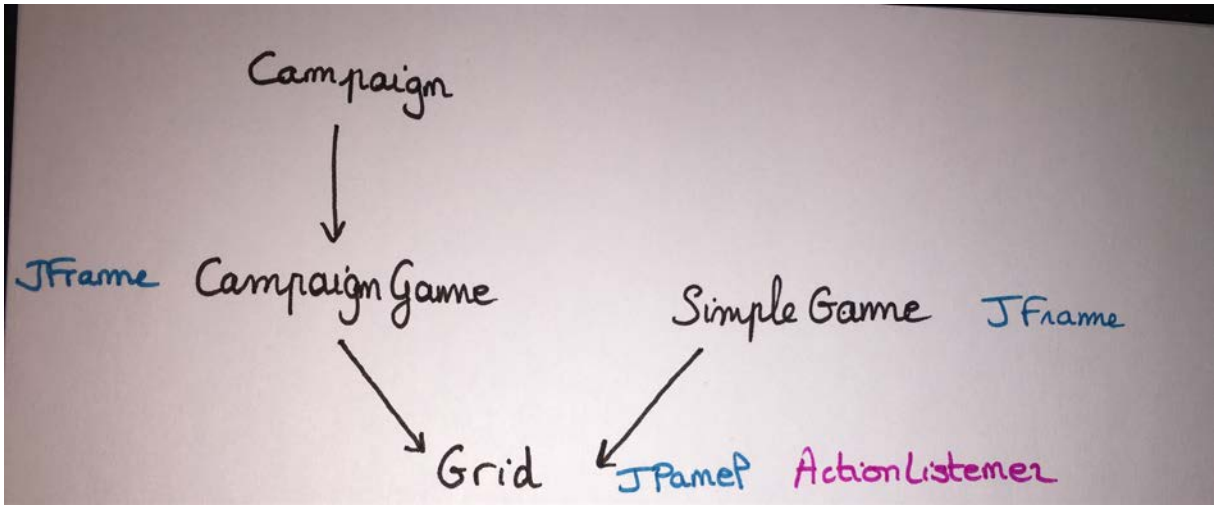
Thomas FALCONE
Nathan STROBBE

PROJET DEMINEUR

I. Fonctionnement global du jeu

Le jeu est basé sur deux modes de jeu. Le premier est un mode Carrière : on vous propose de vous lancer dans une carrière de démineur, dans laquelle vous pourrez progresser (vous commencerez ainsi par des niveaux relativement faciles puis de plus en plus compliqués). Le second mode est classique : vous pourrez faire avec celui-ci une partie simple. En termes de code ceci se traduit de la manière suivante :

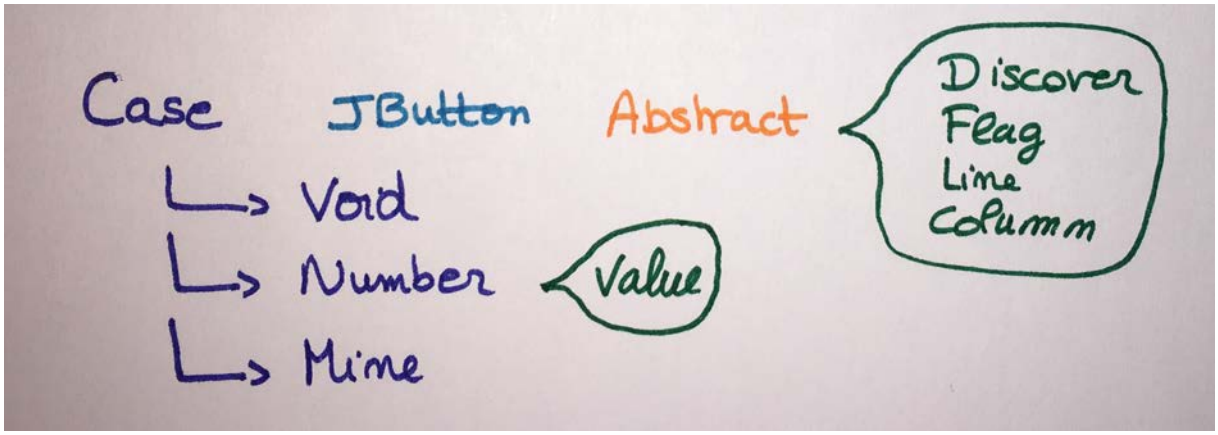


- **Campagne :**

La classe Campaign est une classe qui gère la campagne et dont les attributs évoluent au fur et à mesure des différents niveaux qu'elle crée. Ces niveaux sont des parties, instances de la classe CampaignGame (sur ce graphe, les flèches ne représentent pas des liaisons d'héritages). Cette classe est une classe qui hérite de la classe JFrame. Ainsi lorsque l'on crée un nouveau niveau, on crée une nouvelle fenêtre (que l'on prend soin de mettre en forme en accord avec notre thème).
Ce JFrame contient un JPanel qui est en réalité une classe Grid (qui va représenter une grille de case, constituant en soi l'endroit où l'on joue au Démineur).
Ce JPanel est organisé en GridLayout afin de pouvoir former une grille facilement. Dans cette grille il y a des objets de type Case (Nous verrons plus tard les détails de cette classe). Finalement, cette classe Grid implémente l'interface ActionListener afin « d'écouter » les actions que l'utilisateur fera sur les Cases.

- **Partie Simple :**

La classe SimpleGame est un équivalent de la classe CampaignGame. C'est aussi un JFrame qui contient une instance de Grid ; mais les deux classes diffèrent car elles n'ont pas le même comportement (Initialement, nous utilisions uniquement SimpleGame, pour la campagne comme pour les parties simples, mais le code devenait trop important et pas assez clair ; d'où le besoin de faire deux classes distinctes)



- **Case** :

La classe Case est une classe abstraite qui hérite de la classe JButton. Elle contient plusieurs attributs (Discover pour savoir si la case a été découverte ou non (true/false), Flag pour savoir si l'utilisateur a positionné un drapeau dessus ou non (true/false), et deux attributs servant à connaître la position de la case dans la grille). L'instance de Grid, créée dans le JFrame, contient une grille de Cases (grâce au GridLayout).

- **Void / Number / Mine :**

Ces trois classes héritent de la classe Case. Elles ont donc les mêmes attributs, et la classe Number a l'attribut « Value » en plus (permettant de connaître le nombre de mine autour de la case).

Chacune des classes va contenir une méthode que l'on a appelée révélation qui va agir différemment selon la classe (par exemple en révélant une mine on perd alors qu'en révélant une case Vide on enclenche une cascade de révélation)

- **Génération de la grille (Grid) de Cases :**

Lorsque l'on génère la grille on crée uniquement des cases de type Void puis on génère aléatoirement deux entiers (représentant la ligne et la colonne d'une certaine case). On change alors cette case en Mine. Une fois l'action répétée autant de fois qu'il le faut pour avoir le bon nombre de mine dans la grille, on passe à la génération des nombres. Pour cela on prend chaque case qui n'est pas de type Mine et on compte le nombre de mine dans les cases voisines. Si ce nombre est 0 alors on laisse la case Void sinon on crée une case Number avec comme value ce nombre.

(Aussi, une liste nommée content permet de garder le contenu de la grille. C'est une ArrayList d'ArrayList de Case (content est une ArrayList de ligne, et chaque ligne est une ArrayList)

Notre grille de case est prête pour le jeu !

- **Révélation des cases et gestion du clic droit :**

Lorsque l'ActionListener détecte un clic sur une case alors la fonction révélation de cette case est appelé. Celle-ci diffère en fonction du type de la case concernée :

Mine : Si aucun drapeau est posé sur cette case alors on perd le jeu ! La fonction renvoie True et le jeu s'arrête avec un message et la possibilité de voir la grille totalement révélée avec des indications pour comprendre notre (ou nos) erreur(s).

Number : Simplement, on change le texte de la case (qui est un simple JButton donc on peut utiliser setText()). Ainsi le joueur voit le nombre de mine autour de la case cliquée.

Void : Ici on fait un appel récursif : Lorsque l'on révèle une case vide alors on déclenche la révélation des cases voisines. Lorsqu'on trouve une case de type Number, on ne fait que la révéler sans faire de récursivité. Ainsi on a une cascade de révélation. (On a pensé à prendre en compte par un try & catch, une exception lorsque l'on sort des dimensions de la grille)

Chacune de ces révélations change l'état de l'attribut Discover pour connaître quelles sont les cases découvertes ou non. (Afin d'éviter les problèmes de double révélation et autres)

- **Gestion du clic droit :**

Pour le clic droit nous avons créé une classe : « MouseRightClickAction » qui implémente l'interface MouseListener. Ainsi lorsque l'on fait un clic droit sur une case, on change l'état de Flag. S'il y a un drapeau on l'enlève, sinon on en met un. Cela est accompagné d'une mise à jour de l'icône de la case pour que l'utilisateur puisse voir qu'il vient de poser ou d'enlever un drapeau.

- **Victoire ou Défaite ?**

La grille s'occupe de vérifier si l'utilisateur a gagné ou non, puis elle agit en conséquence :

En mode campagne, on passe au niveau suivant s'il on gagne, et on arrête la carrière en cas de défaite.

En mode classique, on arrête juste le jeu.

Dans chaque cas, on laisse au joueur la possibilité de voir ses fautes avant de quitter la fenêtre.

- **Gestion du score :**

Au démarrage du jeu, si l'on ne trouve pas de fichier Score.txt alors on en crée un qui contiendra le TOP10 par catégorie. Ce fichier contient une répétition de ce « pattern » :

Catégorie
1 0
2 0
3 0
…
10 0

(Les catégories étant Facile, Moyen, Difficile et Carrière)

Dans la classe Main il y a une variable static nommée SCORE qui est une ArrayList d'ArrayList de int. Elle contient les meilleurs scores pour chaque catégorie (chaque catégorie étant une liste).

Au démarrage du jeu, cette variable est initialisée avec uniquement des 0 si le fichier Score.txt n'est pas trouvé sinon on lit les lignes du fichier pour remplir les ArrayList.

D'accord mais il représente quoi ce score ?

En mode carrière, le score correspond au niveau que vous avez atteint, alors qu'en mode carrière c'est un temps, en seconde.

Comment s'enregistre un nouveau score ?

Pour enregistrer un nouveau score, on va comparer la valeur à enregistrer avec les valeurs de la liste dans SCORE (on prend la liste qui correspond à la catégorie du score, par exemple Facile ou Carrière).

On ajoute le score à la bonne place et on modifie le fichier Score.txt pour qu'il soit à jour. (On modifie le fichier grâce à la fonction saveScore())

Où voit-on ce score ?

Lorsque l'on se trouve dans le menu principal on peut cliquer sur Score pour voir la fenêtre des scores. Aussi, en jeu vous pouvez voir votre score dans la barre supérieure de la fenêtre (ou dans le titre de la fenêtre).

II. Problèmes et possibilité d'évolution :

Lors du codage nous avons rencontré divers problèmes comme par exemple avec le Timer qui n'était vraiment pas facile à mettre en place (en particulier pour « rafraichir » la valeur dans le JMenuBar et pour que les Items de ce menu soit positionnés correctement).

Avec plus de temps, nous aurions pu faire (du moins essayer de faire) différentes fonctionnalités :

Pour le mode carrière nous avions pensé à un système de déblocage d'outil avec la progression. Par exemple une bombe IEM pourrait être utilisées pour révéler d'un coup 9 cases (la case sur laquelle on la déclenche et les cases adjacentes). Nous voulions également mettre un compte à rebours dans le mode carrière pour corsé les choses.

L'ajout d'un peu de son aurait été parfait pour notre projet principalement basé sur Super Meat Boy.

Aussi il aurait été intéressant de gérer les niveaux du mode campagne de manière automatique et infinie (avec des formules mathématiques par exemple, générant les dimensions de la grille et le nombre de mine en fonction du niveau). Ici nous avons fait un simple tableau contenant les informations nécessaires.