POO – Programmation Orientée Objet en Java

Fiche de TP numéro 6

Un jeu de memory

Le jeu de memory consiste en un jeu de cartes avec des images. Chaque image est représentée sur deux cartes. Au départ, toutes les images sont posées sur la table, face cachée. Le joueur choisit deux cartes et les retourne. Si elles représentent la même image, il les "gagne" et elles sont retirées du jeu. Sinon, il les retourne à nouveau sur la table (face cachée). Le but du jeu est de gagner toutes les images en le plus petit nombre d'essais.

Étape 1 : *Un petit aperçu*

Vous trouverez sur Moodle un fichier `memory.jar`. Vous pouvez le tester par

```
java -jar memory.jar
```

Vous aurez ainsi un aperçu de ce qui vous est proposé ici.

Il ne s'agit que d'un exemple. Votre memory sera sûrement plus joli et plus agréable à utiliser, avec plus d'options , etc...Les images que nous avons utilisées sont celles qui sont déjà disponibles sur Moodle. Vous pouvez utiliser celles-ci, mais vous pouvez également choisir ou dessiner vos propres images. Ne soyez donc pas bridés par cet exemple. La seule condition, si vous utilisez d'autres images, est de prendre des images sous licence libre (creative commons, ...). L'objectif de la fiche TP est de vous amener à réaliser un premier jeu en ayant une réflexion sur l'organisation de votre code.

Étape 2 : *Comment organiser votre progression ?*

Tout d'abord, réfléchissez ! Les principaux points techniques nécessaires à la réalisation de ce projet ont déjà été vus en TP. Ceux qui ne l'ont pas été sont présentés dans le sujet. Enfin, le tutorial Java sur Swing et la documentation de l'API sont une mine de renseignements. Ce qui importe donc c'est la conception.

Quels sont les différents objets qui vont intervenir ? Quelles sont leurs propriétés ? Leurs comportements ? Réfléchissez d'abord au modèle.

Étape 3 : *Le modèle*

Quel est le moyen le plus simple de représenter le jeu ? Quelles sont les informations dont vous avez besoin ? Quelles sont les informations qui sont de la responsabilité du modèle ? Une fois que vous avez décidé des attributs nécessaires, passez aux méthodes. Quelles sont les questions auxquelles le modèle devra répondre et qui pourront modifier son contenu ?

Écrivez votre classe `ModeleMemory`. Dans un premier temps, on ne s'occupera pas de mélanger les cartes : le memory rangera les cartes dans l'ordre que vous souhaitez.

Étape 4 : *L'interface graphique*

Dans un premier temps, construisez un jeu sans message, sans clignotement, sans barre d'outils, sans menu, qui se construit à partir d'une instance de `ModeleMemory`. Affichez vos images en cohérence avec le modèle. (toujours dans le même ordre). N'oubliez pas que les images doivent être stockées dans le répertoire `bin`.

Étape 5 : *La gestion des événements*

Dans un second temps, occupez-vous de la gestion des événements principaux : sélectionner une case, puis lorsque la deuxième est sélectionnée prévenir le modèle. Si les cases concordent, elles doivent afficher l'image `Rien.gif`. Ne recommencez pas une partie lorsque vous avez gagné. Vous remarquerez quand même que lorsqu'une image a été retirée, le composant graphique sur laquelle elle était posée n'écoute plus les événements. Quelle méthode devrez-vous utiliser ?

Étape 6 : *Un peu de temps...*

Ne serait-ce que pour tester votre jeu, laissez-les cartes découvertes affichées un certain moment. Pour cela, vous pourrez utiliser `javax.swing.Timer`. Quelle est la méthode pour demander à un `Timer` de n'envoyer son premier mesage `actionPerformed` qu'un certain délai après l'instruction `start()` ?

Profitez-en pour gérer le clignotement. Vous remarquerez que pendant que les cartes sont affichées, les autres cartes ne réagissent plus aux "clics" de l'utilisateur. Quelle est la méthode pour signaler à un composant graphique `JButton` qu'il n'est pas "clickable" ?

Étape 7 : *Un peu d'aléatoire...*

Mélanger une collection d'objets, en Java, c'est trés simple. C'est la méthode `shuffle` qui fait tout. Le problème est que trés certainement, pour initialiser le jeu, vous voudrez mélanger un tableau, probablement un `int[]` ou un `int[][]`. Appelons `monTableau` votre tableau. En adaptant/complétant les quelques lignes de code suivantes, vous pourrez mélanger vos cartes de memory.

```
// je créé une liste d'entiers
List<Integer> l = new ArrayList<Integer>();
// je pose dans l toutes les valeurs que je veux retrouver dans monTableau
... // à écrire
// je mélange ma liste, qui est une collection
Collections.shuffle(l);
// je remplis monTableau avec les valeurs de l
... // à écrire, avec une seule boucle ou une double boucle
    // suivant que monTableau est à 1 ou 2 dimensions
```

Vous devrez importer les classes nécessaires du paquetage `java.util` pour pouvoir les utiliser. La déclaration de `l` comme une variable de type `java.util.List` est obligatoire si vous importez également tout le paquetage `javax.swing` : il vous faudra alors différencier `java.util.List` et `javax.swing.List`.

Attention, vous pourrez avoir également quelques problèmes du même ordre car il existe une classe `Timer` dans le paquetage `javax.swing` et dans le paquetage `java.util`. C'est `javax.swing.Timer` que vous voulez utiliser.

Étape 8 : *Recommencer une partie*

Il est agréable de pouvoir recommencer une partie dès que vous en avez terminé une. Ajoutez cette fonctionnalité à votre projet.

Étape 9 : *Ajouter une barre d'outils*

Ajoutez à votre programme une barre d'outils `JToolBar` avec deux commandes proposées : débuter une partie, et quitter le programme. Le plus simple est le plus réutilisable sera d'ajouter dans votre `JToolBar` des instances de `JButton` initialisées avec en paramètre une sous-classe de l'interface `Action`.

Ecrivez deux classes héritant de `AbstractAction`. Dans cette classe, vous ne devez redéfinir que la méthode `actionPerformed()`. L'une de ces classes sera chargée de réinitialiser une partie, l'autre de quitter le programme (`System.exit(0);`). Choisissez comme unique constructeur celui

qui fera appel au constructeur à deux paramètres de `AbstractAction` (public AbstractAction(String name,Icon icon)).

Étape 10 : *Une barre de menus*

Pour améliorer la convivialité de votre programme, ajoutez une barre de menu (classes `JMenuBar`, `JMenu` et `JMenuItem`, et méthode `setJMenuBar` de la classe `JFrame`). Il y a deux menus : le menu `Jeu`, qui permet de relancer une nouvelle partie, ou de quitter le programme (ce sont donc exactement les mêmes actions que celles associées à la barre d'outils) et le menu `Aide`, décrit ci-dessous.

Étape 11 : *Aide*

Ce menu ne contient qu'une entrée `À propos...` qui ouvre une fenêtre dans laquelle vous pourrez signer votre œuvre (classe `JOptionPane` et méthode statique `showMessageDialog`). Utilisez la même technique que pour la barre d'outils, et créez une sous-classe d'`AbstractAction` pour faire ce travail.

Vous pouvez maintenant afficher un message de félicitations au joueur lorsqu'il a terminé sa partie (avant d'en recommencer une autre - pas besoin d'écrire une nouvelle classe).

Étape 12 : *Le nombre d'essais et le nombre d'images*

Ajouter sous votre grille (au sud, par exemple), un conteneur avec deux étiquettes (`JLabel`) qui devront afficher le nombre d'essais actuellement faits, ainsi que le nombre d'images gagnées.

Étape 13 : *Distribuer votre jeu*

Une application java se distribue sous la forme d'un `.jar` : vous allez créer une archive java compressée qui permettra d'exécuter votre code. Placez-vous dans le répertoire `bin`, supposons que c'est la classe `Main` qui contient votre méthode `main` et lancez la commande suivante :

```
moi@mamachine:~/java/memory/bin$ jar cfe memory.jar Main *.class images
// vous créez memory.jar, en indiquant que Main est le point d'entrée,
// et en ajoutant dans l'archive tout ce dont vous avez besoin :
// les .class et les images
```

Vous pouvez tester votre archive en lançant la commande :

```
moi@mamachine:~/java/memory/bin$ java -jar memory.jar
```

L'intérêt c'est que vous pouvez maintenant déplacer `memory.jar` n'importe où et l'exécuter avec la commande `java -jar`.

Bon courage !