UMONS
Université de Mons

Faculté
des Sciences

Rapport – Projet Informatique

OZUDOGRU Huseyin
Université de Mons
Année Académique 2019-2020

Sources des images : Logo de l'Université ainsi que celle de la Faculté - https://web.umons.ac.be/fr/journaliste/photos-logos-tournages/ // Image du jeu - Capture d'écran prise lors d'une partie.

Table des matières

Introduction

Ce projet a été proposé par monsieur Mélot et son équipe dans le but de réaliser une application graphique en Java permettant de jouer au jeu « Démineur » tout en appliquant les concepts de la POO[2] vu aux cours de « Programmation et Algorithmique I » et « Programmation et Algorithmique II ».

Description des algorithmes utilisés

Tout au début du projet, j'ai créé les tableaux nécessaires pour le plateau de jeu.
Il était donc évident d'implémenter tout d'abord un algorithme qui permet de placer des mines.

Algorithme de placement des mines

Pour cet algorithme, j'ai décidé d'aller voir la page Wikipédia[3] du jeu.
Le site propose un algorithme :

« *Pour pouvoir placer des mines aléatoirement, il faut avoir accès à un générateur de hasard. Puis on lui demande un nombre compris entre 1 et n, ce sera le numéro de la ligne, et un second nombre compris entre 1 et m, ce sera le numéro de colonne. Ensuite on regarde si la case ainsi choisie comporte déjà une mine, auquel cas on ne fait rien et sinon on y place une mine. On effectue ce schéma jusqu'à ce qu'on ait placé x mines*. »[3]

Ma méthode « placesMines(n) » prend un entier n en entrée, où n est le nombre de mines à placer et ne retourne rien. Nous entrons d'abord dans une boucle qui permet de placer un par un les mines. Dans cette boucle, nous créons un objet « Random » que l'on va utiliser dans les variables « randLine » et « randColumn ». Ces variables servent à générer aléatoirement une position en abscisses et en ordonnées. Ensuite, nous créons une condition pour vérifier si dans la position générée, il y a déjà une mine : si c'est le cas, on revient en arrière dans la boucle pour regénérer une position aléatoire, sinon on place la mine.

Nous aurons donc une complexité algorithmique de O(n).

[2] Programmation orientée objet.
[3] Source consultée le 31/07/2020 à 20h37 :
https://fr.wikipedia.org/wiki/D%C3%A9mineur_(genre_de_jeu_vid%C3%A9o)

Algorithme de calcul du nombre de mines de chaque case

Je suis également allé voir la page Wikipédia[3] mais je n'ai pas fait exactement pareil. Le site propose deux grands algorithmes, je me suis inspiré celui du par « case » :

« *Le premier algorithme est dit naïf : c'est la méthode la plus évidente à comprendre et à mettre en place, mais ce n'est pas la plus optimisée.*

Après le placement des mines, l'algorithme parcourt le tableau en entier, case par case. Pour chacune d'entre elles, il compte le nombre de mines dans le voisinage direct, et lui assigne ce nombre. »[3]

Ma méthode « neighboursCalculator() » ne prend rien en entrée et ne retourne rien. Elle va tout simplement parcourir deux fois le tableau de mines un par un : la première fois pour les mines et la deuxième fois pour les cases voisines. Si les coordonnées des cases parcourues sont voisines (vérifiée par la méthode « isNeighbour() »), alors on incrémente un aux cases avec la variable « neighbors » et on assignera en position x et y la variable dans le tableau « neighbours ».

Parlons également de la méthode « isNeighbour() » qui prend en paramètres quatre entiers qui représentent deux couples de coordonnées et vérifie tout simplement si les cases en ces coordonnées sont voisines. Elle retourne « true » si oui, « false » sinon.

Algorithme de reset

Comme son nom l'indique, cet algorithme permet de réinitialiser une partie en cours.

On prend un entier n en entrée qui est le nombre de mines qui sera utilisé dans l'invocation de la méthode « placeMines(n) » plus tard et ne retourne rien.
On met en « true » la variable « reset », en « false » les variables « victory » et « defeat » et remise du « timeCounter » à zéro. Ensuite, on retire les mines et referme les cases avec une boucle qui parcoure le tableau de mines. Pour finir, on invoque la méthode « placeMines(n) » pour replacer les mines et la méthode « neighboursCalculator() » pour recalculer le nombre de mines autour d'une case.
On remet la variable « reset » à « false » pour dire qu'on a fini de réinitialiser.

Algorithme de vérification de victoire ou défaite

Il faut savoir quand est-ce que le jeu se termine.
On peut dire que c'est une défaite lorsque l'on clique sur une mine et on peut dire que c'est une victoire lorsque toutes les cases fermées restantes sont des mines.

Lorsque l'on invoque la méthode « checkVictory() » qui ne prend rien en entrée et ne retourne rien, on entre dans une condition où la variable « defeat » est « false ». Dans cette condition, on parcoure le tableau de mines avec une boucle : si une case en

coordonnées (x,y) est ouverte et que c'est une mine, la variable « defeat » devient « true » et le jeu se termine. Sinon, on vérifie que le nombre de cases ouverts par le joueur est plus grand ou égal au nombre de cases qui ne contiennent pas de mines et que la variable « victory » est « false ». Si c'est le cas, « victory » devient « true » et la partie s'achève sur une victoire.

Algorithme de calcul du nombre total de mines et de cases ouvertes

Les deux méthodes sont presque identiques. Elles sont utilisées dans la méthode « checkVictory() ».
On initialise un entier à zéro. Ensuite, on parcourt le tableau de « mines » (ou « opened ») avec une boucle et si le tableau contient une mine (ou si la case est ouverte), on incrémente l'entier initialisé au début.
Cette méthode retourne l'entier initialisé au début.

Algorithme flood fill

Dans le vrai jeu, lorsque l'on clique sur une case qui ne contient pas de mines autour de lui (une case qui contient « 0 » finalement), le jeu va ouvrir « par diffusion » les autres cases autour qui contiennent également des « 0 » jusqu'à ce que l'on croise des cases qui contiennent autre nombre que « 0 ». Après avoir recherché sur internet, j'ai trouvé que c'est le principe de l'algorithme appelé « flood fill ». Voici sa définition selon Wikipédia :

« *L'algorithme de remplissage par diffusion est un algorithme classique en infographie qui change la couleur d'un ensemble connexe de pixels de même couleur délimités par des contours. Il est fréquemment utilisé par les programmes de manipulation d'images matricielles comme Paint. Il trouve également son application dans certains jeux tels que le démineur, Puyo Puyo et Lumines afin de déterminer quels éléments du plateau de jeu sont à révéler.* »[4]

Après quelques recherches sur le web afin de trouver un pseudo code, je suis tombé sur un topic du site Stackoverflow où une personne partage une façon de l'implémenter en Java[5]. Je l'ai alors implémenté dans mon code source, tout en l'adaptant par rapport à mon propre codage.

Cependant, un problème persiste pendant le jeu. Je vais l'évoquer plus tard dans la partie « Erreurs connues du programme ».

[4] Source consultée le 9/08/2020 à 17h39 :
https://fr.wikipedia.org/wiki/Algorithme_de_remplissage_par_diffusion
[5] Source consulté le 11/08/2020 à 21h48 : https://stackoverflow.com/questions/14076090/floodfill-minesweeper-explanation-needed

Points forts du projet

On peut citer quelques points fort du projet.
Pour commencer, le code source est divisé en plusieurs classes, ce qui permet de lire et comprendre le code beaucoup plus facilement. De plus, les méthodes et les classes sont commentées grâce à la JavaDoc.

Ensuite, la plupart des algorithmes utilisés sont en général de complexité linéaire.
Si l'on prend par exemple l'algorithme du placement des mines, elle est en O(n) car nous avons une boucle qui répète l'opération n fois. Dans cette boucle ainsi qu'à l'extérieur, nous avons des opérations élémentaires, ce qui fait que O(n) emporte sur O(1).

Pour finir, pratiquement tous les autres algorithmes ont une complexité de O(m x n) car on parcourt des tableaux de deux dimensions, ce qui implique que nous avons une boucle dans une boucle. Et vu que dans et en dehors de ces boucles, nous avons que des opérations élémentaires, on a que O(m x n) emporte sur O(1).

Points faibles du projet

Je pense que je n'ai pas assez utilisé le concept « d'encapsulation ». Je voulais l'utiliser beaucoup plus que ça mais l'IDE que j'utilise (Eclipse) sortait beaucoup d'erreurs un peu partout, ce qui ne m'a pas laissé le choix de mettre des « private static » sur certains de mes variables ou encore mes tableaux en « public static ».

Le fait d'utiliser le toolkit graphique Java Swing est un peu dépassé. En effet, utiliser JavaFX aurait été beaucoup plus performant et meilleur afin d'adapter au mieux l'interface graphique sur différents types d'écrans ou systèmes d'exploitation.

Ensuite, l'absence des tests unitaires pour tester les victoires et les défaites pour voir si la partie s'arrête dans les cas prévus par les règles.

Pour terminer, utiliser la classe « Date() » est aussi un mauvais choix de ma part car c'est ce que j'ai utilisé pour le chronomètre des parties mais il est très difficile et pas pratique de le manipuler.

Erreurs connues du programme

Durant mes tests et les parties que j'ai joué j'ai remarqué quelques problèmes.
J'aimerais tout d'abord aborder le problème avec l'algorithme flood fill. Effectivement, l'algorithme n'est pas fonctionnel à cent pourcents. Lorsque l'on clique sur une case contenant un « 0 », celui-ci n'ouvre pas toutes (voir pas dans certains cas) les cases

qu'il est censé ouvrir. J'explique ce problème par le déclenchement de 2 exceptions : ArrayOutOfBoundsException et StackOverFlowException que j'ai bien évidemment capturé. J'ai essayé de résoudre le problème en essayant de mettre des conditions mais le problème persiste toujours.

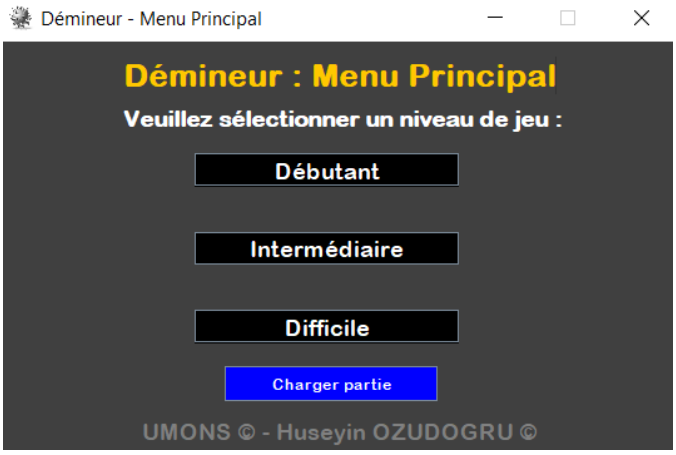
Le système de sauvegarde est présent mais incomplète et non fonctionnel (manque de temps…). Malgré beaucoup de tutoriels et de documentations consultés sur internet, j'ai été incapable d'implémenter un système correct et fonctionnel. J'étais parti sur le concept de (dé)sérialiser car je voulais sauvegarder mon objet « Gameboard » mais je n'y arrivais pas à sauvegarder correctement l'objet, encore moins de le charger. J'ai quand même laissé la classe « Save.java » dans les fichiers sources.

La veille de la remise du projet, j'ai testé mon jeu sur une machine virtuelle Ubuntu. La taille de la fenêtre, ainsi que certaines écritures sont affichées un peu plus grand que prévu. Je ne comprends pas pourquoi ça fait cet effet sous Ubuntu or que sur Windows, tout fonctionnait très bien.

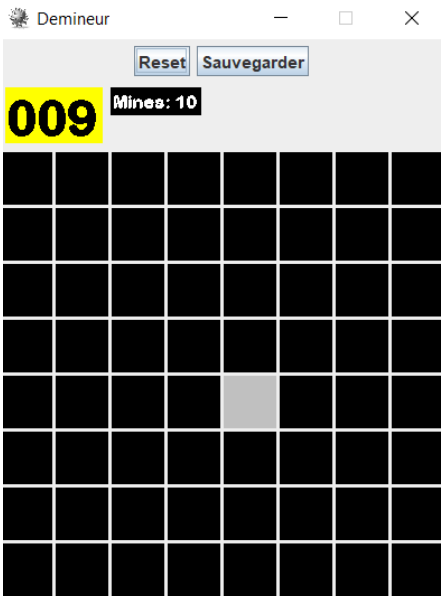
Les apports positifs de ce projet

Durant son développement, le projet m'a aidé énormément à apprendre non seulement sur le langage Java, mais aussi sur la programmation en général.
Moi qui ne comprenais pas comment mettre en œuvre ce que l'on apprenait et ce que l'on codait depuis le début de l'année, le projet m'a permis de comprendre comment on faisait une interface graphique, comment mettre en relation les périphériques avec certains évènements qui se passe dans le programme, faire fonctionner les méthodes dans l'interface graphique. Cela m'a permis également de me rendre compte comment la technologie qui nous entoure est très complexe et que pour atteindre le niveau pour produire des projets de tels envergures, il faut que je travaille et que je m'entraîne beaucoup plus de mon côté.

Mini Guide d'utilisation du jeu



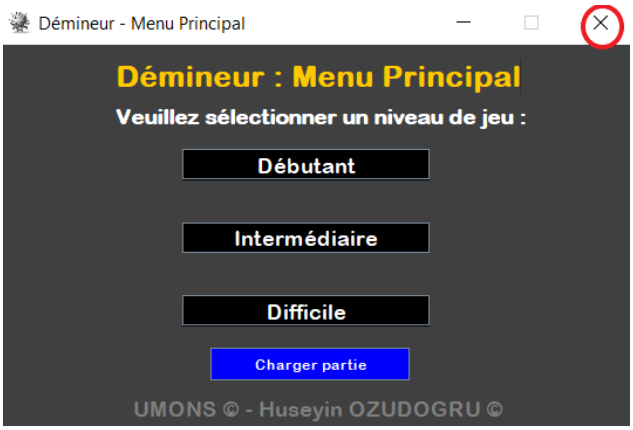
Une fois le jeu lancé, vous vous retrouvez directement dans le menu principal, il suffit de cliquer sur une des 3 difficultés de jeu afin de lancer une partie ou de cliquer sur « Charger Partie » si vous avez sauvegardés une partie au préalable.



Une fois la partie lancée, une nouvelle fenêtre s'ouvre et la partie est démarrée.

Si vous souhaitez recommencer la partie ou de sauvegarder, vous pouvez cliquer sur les boutons au-dessus.

Fermer la fenêtre permet de quitter la partie en cours et de retourner dans le menu principal (qui est toujours exécuter en arrière-plan).



Pour fermer le jeu, il suffit de fermer le menu principal.

Remerciements

Je remercie monsieur Dubrulle pour avoir réservé une heure de réunion avec moi sur Teams afin de me donner son avis ainsi que quelques conseils pour l'amélioration du projet durant son développement.