TP : rétro-conception

La rétro-ingénierie (traduction littérale de l'anglais *reverse engineering*), également appelée rétro-conception ou ingénierie inversée consiste à étudier un objet pour en déterminer le fonctionnement interne ou sa méthode de fabrication.

Plusieurs objectifs peuvent être visés par cette analyse :

- comprendre le fonctionnement de cet objet, pour être en mesure de l'utiliser correctement ou de le modifier (maintenance)
- créer un nouvel objet ayant des fonctionnalités identiques à l'objet de départ, sans violer un ou des brevets
- analyser un objet produit par un concurrent dans le cadre d'une activité de veille concurrentielle.

Le but de se TP est d'analyser et de restructurer un code Java implémentant le jeu du démineur.

Le code a été réalisé dans le cadre d'un projet étudiant : le jeu fonctionne quasiment-correctement (il y a un bug difficile à trouver). Le code du jeu sans aucune modification est déposé dans le moodle.

Règles du jeu du Démineur

Un champ de mines est représenté par une grille contenant un nombre prédéfini de mines (voir image sur la page suivante). Le but du jeu est de trouver toutes les cases de la grille contenant des mines sans faire exploser une seule mine.
Le jeu est composé d'un plateau rectangulaire, d'un chronomètre et d'un compteur de mines.

Au début toutes les cases de la grille sont dite « couvertes »; le compteur de mines indique le nombre de mines à localiser. Le chronomètre compte le nombre de secondes écoulées depuis le début de la partie.

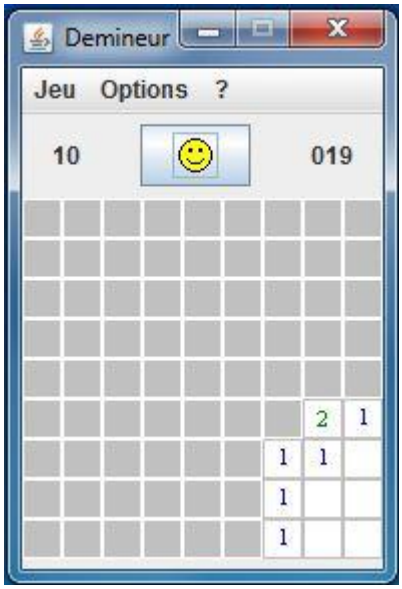
Lorsque le joueur clique sur une case, cette case est découverte et son contenu est affiché :

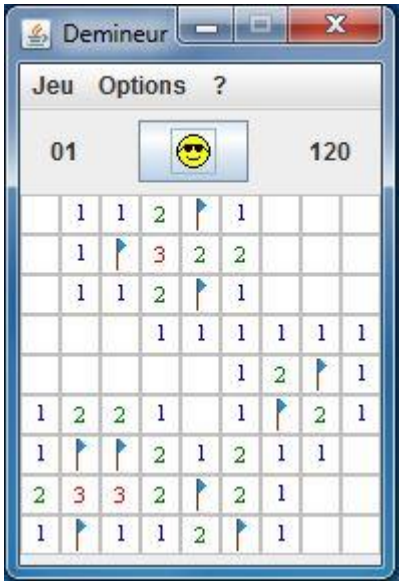
- la case est minée, le joueur a perdu.
- le contenu de la case est un chiffre. La case n'est pas minée, le chiffre (entre 1 et 8) indique le nombre de mines présentes dans les cases adjacentes. Deux cases sont dites adjacentes si elles sont voisines horizontalement, verticalement ou diagonalement.
- la case est vide. La case et les cases adjacentes ne sont pas minées. Les 8 cases adjacentes sont découvertes. Une réaction en chaîne de découverte de cases est générée.

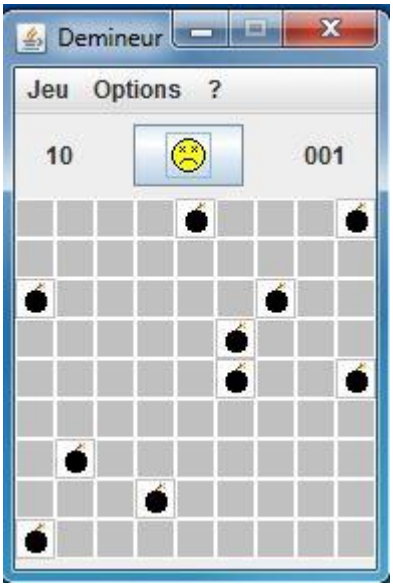
Un clic droit de la souris lorsque la souris est localisée sur une case « couverte et sans drapeau » signale que cette case contient une mine. Un drapeau est positionné sur la case et le compteur de mines est décrémenté.

Une case avec un drapeau ne peut être découverte.

Un drapeau peut être retiré par un clic droit de la souris lorsqu'elle est localisée sur la case à démarquer. Le compteur de mines est incrémenté et la case perd son drapeau.







Plan d'actions pour restructurer le code

- **Nettoyer code :** éliminer les attributs, méthodes puis classes inutiles. Fusionner les classes.
- **Restructurer** : définir les classes métier (gestion du jeu) et mettre tout le code relatif à la gestion du jeu (décision sur l'état du jeu et des cases) dans ces classes.
- **Nettoyer la structure :** éliminer les dépendances entre classes au maximum.

Nettoyer le code

1. La classe Onglet est-elle utilisée (une méthode de cette classe est-elle appelée) ?
2. La classe DButton est-elle utilisée (une méthode de cette classe est-elle appelée) ?
3. L'attribut « perdu » de la classe DPartie est-elle utile (lu) ?
4. L'attribut « commence » de la classe DPartie est-elle utile (lu) ?
5. Que calcul le code de la méthode fini() de la classe DPartie ? Cette méthode elle-t-elle appelée ?
6. L'attribut type, tailleX et tailleY de la classe DFenetre sont-ils utiles (lus) ?
7. Les méthodes getMenuPartie et getStats de la classe DFenetre sont-elles appelées ?
8. Mettez les attributs de DFenetre avec une visibilité privée (retirez les attributs inutiles).

Restructurer la partie « Gestion du jeu »

1. Quel est le rôle de la classe DPartie, après avoir retiré les éléments (attributs ou méthodes) inutilisés ? Que proposez-vous pour améliorer la conception du jeu ?
2. Eliminer la méthode getMatrice de la classe DPartie.
3. Créez un paquetage « metier » contenant les classes « métier » hors IHM, écouteur, …

Restructurer DPanneau et DImageur

1. Retirez les attributs hauteur et largeur de la classes DPanneau (utiliser les méthodes de DFenetre).
2. Donnez la liste des classes dont dépend la classe DImageur.
3. Expliquez la fonction de la méthode getImage de DImageur. Le calcul de l'état d'une case doit – il être réalisé dans le module Image ou Métier ? Nous allons reporter ce calcul dans la classe

DPartie.

a. Etablissez une énumération « EtatCase » contenant les différents états possibles d'une case du démineur.
b. Dans la classe DPartie, réalisez la méthode getEtatCase(int i, int j) retournant l'état de la case (i, j) en fonction de l'état de la partie : en cours, gagné ou perdue (aidez-vous du code de la méthode getImage de DImage).
c. Dans la classe DImageur, créez une méthode getIcon(EtatCase) retournant l'image associé à l'état (aidez-vous du code de la méthode getImage de DImage).
d. Dans la classe DPanneau, utiliser la méthode getIcon à la place de la méthode getImage.
e. La méthode getImage() de la DImageur est maintenant inutile. Après avoir retiré cette méthode, transformez la classe DImageur pour quel dépend que de EtatCase.

4. Donnez la liste des classes dont dépend la classe DPanneau. Modifier cette classe pour qu'elle dépende uniquement de DFenetre (il faut coder la méthode getIcon(i ,j) dans DFenetre).

Restructurer la partie « Gestion du jeu » - suite

1. Listez les classes contenant le nombre de cases par ligne ; celles contenant le nombre de cases par colonne, et celles contenant le nombre total de mines.
2. Modifiez le code pour que seulement la classe DPartie ou DCase contiennent les informations sur le déroulement du jeu (nombre de cases par ligne, le nombre de cases par colonne, nombre total de mines, …).
3. Eliminez la méthode getCase() de la classe DPartie.
4. Réalisez l'interface IDPartie contenant les services (méthodes) utilisés par DFenetre.

Simplifier les écouteurs

1. Retirez du code EcouteurSouris les contrôles appartenant à la partie « Métier » ; par exemple
```
« if(!partie.perdu() && !partie.gagne()
        i.  && !(partie.getMatrice().getCase(sourisY,
                                        sourisX).yaDrapreau())){ … } »
```
2. Modifiez le code pour que les classes EcouteurGo et EcouteurMenu, EcouteurSouris et Pref dépendent uniquement de la classe DFenetre.

Premier Bilan

- Nous avons créé un composant « métier » (gestion du jeu). Ce module contient toutes les actions relatives à la gestion du jeu. Les services offerts par ce module sont définies dans l'interface IDPartie et la réalisation des services est effectués dans les deux classes DCase et DPartie. DPartie dépend uniquement de DCase, qui ne dépend d'aucune classe.
- Seule la classe DFenêtre dépend de ce composant.
- La classe DImageur s'occupe uniquement de l'association d'état avec une image (cette classe dépend d'aucune classe).
- Nous avons simplifié la structure en éliminant 3 classes et en éliminant des dépendances entre les classes.

Approfondissement

En vue de réaliser un diagramme de communication, vous ajoutez au code de chaque méthode publique de DCase, DPartie et DFenetre un affichage sur la sortie standard « nom classe – nom méthode » pour lister les appels aux méthodes. Vous constatez un nombre considérable d'appels lors que l'on pose un drapeau.

Sans changer la structuration du code, réalisez des modifications dans le code diminuant le nombre d'appels aux méthodes.

Simplifier encore plus les « écouteurs »

1. Dans DFenetre, utilisez des « Lambda Expression » pour éliminer les classes EcouteurGo et EcouteurMenu.
2. Eliminez la classe EcouteurFenetre (DFenetre doit implémenter l'interface de l'API WindowsListener).

Nettoyer le code

1. Eliminez tous les « warning » indiqués par l'IDE.