Rapport SAE 2.01_2021

Projet réalisé par :

Baptiste Nevejans
Félix Brinet

Sommaire

Introduction

Introduction générale du sujet :

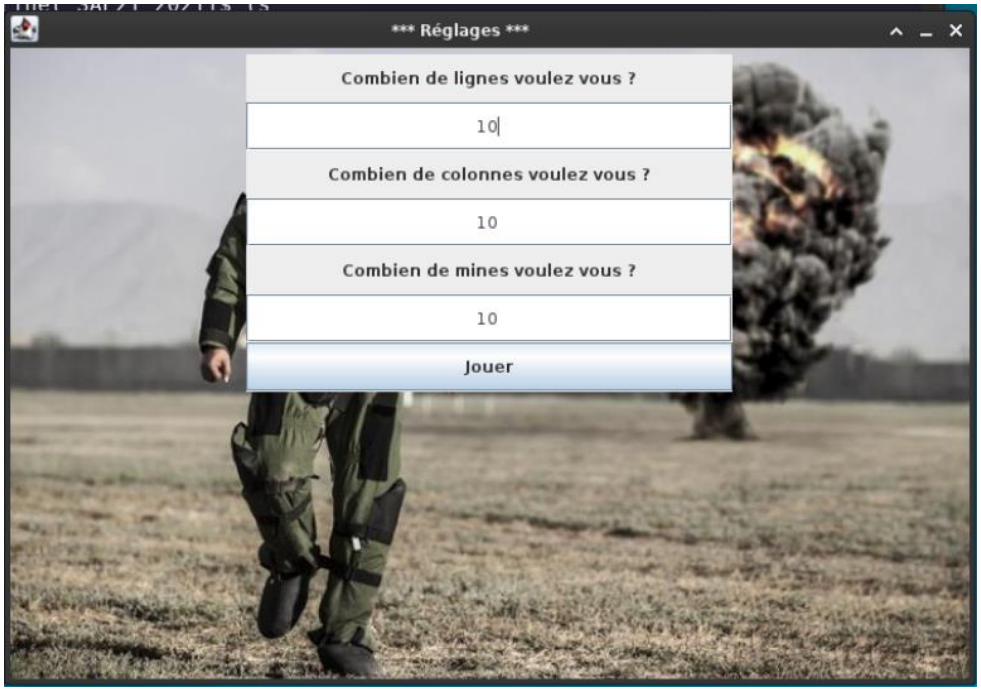
Ce projet consiste à réaliser un jeu se nommant le « démineur » à l'aide du langage de programmation Java. Pour gagner à ce jeu il faut révéler toutes les cases sauf les cases avec une mines. Si le joueur révèle une case et une mine se cachait en dessous alors la partie est terminé. Pour aider le joueur à éviter les mines, une case révélée qui n'est pas une mine affiche un chiffre entre 1 et 8. Ce chiffre correspond au nombre de mines adjacentes à la case révélée. Pour jouer au jeu, l'utilisateur peut exclusivement utiliser sa souris. Une fois qu'il a résolu le démineur, cela lui affiche un menu de victoire. Au contraire, s'il a fait apparaître une mine, un menu de défaite apparaît. Il peut dans les deux cas recommencer une nouvelle partie ou bien fermer le programme. Il peut également arrêter une partie et la recommencer plus tard en cliquant sur « quitter et sauvegarder ».

Fonctionnalités du programme

Comment jouer à notre démineur ?

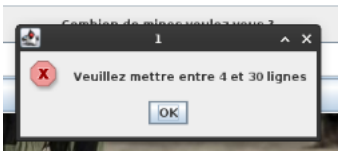
Pour jouer à notre démineur, l'utilisateur doit lancer la commande « make run ».
Un premier menu d'accueil s'affiche avec trois choix :
« Lancer une nouvelle partie », « reprendre une partie » (pour cela il faut qu'il soit déjà lancé une partie au préalable)
ou « quitter le jeu ».

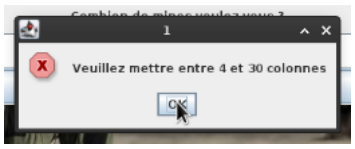
Si l'utilisateur clique sur nouvelle partie alors un autre menu de réglage s'affiche dans lequel il peut taper le nombre de lignes, colonnes et de bombes qu'il souhaite.En fonction de ses choix, la partie sera plus facile ou plus difficile.

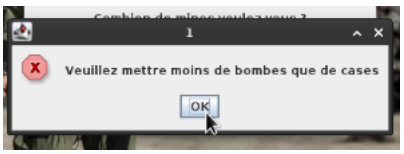


Si l'utilisateur écrit un nombre de lignes ou colonnes inférieur à quatre ou supérieur à 30, il ne peut pas lancer une nouvelle partie. De plus il ne peut pas mettre plus de mines que cases. S'il le fait alors il aura un message d'erreur
qui est affiché à la console et sur l'interface.
Ce qui fait que la partie ne se lancera pas.

En fonction de l'erreur, voici les différentes fenêtre qui peuvent s'afficher.







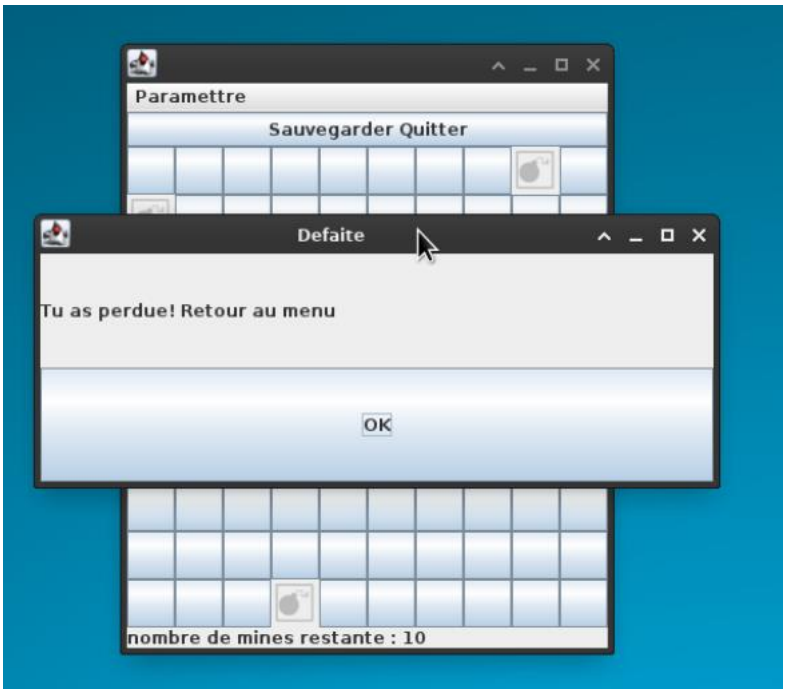
Une fois que l'utilisateur à cliquer sur le bouton jouer, la fenêtre suivante apparaît et l'utilisateur peut jouer en cliquant sur les cases. Le nombre de mines restantes est indiqué en bas à gauche. Il y a également un bouton « Sauvegarder et Quitter » qui créer un fichier et renvoie au menu afin que l'utilisateur puisse recharger ce fichier et reprendre sa partie plus tard. Dès que l'utilisateur clique sur une case qui n'est pas une mine, une image avec un nombre apparaît lui indiquant le nombre de mines adjacentes ou une case vide. si aucune mines autour.

Voilà à quoi ressemble les images de chiffres et la bombe :

Malheureusement les couleurs ne s'affichent pas sur le démineur, nous ne savons pas pourquoi.

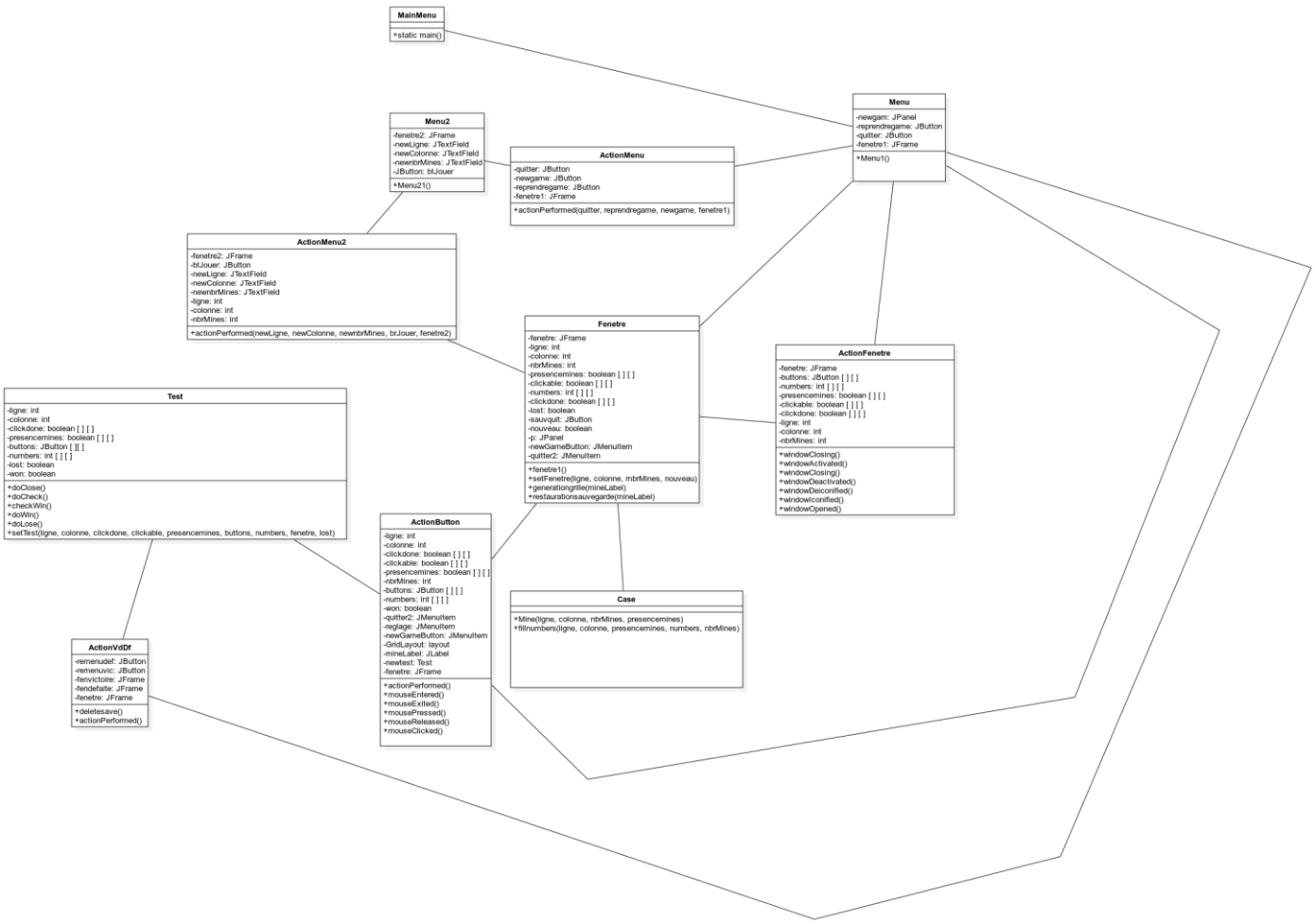
Si l'utilisateur réussi à révéler toutes les cases sans révéler de mines alors une fenêtre de victoire apparait avec un bouton « OK » qui permet de retourner au premier menu pour relancer une nouvelle partie ou quitter le programme.

Si l'utilisateur découvre une mine alors une fenêtre de défaite apparait avec un message qui annonce que l'utilisateur a perdu avec un bouton « OK » qui permet de retourner au premier menu pour relancer une nouvelle partie ou quitter le programme.



Structure du programme

Notre programme est divisé en 10 fichiers, le fichier main qui permet l'exécution du programme est le fichier « MainMenu ». Le choix de lancer, quitter ou revenir sur la partie est sur le « Menu » qui pour faire ses actions appelle « ActionMenu », il peut également appeler directement « Fenetre » sans passer par « Menu2 » pour restaurer le fichier précédemment enregistrer. Ensuite les réglages de la partie se trouve dans « Menu2 » qui pour faire ses actions appelle « ActionMenu2 » Ensuite la classe précédente lance la classe « Fenetre » qui appelle « ActionFenetre » qui vas permettre de sauvegarder quand on ferme la fenêtre. Il appelle également « ActionButton » qui permet d'appliquer toutes les options proposer par « Fenetre » et réaliser les tests quand le joueurs clique sur un bouton. Ensuite la classe « ActionVdDf » est appellé pour vérifier si le joueur perd ou gagne. La compilation des classes se fait dans l'ordre inverse des appels de classes décrite précédemment dans le fichier « MakeFile ».



Mécanisme de Sauvegarde

Pour sauvegarder on commence par créer le fichier appelé save.dat. Ensuite on vient y insérer les informations petit à petit. On commence par récupérer le nombre de ligne et le nombre de colonne. On fait ensuite boucle pour récupérer le nombre présent dans les cases. On fait une boucle qui récupère la présence de mine et qui écrit un byte de 1 s'il y en a une te un byte de 2 s'il n'y en a pas. On fait une nouvelle boucle qui vérifie si la case est cliquable ou non, si c'est cliquable alors le byte est de 1, si c'est non cliquable alors le byte écrit est de 2. On fait une nouvelle boucle qui vérifie si la case est cliquée, si la case est cliquée alors le byte écrit est de 1, si la case n'a pas été cliqué alors le byte écrit est de 2. On fait une nouvelle boucle qui récupère le caractère présent dans chaque bouton, si c'est un espace le byte écrit est 1, si c'est ? Le byte écrit est de 2 et si c'est une étoile le byte écrit est de 3 lorsque la valeur n qui est le nombre de case que le joueur est sûr d'avoir trouvé augmente de 1. On vient ensuite récupérer le nombre de mine restante que l'on a calculé préalablement en soustrayant le nombre de mine initial et le nombre d'étoile étant marqué d'une étoile. Puis on ferme le fichier et on retourne au premier menu

Ensuite pour récupérer les données on extrait les informations du fichier « save.dat ».
On récupère le nombre de colonne et de ligne. On fait une boucle qui donne pour chaque son nombre associé. On fait une boucle qui vérifie la valeur du byte si le byte est de 1 il y a une mine sur cette case, si le byte est de 2 il n'y a pas de mine sur cette case. On fait une boucle qui vérifie la valeur du byte si le byte est de 1 la case devient cliquable, si le byte est de 2 la case devient non cliquable . On fait une boucle qui vérifie la valeur du byte si le byte est de 1 il y a une mine sur cette case, si le byte est de 2 il n'y a pas de mine sur cette case. Ensuite on ajoute les boutons avec le listener en plus. On fait une boucle qui met clickdone false par défaut et vérifie la valeur du byte si le byte est de 1 alors on clique sur la case, si le byte est de 2 alors on passe à la case suivante. On récupère le nombre de mine puis on écrit le nombre de mine restant.

Algorithme révélation cases

Pour révéler les cases je commence mon action en proposant à l'utilisateur qu'à chaque clique gauche il vérifie si c'est un bouton, de quel bouton il s'agit et s'il est cliquable pour ensuite lancer les tests.

On va d'abord commencer par écrire chaque condition de test, c'est à dire regarder s'il y a une mine en haut, en bas, à droite, à gauche, en haut à droite, en haut à gauche, en bas à droite et en en bas à gauche.

Lorsque l'utilisateur clique sur une case, la méthode test va regarder si cette case possède un nombre qui est enregistré comme étant 0, que ce n'est pas une mine et que le joueur n'a pas gagné. Si c'est le cas il va ensuite vérifier les cases adjacentes et il va révéler les cases si elles n'ont pas été cliqué en utilisant les variables left qui correspond à la position de la case décalé à gauche, right qui correspond à la position de la case décalé à droite, up qui correspond à la position de la case décalé en haut , down qui correspond à la position de la case décalé en bas. Et les cliques en utilisant la méthode « doClick » sur ces cases et celle-ci passe en cliquer.

Lorsque qu'une case possède un chiffre différent de 0 . Elle est révélée avec le chiffre correspondant aux nombres de mines adjacentes.
Lorsque c'est une mine alors la méthode « doLose » est lancé et le joueur perd.

Et ça révèle une bombe. Et la fonction doLose() va également appliquer cette algorithme.

Conclusions personnelles

Conclusion Félix Brinet :

Pour ma part ce projet m'a permis d'apprendre comment découper un programme en différentes classe. J'ai compris que créer une classe pour chaque partie du programme permet de mieux diviser le travail afin de travailler en groupe. De plus il est plus facile de détecter les problèmes dans une des classes que dans tous un programme.

Je rencontrais quelques difficulté en Java mais grâce à l'aide Baptiste et de mes camarades, je peux affirmer que cela m'a aidé à mieux comprendre comment organiser son code. En réalisant un diagramme de classe par exemple. J'ai aussi appris à mieux utiliser les ''actionlistener'' .

Pour finir, j'ai trouvé ce projet assez amusant à faire car la création d'un jeu que l'on doit tester et donc y jouer pour vérifier qu'il fonctionne bien. C'est une manière de programmer qui est ludique.

Conclusion Baptiste Nevejans :

Pour ma part ce projet m'a permis de mettre en lien de nombreuse classe et de comprendre l'importance de divisé son programme correctement. Il m'a aussi permis d'apprendre l'utilté de faire des tests pour chaque fonction afin de visualiser la cause exact du problème.

Ce projet m'a permis de renforcé mes compétences en apprenant à utilisé des ActionListenners, d'apprendre à être plus efficace. Ce projet m'a aidé à mieux structuré un programme et à optimisé des applications. On a utilisé un diagramme de classe pour savoir ce qu'on allait faire au debut du projet. Et l'evolution du projet c'est vue sur le diagramme.

Pour finir ,j'ai trouvé ce projet très intéressant, faire le code d'un jeu que tu connais bien est très amusant, tu peux t'amuser à y jouer pour vérifier ton code.