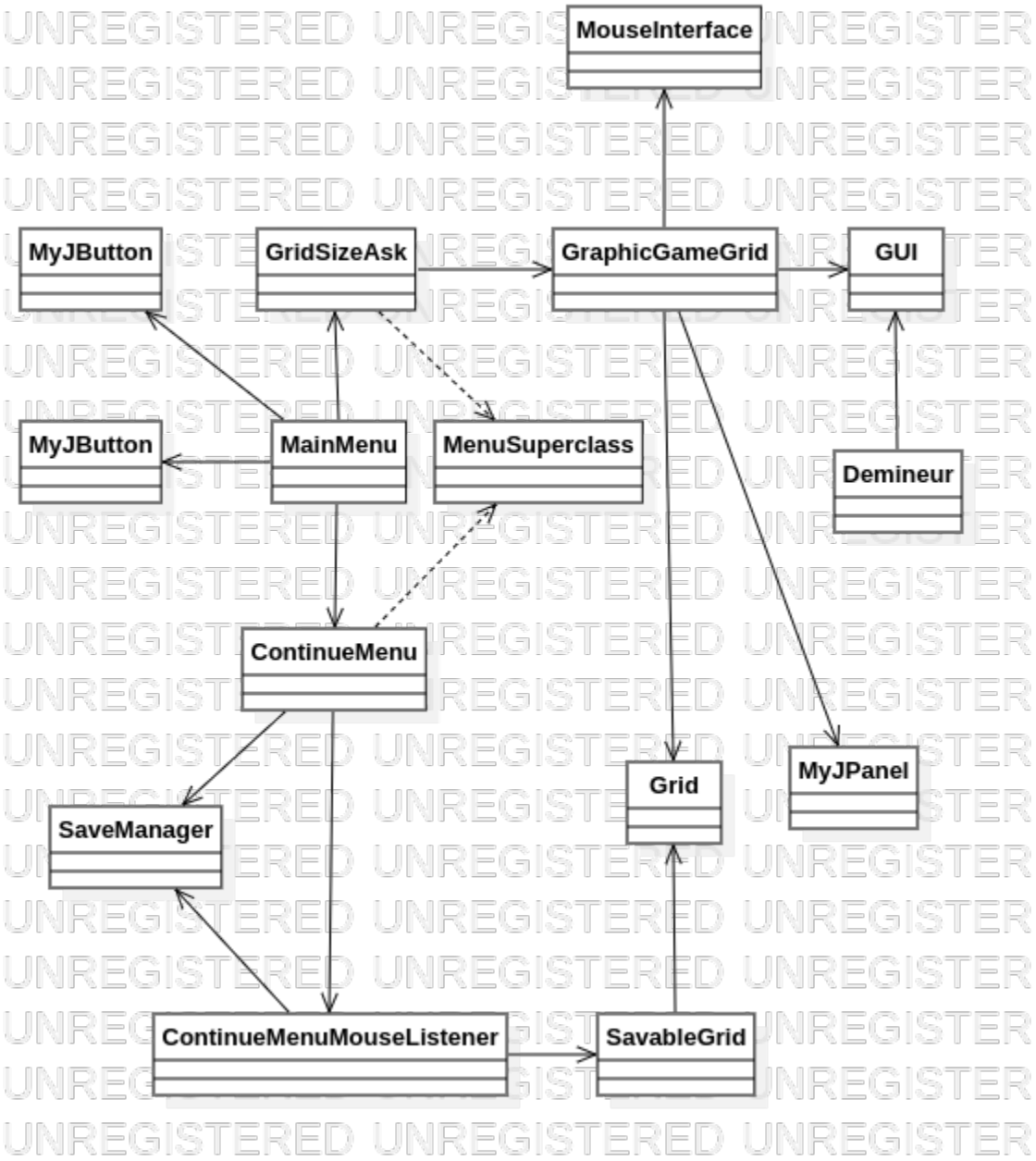
Notre demineur cache quelques surprises, ceci est une conseil, ne tombez pas dedans.



Une fois que vous souhaitez interrompre une partie, vous pouvez si vous le désirez, cliquer sur la croix située tout en haut à droite de votre fenêtre, un message de sauvegarde vous sera proposé.
Vous pourrez ainsi, à tout moment, revenir sur votre partie et continuer d'y jouer.

Cela vous permettra de fiare une pause et votre partie ne sera pas perdue !

Présentation de la structure de notre programme, avec un diagramme de classe :



Explication du mécanisme de sauvegarde :

Pour la sauvegarde, cela créer un fichier, puis sa transforme l'objet à sauvegarder en un paquet d'octet que la machine est capable de retransformer en objet lors du chargement de la sauvegarde.

Algorithme permettant de révéler plusieurs cases :

I. Algorithme complet :

Vous pouvez retrouver ci-dessous, l’intégralité de l’algorithme permettant de révéler plusieurs cases. Puis nous vous avons couper celui-ci en 3 parties avec chacune, leur fonctionnement et leur rôle dans notre code.

```
public static void squareReveal(int[][] viewGrid0, Grid grid0, int x0, int y0){
    int nbMine = 0, xNeighSquare, yNeighSquare;

    for(int i = 0; i < 9; i++){
        xNeighSquare = x0+COOR_NEIGHBOR[i][0];
        yNeighSquare = y0+COOR_NEIGHBOR[i][1];

        if(xNeighSquare >= 0 && yNeighSquare >= 0 && xNeighSquare < grid0.getGridWidth() && yNeighSquare < grid0.getGridHeight()){
            if(isThereAMine(grid0, xNeighSquare, yNeighSquare) && grid0.getGrid()[xNeighSquare][yNeighSquare] != DISCOVERED_SQUARE){
                nbMine++;
            }
        }
    }

    grid0.setGrid(x0, y0, DISCOVERED_SQUARE);
    viewGrid0[x0][y0] = nbMine;

    if(nbMine == 0){
        for(int i = 0; i < 9; i++){
            xNeighSquare = x0+COOR_NEIGHBOR[i][0];
            yNeighSquare = y0+COOR_NEIGHBOR[i][1];

            if(xNeighSquare >= 0 && yNeighSquare >= 0 && yNeighSquare < grid0.getGridWidth() && xNeighSquare < grid0.getGridHeight()){
                if(grid0.getGrid()[xNeighSquare][yNeighSquare] != DISCOVERED_SQUARE){
                    squareReveal(viewGrid0, DISCOVERED_SQUARE, grid0, xNeighSquare, yNeighSquare);
                }
            }
        }
    }
```

II. Décompostion :

Maintenant, nous allons décomposer ce morceau pour mieux comprendre.

a) 1ère boucle :

```
for(int i = 0; i < 9; i++){
    xNeighSquare = x0+COOR_NEIGHBOR[i][0];
    yNeighSquare = y0+COOR_NEIGHBOR[i][1];

    if(xNeighSquare >= 0 && yNeighSquare >= 0 && xNeighSquare < grid0.getGridWidth() && yNeighSquare < grid0.getGridHeight()){
        if(isThereAMine(grid0, xNeighSquare, yNeighSquare) && grid0.getGrid()[xNeighSquare][yNeighSquare] != DISCOVERED_SQUARE){
            nbMine++;
        }
    }
}
```

Dans cette première boucle, l'algorithme vérifie tout d’abord que le voisinage de la case ne contient AUCUNE mine autour d’elle.

b) L’Entre boucle :

```
grid0.setGrid(x0, y0, DISCOVERED_SQUARE);
viewGrid0[x0][y0] = nbMine;
```

Sur cette « entre boucle », nous avons fait en sorte que si le voisinage de la case contient une mine, alors le nombre de mine est associé à la case, et ainsi la fonction se termine.

c) 2nd boucle :

```
if(nbMine == 0){
    for(int i = 0; i < 9; i++){
        xNeighSquare = x0+COOR_NEIGHBOR[i][0];
        yNeighSquare = y0+COOR_NEIGHBOR[i][1];

        if(xNeighSquare >= 0 && yNeighSquare >= 0 && yNeighSquare < grid0.getGridWidth() && xNeighSquare < grid0.getGridHeight()){
            if(grid0.getGrid()[xNeighSquare][yNeighSquare] != DISCOVERED_SQUARE){
                squareReveal(viewGrid0, DISCOVERED_SQUARE, grid0, xNeighSquare, yNeighSquare);
            }
        }
    }
}
```

Sur cette dernière boucle, nous avons fait de façons à ce que si le voisinage de la case ne contient AUCUNE mine, alors la deuxième boucle va appeler de manière récursive, la fonction pour les cases environnantes, et ainsi de suite, jusqu'à ce que toutes les cases, soient découvertes ou contiennent une mine dans leur voisinage ou une des cases déjà découvertes.

<u>Conclusion personnelle :</u>

Aubin TOURAIS :

Pour cette conclusion finale, j'ai trouvé que ce projet était intéressant, avec une liberté de choix, ce qui nous a permis de prendre plaisir à faire ainsi que se partager les tâches.
Une bonne coalision avec mon binome, nous a permis de bien avancé et de s'entendre sur nos objectifs.

C'était une bonne expérience, et amusante.

Romain BESSON :

Pour cette conclusion finale,
J'ai pris du plaisir à faire ce projet, une expérience disons le « fun », on a pu se faire plaisir tout en res ortant un résultat et ceux dans les temps.
Nous avons pu partager nos connaissance, apprendre ensemble et construire un jeu en respectant les règles.
Très amusant.