Université de Nice-Sophia Antipolis
PeiP2

12 décembre 2012
Durée : 1,5 mois

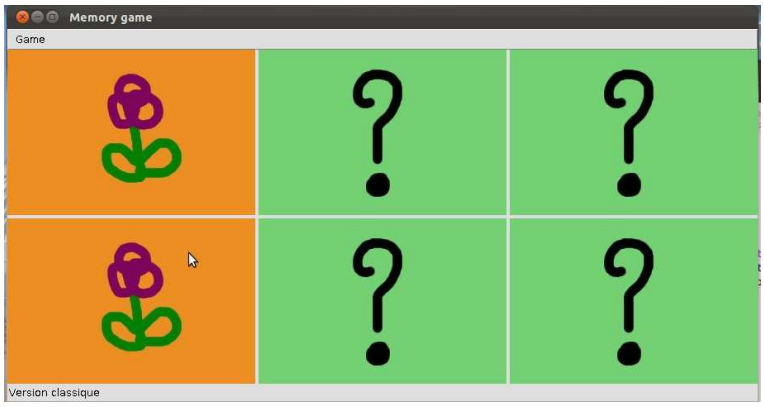
Projet Java : Variantes autour du jeu Memory

Vous allez programmer un jeu de Memory comportant plusieurs variantes. La première partie de ce travail consiste à programmer le jeu classique de Memory en respectant les contraintes données dans l'énoncé tandis que la seconde partie consiste à ajouter des variantes autour de ce jeu.

1 Le jeu de Memory

1.1 Interface graphique et classes

La figure ci-dessous présente l'interface d'un jeu de Memory. Dans ce jeu de cartes, les cartes sont présentés par paires et sont disposées faces cachées en début de jeu. Lors d'un coup, le joueur sélectionne deux cartes et les retourne. Si les cartes sont identiques, elles restent visibles, sinon, elles sont retournées. Le jeu s'arrête lorsque toutes les paires sont visibles.



La fenêtre principale de l'application s'appelle `MemoryFrame` et hérite de la classe `Frame`. Cette fenêtre doit comporter une barre de menus permettant au minimum de démarrer à tout moment une nouvelle partie ou de quitter l'application.

Dans cette fenêtre, il y a une zone de jeu et un label permettant d'afficher différents messages tels que le type de jeu (classique ou variante) ou la fin d'une partie.

La zone de jeu est représentée par la classe `MemoryPanel` héritant de `Panel`. Cette classe est responsable de la distribution des cartes. C'est dans cette zone de jeu, les cartes seront affichées.

Les cartes sont représentées par des objets de la classe `Card` héritant de `Canvas`. Une carte possède, parmi ses attributs, deux images : une image lorsque la carte est cachée (commune à toutes les cartes) et une image lorsque la carte est visible. Dans la classe `Canvas`, la méthode `paint` est redéfinie afin de pouvoir dessiner une image sur la carte.

1.2 Les images

Les images sont représentées par la classe `java.awt.image.BufferedImage`. Le code suivant permet de créer une image à partir d'un fichier (attention, ce code est susceptible de générer une exception de type `IOException`) :

```
BufferedImage b = ImageIO.read(new File("pile.jpg");
```

On donne ci-dessous le code nécessaire pour afficher une image :

```
  public void paint(Graphics g) {
    ...
   g.drawImage(b, 0, 0, null);
   ...
}
```

Les images ne sont pas fournies : c'est à vous de les choisir tout en respectant les droits associés. Vous pouvez soit les créer à partir de dessins, soit à partir de photos. Cependant, ne choisissez pas des images trop grandes. Commencez, par exemple, une résolution 300×200 pixels.

1.3 Programmation du jeu

Initialement, toutes les cartes sont cachées. Affichez l'interface dans cette configuration.

Ajoutez ensuite la gestion des événements souris sur les cartes de telle sorte que lorsque qu'une carte est cliquée, si elle était cachée, elle devient visible, sinon elle reste cachée.

Faites ensuite en sorte que lorsqu'une seconde carte est sélectionnée, elle est comparée à la précédente. Si les cartes sont identiques, elles restent visibles jusqu'à la fin de la partie, sinon, elles sont retournées. Afin d'avoir le temps de visualiser la seconde carte, on peut soit retourner la carte sur l'événement `mousePressed` et faire la comparaison lors de l'événement `mouseReleased`, soit attendre qu'une troisième carte soit sélectionnée. À vous de choisir.

Il ne reste plus qu'à détecter la fin de partie.

1.4 Finitions obligatoires

Proposez à l'utilisateur de faire une nouvelle partie. À cette effet, vous pourrez utiliser la classe(`Dialog`).

Permettez de sélectionner le nombre de lignes et de colonnes du jeu par l'intermédiaire de la barre de menus. Attention, il faut que le nombre de cartes résultant soit pair !

Le but du jeu est de terminer en un minimum de coups. Dans cette première version, le jeu est mono-utilisateur et vous garderez une trace (dans un fichier) du nombre de coups (le score) effectué par l'utilisateur pour reconstituer les paires de cartes. Vous conserverez les 5 meilleurs de score de l'utilisateur. A chaque fin de nouvelle partie, votre programme indiquera à l'utilisateur son score et sa progression (ou sa régression). Votre programme permettra aussi de réinitialiser la liste des scores.

2 Variations

On propose ici d'ajouter différentes variantes au jeu de Memory afin de le rendre plus difficile. Vous devez choisir une variante parmi les deux premières variantes proposées et en proposer une de votre invention. Quelques idées vous sont données et n'hésitez pas à proposer vos propres idées à votre encadrant de TD.

2.1 Transformation des images

À chaque fois qu'une carte est retournée, l'image sera dégradée à son prochain affichage. Plusieurs transformations sont données dans la classe `TransfoImage` que vous pouvez télécharger à l'adresse suivante : `http://www.polytech.unice.fr/~lingrand/TransfoImage.java`. Vous avez tout à fait le droit de modifier ensuite ce fichier.

Dans ce fichier, des constantes spécifient le type de transformations et la méthode `filter` les applique. On notera que cette méthode est une méthode statique (`static`) et ne nécessite pas la construction d'un objet pour être utilisée.

Voici un exemple de code qui applique un filtre moyenneur de taille 5×5 sur l'image `binput` de type `BufferedImage`. La méthode `filter` renvoie une nouvelle image transformée qui est affectée à la variable `boutput` de type `BufferedImage`.

```
BufferedImage boutput = TransfoImage.filter(binput, TranfoImage.Filter.MOYENNEUR5x5);
```

On pourra choisir une transformation pour toute la partie ou bien changer de transformation à chaque nouvelle sélection d'une image, ou encore choisir la transformation de façon aléatoire.

2.2 Déplacement des cartes

À chaque paire manquée, toutes les cartes sont déplacées (par exemple, d'un cran vers la droite puis de haut en bas, ...), ou bien les deux cartes sélectionnées sont inversées. On prendra soin de seulement modifier l'arrangement des cartes dans le `Layout` du `MemoryPanel`.

2.3 Et vous ?

Que proposez-vous ? Si vous manquez d'idées, en voici quelques unes :
- D'autres transformations sur les images.
- Plutôt que des paires, on doit retrouver des triplets, des quadruplets, ... de cartes.
- Plutôt que de faire évoluer les transformations précédentes à chaque paire manquée, on pourrait utiliser un chronomètre et appliquer les transformations à intervalle de temps constant (toutes les 20 secondes, par exemple).
- Gérer plusieurs utilisateurs et des pénalités ayant des conséquences sur les transformations appliquées.

3 Remise du projet

Votre projet est à faire en binôme *intra groupe d'info*, pas de trinôme, donc au plus un monôme. Il est à rendre au plus tard :

vendredi 25 janvier 2013 à 16h – aucun délai ne sera accordé –

sous forme d'une archive `nom1-nom2-groupe.jar` que vous déposerez sur le site moodle :

`http://moodle.i3s.unice.fr/course/view.php?id=23`.

Le rendu se présente sous la forme d'une archive JAVA exécutable (`.jar`) comprenant :
- les fichiers sources (`.java`)
- et compilés (`.class`)
- ainsi qu'un fichier pdf `Readme.pdf` dans lequel vous décrirez de façon succinte les choix que vous avez fait (variante, ...).

On rappelle que le code source doit être correctement indenté, commenté et qu'il doit être clair et lisible (noms de variables adaptées, ...).

Il ne devrait pas être nécessaire de rappeler que le travail doit être personnel et que toute ressemblance entre des projets sera sévèrement sanctionnée. Mieux vaut donc un projet modeste personnel qu'un très beau projet copié.

Il ne devrait d'ailleurs pas être nécessaire de rappeler que le but premier d'un tel projet est de vous faire progresser en programmation en vous confrontant à une expérience de plus grande envergure qu'un simple TP.