

PROJET ALGORITHMIQUE DEMINEUR

Table des matières

Introduction.....	3
Règles du démineur.....	3
Graphisme.....	4
Fonctions.....	5
Menu principal.....	6
Déroulement d'une partie.....	7
Rejouabilité.....	11
Classement des joueurs.....	11
Gestion des fichiers.....	11
Améliorations.....	12
Conclusion.....	12

Introduction

Nous avons choisi dans le cadre de ce projet de programmer le jeu du démineur. Il nous a paru intéressant de choisir ce sujet.

En effet, bien que simple en apparence le fonctionnement de ce jeu est difficile à retranscrire mathématiquement (cf. Récurrence). Le programme est écrit en langage JAVA, il est composé de plusieurs fonctions commentées et d'un main. Le jeu dispose de plusieurs fonctionnalités, tel qu'un classement des joueurs, d'un historique des parties et du choix de plusieurs difficultés. Vous pouvez trouver ci-joint le .java nommé « projet.java »

Règles du démineur

Le champ de mines est représenté par une grille constituée de cases vides ou de mines. Le joueur actionne un bouton pour révéler ce que contient la case. Si la case est vide le nombre de mines autour s'affiche sur la case. Si la case contient une mine, un symbole représentant une mine s'affiche sur la case et le joueur perd la partie. Pour gagner une partie le joueur doit révéler toutes les cases vides sans cliquer sur une case contenant une mine. Le joueur dispose également du drapeau pour l'aider à retenir où se trouve les mines. Quand il estime avoir deviné la position d'une mine, il peut actionner une autre touche pour afficher un drapeau sur la case. Cette aide n'est que graphique, si le joueur clique sur un drapeau, contenant préalablement une mine, il perd.

Graphisme

Pour l'affichage de notre programme nous avons utilisé la class EcranGraphique avec les fonctions suivantes

`static void flush()`

affichage de l'image.

`Static void clear()`

Effacement de l'image.

`static void setColor(int r, int v, int b)`

Définition de la couleur de trace (blanc par défaut).

`static void fillRect(int xi, int yi, int tx, int ty)`

Remplissage d'un rectangle

`static void drawString(int x, int y, int police, java.lang.String s)`

Traçage d'une chaîne de caractères composée uniquement de caractères non accentués.

`static int [][] loadPNGFile(java.lang.String filename)`

Chargement d'une image au format PNG.

`static void drawImage(int px, int py, int[][] img)`

Affichage d'un rectangle de pixels dans la fenêtre d'affichage.

`static void exit()`

Interruption du fonctionnement de l'application.

`static void wait(int ms)`

Temporisation

`static int getMouseX()`

Retour de la position en x de la souris au sein de la fenêtre (-1 si en dehors).

`static int getMouseY()`

Retour de la position en y de la souris au sein de la fenêtre (-1 si en dehors).

`static int getMouseState()`

Retour de l'état actuel de la souris (0 pour bouton non presse, 1 pour bouton presse, 2 pour clic réalisé).

`static int getMouseButton()`

Retour du dernier bouton utilise de la souris (-1 pour pas de bouton utilise, 1 pour bouton gauche, 2 pour bouton central, 3 pour bouton droit).

`static char getKey()`

Retour du dernier caractère tape au clavier (caractère de code ASCII 0 si pas de caractère saisi au clavier).

`static int getSpecialKey()`

Retour du code de la dernière touche « spéciale » (touche de curseur, touches de fonction, ...) tapée au clavier (0 si pas de touche frappée).

`static void init(int px, int py, int wx, int wy, int tx, int ty, java.lang.String titre)`

Initialisation et ouverture d'une fenêtre d'affichage graphique.

Fonctions

Fonctions utilisées durant la création du programme :

```
private static void affiche_getKey()  
    Affiche la valeur de EcranGraphique.getKey().  
private static void affiche_getSpecialKey()  
    Affiche la valeur de EcranGraphique.getSpecialKey().  
private static void affiche_position_souris()  
    Affiche la position de la souris.  
private static void affiche_tableau (int [][] tableau)  
    Permet d'afficher un tableau de int dans la console.  
private static void affiche_tableau_char (char [][] tableau)  
    Permet d'afficher un tableau de char dans la console.  
private static void affiche_tableau_string (String [][] tableau)  
    Permet d'afficher un tableau de String dans la console.  
private static void afficher_liste(double[] liste)  
    Affiche une liste de double dans la console;  
private static void afficher_liste_string(String[] liste)  
    Affiche une liste de String dans la console.  
private static void afficherContenu(String fichierPhysic) throws IOException  
    Affiche le direct.dat contenant l'historique des joueurs ayant terminé une partie.
```

Fonctions utilisées pour le jeu :

```
private static void afficher_image (String nom_img, int x, int y)  
    Permet d'afficher l'image nom_img aux coordonnées (en pixels) x et y.  
private static void menu (int [][] tableau_bombe, int[][] lvl, int[][][] tableau_position_casePixel, String[]  
classement, char[][] tableau_voisin, int[][][] tableau_position_casePixel_String, int difficulte,  
double[]temps_classement) throws IOException  
    Fonction principale qui affiche le menu.  
private static void ajouter_liste(double[] classement, double temps)  
    Ajoute a une liste de double un élément différent de 0.  
private static void ajouter_liste_string(String[] classement, String joueur)  
    Ajoute a une liste de String un élément.  
private static void ajouterLignes(String fichierPhysic, String append) throws IOException  
    Ajoute le contenu append à un fichier fichierPhysic.  
private static void drawString(int x, int y, int[][][] tableau_position_casePixel_String, int r, int g, int b, String  
string)  
    Affiche un string au (x,y) donnés.  
private static double duree(double tempsDepart, double tempsFin)  
    Calcul le temps entre deux System.currentTimeMillis() en seconde.  
private static boolean existe(int x, int y, char[][] tableau_voisin)  
    vérifie si une case du tableau existe (renvoie un booléen true si tel est le cas).  
private static int getKey()  
    acquisition d'un int correspondant a une touche du clavier.  
private static int getMouse()  
    acquisition d'un int correspondant a une touche de la souris.  
private static int getSpecialKey()  
    acquisition d'un int correspondant spéciale du clavier.
```

```
private static void jeu(int[] position_souris, char[][] tableau_voisin, int[][][] tableau_position_casePixel,
boolean etat, int[][][] tableau_position_casePixel_String, int[][] tableau_bombe, int[][] lvl, String[]
classement, int difficulte, double[] temps_classement)
```

Fonction jeu qui permet de choisir en ouvrir une case ou poser un drapeau.

```
private static void leaderboard (String[] classement, double[] temps_classement)
```

Affiche le sous menu leaderboard avec plusieurs cas, si aucune partie n'est terminée ou si au moins une partie est terminée.

```
private static int nbBombesVoisines(int x, int y, char[][] tableau_voisin)
```

vérifie si la case contient un 'b' (bombe/mine) pour que la variable « nb » augmente de 1. Pour une case donnée on vérifie toutes les cases autour, dans toutes les directions (8 possibilités).

```
private static void ouvre(int x, int y, char[][] tableau_voisin, int[][][] tableau_position_casePixel,int[][][]
tableau_position_casePixel_String,int[][] tableau_bombe, int[][] lvl, String[] classement, int difficulte,
double[] temps_classement)
```

La fonction « ouvre » va ouvrir toutes les cases vides autour de la case sélectionnée (si la case était vide).

```
private static void paint(int x, int y, int[][][] tableau_position_casePixel, int r, int g, int b)
```

affiche un rectangle au (x,y) donnés

```
private static int[] position_souris()
```

acquisition des coordonnées x_pixels / y_pixels de la souris puis conversion en coordonnées ligne / colonne.

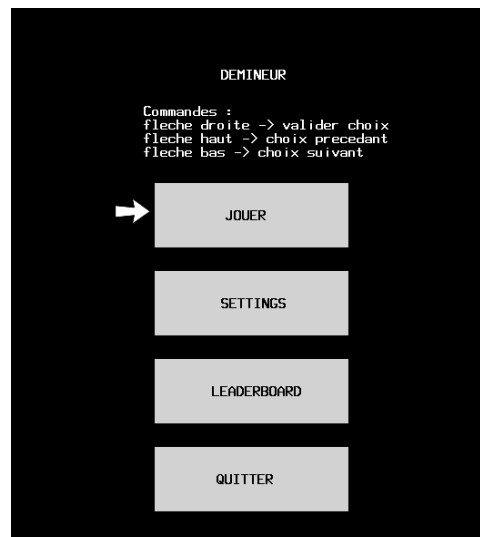
```
private static int[][] randomisation_normal/difficile/facile(int[][] tableau_bombe, char[][] tableau_voisin)
```

Renvoie les tableaux de mines et de voisins avec une distribution des bombes aléatoires

Menu principal

Le menu se lance par la fonction « menu ». Il y a un affichage des commandes par les fonctions « EcranGraphique.drawString » puis un affichage des cases correspondants aux sous menus. L'utilisateur comprends que la flèche à coté des cases permet de sélectionner un sous menu. La position de la flèche est enregistrée par la variable « positionFleche ». Lorsque le joueur appuie sur une flèche directionnelle il enregistre sa saisie avec la variable « specialInput » qui le fait rentrer dans un switch. En fonction de la valeur de « specialInput » plusieurs cas se réalisent :

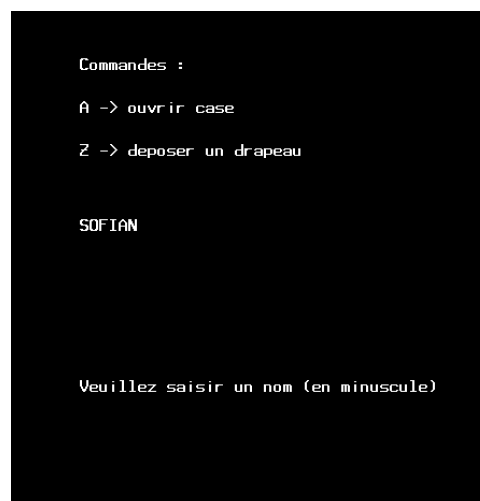
- L'utilisateur actionne la flèche du haut :
Le menu s'efface avec la fonction « EcranGraphique.clear » puis s'affiche avec la flèche sur une case en haut ou en bas si la case précédente était le plus en haut.
- L'utilisateur actionne la flèche du bas :
Le menu s'efface puis s'affiche avec la flèche sur une case en bas ou en haut si la case précédente était au plus bas.
- L'utilisateur actionne la flèche de droite :
En fonction de la valeur de « positionFleche » plusieurs cas se réalisent : JOUER, SETTINGS, LEADERBOARD, QUITTER



Menu

Déroulement d'une partie

Lorsque l'utilisateur sélectionne la case « JOUER » un menu de saisie du nom du joueur s'affiche. La fonction « saisie » permet d'enregistrer le nom du joueur mais également d'afficher simultanément la saisie du joueur sur l'écran de jeu. Le joueur peut utiliser l'ensemble des lettres du clavier, effacer la dernière lettre saisie avec la touche return. En revanche la taille de la saisie est limitée à 24 caractères (choix arbitraire).



Ecran de saisie

Dès lors que le joueur appuie sur entrée le jeu se lance et la grille est affichée. La grille du démineur est modélisée dans le programme par des matrices :

[illegible][illegible]

8 sur 12

[illegible]

Matrice tableau position casePixel / Matrice tableau position casePixel String :

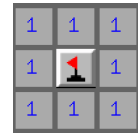
[illegible][illegible]

La première grille est affichée puis la fonction « jeu » s'exécute, l'utilisateur peut alors actionner deux touches :

- Lorsque l'utilisateur actionne la touche « a » la position du curseur va être enregistrée, la fonction « ouvre » va prendre en entrée les coordonnées de la case. Grâce à des appels récursifs de la fonction ouvre les fonctions « existe » et « nbBombesVoisines » (inspirées des fonctions du poly de cours) vont être exécutées. « existe » vérifie si une case du tableau existe, elle renvoie True si tel est le cas. Si la fonction existe renvoie un True alors la fonction « nbBombesVoisines » vérifie si la case contient un 'b' (bombe/mine) pour que la variable « nb » augmente de 1. Pour une case donnée on vérifie toutes les

cases autour, dans toutes les directions (8 possibilités). La fonction « ouvre » va ouvrir toutes les cases vides autour de la case sélectionnée (si la case était vide). Lorsque les cases ouvertes ont des bombes voisines autour d'elles, un String correspondant au nombre de bombes voisines est affiché en plus. Dans « tableau_voisin » les cases ouvertes vont être représentées par 'o' au lieu de 'b' ou de 'v'.

- « z » : exécute la fonction « afficher_image » pour afficher un drapeau aux coordonnées de la case sur laquelle se trouve le curseur.



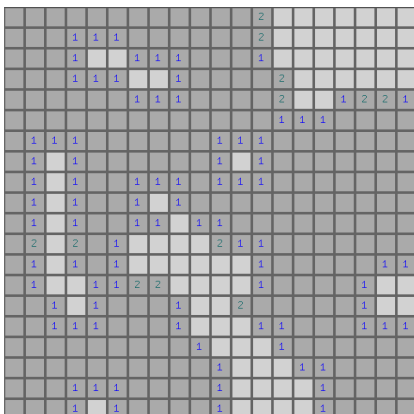
La partie se déroule jusqu'à qu'il n'y ait plus de bombes ou si le joueur a cliqué sur une bombe. A chaque fois que le joueur clique sur une case, l'algorithme parcourt le tableau « tableau_voisin » pour vérifier s'il reste des cases non vides. S'il n'y a que des cases 'b' et 'o' alors la partie est terminée. Une variable booléenne vérifiant la condition d'une boucle while dans laquelle se trouve la fonction « jeu » passe de true à false, le jeu se termine.

Cas de victoire :

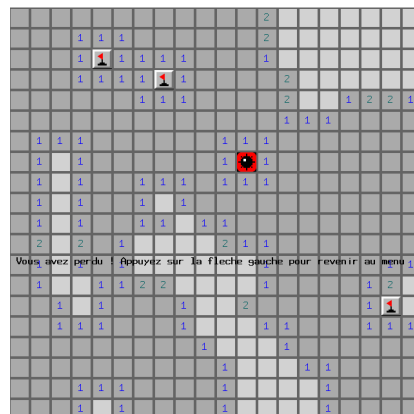
Lorsque le joueur a cliqué sur toutes les cases vides alors il a gagné. Le programme enregistre le nom du joueur ainsi que le temps de la partie (fonction « durée ») qui sont stockés dans le fichier « direct.dat » (cf. gestion des fichiers). Un message de victoire s'affiche, le joueur est renvoyé au menu principal s'il appuie sur la flèche de gauche.

Cas de défaite :

Un message signalant que la partie est terminée s'affiche, une icône de bombe s'affiche, le joueur est renvoyé au menu principal s'il appuie sur la flèche de gauche.



Partie en cours



Partie perdue



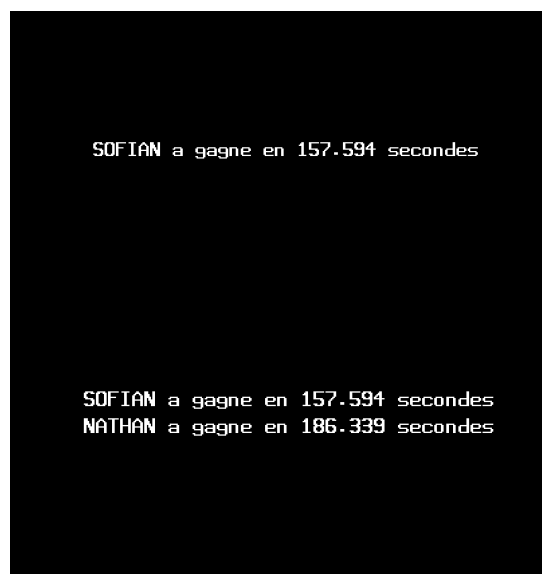
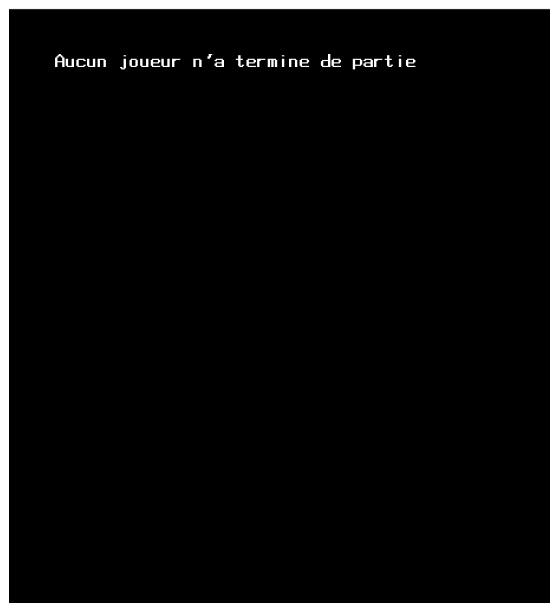
Partie gagnée

Rejouabilité

Dans le cas où le joueur veut faire plusieurs parties, l'algorithme se doit de réinitialiser les tableaux qui ont été modifiés au cours de la partie précédente. Dès que le programme sort de la boucle while de fin de partie les fonctions « reconstitution_bombe » et « reconstitution_voisin » vont réinitialiser les tableaux « tableau_bombe » et « tableau_voisin ». Les bombes seront remplacées au début d'une nouvelle partie avec les fonctions « randomisation_facile/normal/difficile ».

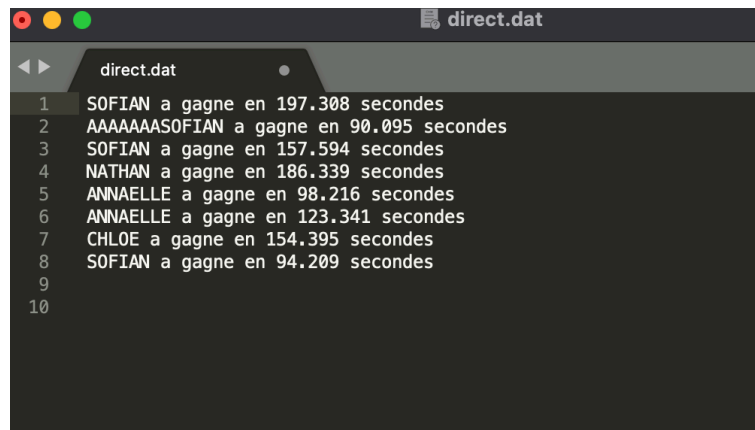
Classement des joueurs

Nous avons voulu faire un classement des joueurs en fonction de leur rapidité. Pour cela, il fallait enregistrer la durée des parties. Nous avons créé des variables double « tempsDepart » et « tempsFin » pour lesquelles on a affecté « System.currentTimeMillis » au début et à la fin de la partie. Avec la fonction « duree » on calcule le temps de la partie. On ajoute à la liste « temps_classement » le temps de partie et dans la liste « classement » le nom du joueur. A chaque fois qu'un joueur terminera une partie l'algorithme va trier ces listes en fonction de la durée des parties pour que lors du parcours de ces dernières, la fonction « leaderboard » affiche les joueurs du plus rapide au plus lent. Si les listes sont vides « leaderboard » prévient l'utilisateur qu'aucun joueur n'a terminé de partie pendant la session de jeu.



Gestion des fichiers

On utilise les fonctions du TP sur la gestion des fichiers. Ici « ajouterLignes » qui va créer un fichier « direct.dat » pour sauvegarder l'historique des parties gagnées au cours de toutes les sessions de jeu. On crée une variable « append » qui est un String qui va contenir le nom du joueur et son temps de jeu. Puis on ajoute append à direct.dat avec « ajouterLignes ». L'essentiel du travail avait été réalisé pour la création du leaderboard ainsi la gestion de fichier fut rapide.



```
direct.dat
1 SOFIAN a gagne en 197.308 secondes
2 AAAAAASOFIAN a gagne en 90.095 secondes
3 SOFIAN a gagne en 157.594 secondes
4 NATHAN a gagne en 186.339 secondes
5 ANNAELLE a gagne en 98.216 secondes
6 ANNAELLE a gagne en 123.341 secondes
7 CHLOE a gagne en 154.395 secondes
8 SOFIAN a gagne en 94.209 secondes
9
10
```

Améliorations

Plusieurs fonctionnalités peuvent être ajoutées au jeu :

Un compteur du nombre de mines restantes à trouver. L'algorithme compterait le nombre de mines ajoutées dans « tableau_bombe » au début de la partie puis à chaque drapeau posé le compteur diminuerait de 1. Il faudrait l'actualiser dès que la touche drapeau serait actionnée.

Le leaderboard pourrait être amélioré en prenant en compte la session de jeu actuelle mais également les sessions de jeu précédentes. Actuellement, il ne prend en compte que la session de jeu actuelle.

Les difficultés pourraient être en fonction du nombre de mines mais également de la taille de la grille. Plus la difficulté augmenterait, plus la taille de la grille serait grande.

Nous avons aussi voulu utiliser les cliques de la souris mais la fonction « getMouse » faisant appel à la fonction « EcranGraphique.getMouseState » ne fonctionnait pas.

Conclusion

Travailler sur ce projet en langage java fut très enrichissant. Cela nous a permis de trouver des méthodes de travail pour programmer en équipe. On a également pu imaginer ce qu'un ingénieur doit faire quotidiennement : poser, étudier et résoudre des problèmes complexes.