Programmation orientée objets : Démineur

Gaillor Jowardo JINORO
Timothée FONTENILLE

16 Avril 2023

1 Introduction

Dans le cadre de ce mini-projet en programmation orientée objet, notre objectif était de concevoir et implémenter une version du jeu démineur en Java. Le démineur est un jeu de réflexion classique, développé à l'origine pour les systèmes d'exploitation Windows.

Ce rapport présente les étapes clés de notre projet : Le fonctionnement du jeu, la structure, le déroulement d'une partie et la sauvegarde.

2 Fonctionnement du jeu

Le but du jeu est de dévoiler toutes les cases qui ne contiennent pas de mines, en évitant de déclencher une mine. Le joueur doit utiliser les indices fournis par le nombre de mines adjacentes à une case pour déduire l'emplacement des mines et progresser dans le jeu.

Au lancement, pour la première partie, l'utilisateur doit rentrer un niveau de difficulté ainsi que la taille du plateau. Si il a sauvegardé une partie, il aura l'occasion de reprendre à la sauvegarde.

Pendant une partie, l'utilisateur a 4 choix : 1 - quitter sans sauvegarder, 2 - marquer une case, 3 - découvrir une case, 4 - quitter et sauvegarder la partie. Lorsque l'utilisateur veut marquer ou découvrir une case, il doit rentrer le numéro de la ligne ainsi que le numéro de la colonne pour sélectionner la case voulue.

Une partie s'arrête si l'utilisateur découvre une mine ou si il découvre toutes les cases vides du plateau. Lorsque l'utilisateur doit rentrer son choix dans le terminal, il doit l'orthographier comme indiqué, sinon le programme va générer une erreur.

ATTENTION : Le programme à été conçu pour fonctionner sur la VM de l'université (Linux).
Pour lancer le jeu, il faut lancer la classe Main du fichier build avec la commande suivante dans le terminal : java Main

3 Structure et affichage

Notre programme est découpé en 4 classes : Cellule pour toutes les cases du jeu, Grid pour tout le plateau du jeu, GamePlay pour les choix de l'utilisateur, Main pour lancer le jeu.
Pour avoir la grille par une matrice doublement chaînée en 8-connexité, on utilise un EnumMap POSITION_VOISINES_MAP dans la classe Cellule contenant les 8 cellules voisines de chaque case. Les Cellules sont stockées dans une matrice de cellules dans la classe Grid, ce qui nous permet de sélectionner une case avec un indice ligne et colonne.

Le plateau est représenté dans le terminal Linux, une case non découverte est représentée par le motif "==", une case vide n'a pas de motif, une case marquée est représentée par un drapeau, une bombe est représenté par le motif "*".
Lorsque qu'une cellule est vide, mais qu'au moins de ses voisines a une bombe, on affiche le nombre de bombe entre les 8 cases voisines.

4 Vérification et programme principal

La méthode uncover() de la classe Grid va gérer la découverte d'une case en prenant comme paramètre le choix de l'utilisateur ainsi que les indices ligne et colonne d'une cellule. Cette fonction retournera un booléen, indiquant si le joueur a perdu la partie (ou non) en découvrant une bombe.
Lorsque une cellule est vide, on découvre toutes les cellules voisines de celle-ci, si une des cellules voisine est vide à la découverte, on continue de découvrir toutes les cellules voisines et ainsi de suite. Ceci est traité par l'appel de la fonction récursive CelluleVideAdj() dans la classe Grid.

La classe GamePlay() traite toutes les entrées de l'utilisateur. Au début de la partie on demande à l'utilisateur de rentrer le niveau de difficulté de sa partie, le nombre de bombe est généré par rapport à la taille du plateau et le niveau de difficulté : Facile : 10% des cases du plateau contiennent une bombe, Moyen : 30%, Difficile : 45% et Légende : 65%. Les bombes sont réparties aléatoirement.

Pendant une partie, on affiche l'état de la grille après chaque action de l'utilisateur. Les actions de l'utilisateur (marquage/découverte) sont elles traitées avec l'appel de la fonction uncover().

5 Sauvegarde et restauration

La méthode save() est appelée lorsque l'utilisateur rentre le choix 4 - quitter et sauvegarder. Elle prend en paramètre un path qui représente le chemin vers un fichier texte (de même pour load()). La méthode save() écrit les informations nécessaires à la sauvegarde de la partie, on a en première ligne du fichier texte toutes les informations (séparée d'un underscore "_") nécessaires à la construction de la grille. Toutes les autres lignes du fichier représentent une cellule et les informations nécessaires pour construire les cellules et reconstituer le tableau de cellules. On écrit toutes les informations (séparée d'un underscore "_") d'une cellule avec la fonction getAll() de la classe Cellule.

Pour restaurer une partie, on utilisera la méthode load() dès le lancement de la partie (sur demande de l'utilisateur). La méthode load() sépare toutes les informations dans une liste de String appelée "element" dans la classe GamePlay. On utilisera les éléments de la première ligne pour construire la grille, ainsi que les autres éléments pour construire les cellules et reproduire le tableau de cellule.

6 Conclusion

Ce mini-projet en programmation orientée objet nous a permis de concevoir et développer le jeu démineur en Java. Grâce à une structure bien organisée et une répartition judicieuse des responsabilités entre les différentes classes, nous avons pu créer un jeu fonctionnel et facile à utiliser.

Nous avons mis en pratique divers concepts clés de la programmation orientée objet, tels que l'encapsulation, l'héritage, et le polymorphisme.