Université de Nice-Sophia Antipolis — mardi 5 mai 2008
POLYTECH CiP2 — Durée: 3h

Algorithmique et Porgrammation
Travaux Pratiques – Séance n° 8

Toiles, contextes graphiques et événements souris

Dans ce TD vous allez programmer un petit jeu d'adresse. Il s'agit de déplacer avec la souris une balle sur damier de $n \times n$ cases en partant de la case située au coin nord-ouest vers la case située au coin sud-est sans toucher les obstables disposés aléatoirement sur le damier. La figure 1 donne une vue du damier, de la balle et des obstacles.

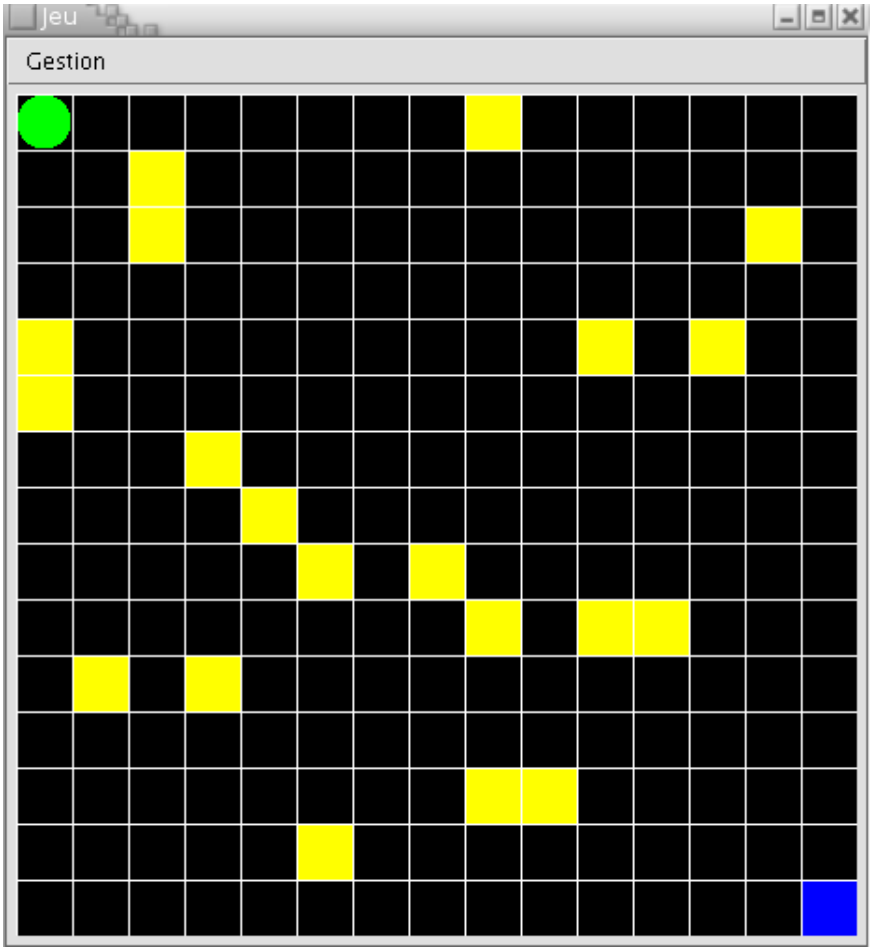

FIG. 1 – Le damier avec la balle et les obstacles

1 Une toile pour dessiner

La classe `Canvas`, héritière de `Component`, définit une zone graphique dans laquelle il est possible de dessiner. Comme tous les composants graphiques, la classe `Canvas` possède une méthode `paint` qui est appelée chaque fois qu'il est nécessaire de redessiner la fenêtre. Une classe héritière de la classe `Canvas` peut dès lors redéfinir la méthode `paint` et afficher toutes les informations voulues. Canvas

Classe `Canvas`
`Canvas()` Construit une toile vide.
`void paint(Graphics gr)` Dessine le contenu de la toile à l'écran dans le contexte graphique `gr`.

Lors de l'appel à `paint` par AWT, un *contexte graphique* (un objet de la classe `Graphics`) est transmis en paramètre. Il est possible d'afficher un texte ou des figures géométriques (lignes, rectangles, arcs de cercle, *etc.*) dans un contexte graphique. Un contexte graphique possède son origine dans le coin supérieur gauche. L'unité utilisée est le *pixel*, c'est-à-dire un point lumineux à l'écran. La matrice de pixels de vos écrans est par exemple, 800×600 ou 1024×768 pixels. Vérifiez-la. contexte graphique `Graphics` pixel

Pour dessiner dans un `Graphics`, on commence par préciser la couleur à utiliser grâce à la méthode `setColor`. On peut ensuite faire appel à l'une des méthodes `draw`*XXX* qui dessinera l'élément désiré en utilisant la couleur fixée.

Classe `Graphics`
`void setColor(Color couleur)` Fixe la couleur à utiliser pour dessiner les objets.
`void drawString(String s, int x, int y)` Affiche le texte s au dessus, à droite du point (`x`, `y`).
`void drawLine(int x1, int y1, int x2, int y2)` Dessine une ligne du point (`x1`, `y1`) au point (`x2`, `y2`).
`void drawRect(int x, int y, int lx, int ly)` Dessine le contour d'un rectangle dans la zone (`x`, `y`) à (`x + lx`, `y + ly`).
`void drawRoundRect(int x, int y, int lx, int ly, int dx, int dy)` Dessine le contour d'un rectangle aux bords arrondis dans la zone (`x`, `y`) à (`x + lx`, `y + ly`). `dx` et `dy` représentent les diamètres horizontaux et verticaux des arcs des quatre angles.
`void drawOval(int x, int y, int lx, int ly)`
Dessine le contour d'une ellipse dans la zone (`x`, `y`) à (`x + lx`, `y + ly`).
`void drawArc(int x, int y, int lx, int ly, int a, int la)` Dessine un arc d'ellipse de centre (`x`, `y`) recouvrant un rectangle de dimension `lx` $\times$ `ly` entre l'angle `a` et l'angle `a + la` (en degrés).
`void drawPolygon(int[] x, int[] y, int n)` Dessine le contour d'un polygône à `n` points formé des segments (`x[i-1]`, `y[i-1]`) à (`x[i]`, `y[i]`).
`void drawPolyline(int[] x, int[] y, int n)` Dessine une ligne brisée à `n` points formée des segments (`x[i-1]`, `y[i-1]`) à (`x[i]`, `y[i]`).

`void fillRect(int x, int y, int lx, int ly)` Dessine un rectangle dans la zone (`x`, `y`) à (`x + lx`, `y + ly`).
`void fillRoundRect(int x, int y, int lx, int ly, int dx, int dy)` Dessine un rectangle aux bords arrondis dans la zone (`x`, `y`) à (`x + lx`, `y + ly`). `dx` et `dy` représentent les diamètres horizontaux et verticaux des arcs des quatre angles.
`void fillOval(int x, int y, int lx, int ly)` Dessine une ellipse dans la zone (`x`, `y`) à (`x + lx`, `y + ly`).
`void fillArc(int x, int y, int lx, int ly, int a, int la)` Dessine un secteur d'ellipse de centre (`x`, `y`) recouvrant un rectangle de dimension `lx` × `ly` entre l'angle `a` et l'angle `a + la` (en degrés).
`void fillPolygon(int[] x, int[] y, int n)` Dessine un polygône à `n` points formé des segments (`x[i-1]`, `y[i-1]`) à (`x[i]`, `y[i]`).

Nous allons utiliser la classe `Canvas` pour définir le damier de jeu.

Un damier est une zone rectangulaire de taille `largeurDamier` × `hauteurDamier` exprimée en nombre de cases. Chaque case du damier est un carré dont le coté mesure `COTÉCASE` pixels.

Écrivez une classe `Damier` héritière de `Canvas`. Définissez la constante entière `COTÉCASE` égale à 30 pixels. Ajoutez les attributs `largeurDamier` et `hauteurDamier`, ainsi qu'un constructeur à deux paramètres qui initialise le damier avec les dimensions voulues. Dans le constructeur de `Damier`, faites appel à `setSize` pour préciser les dimensions du canevas.

1) Définissez la méthode `paint` dans la classe `Damier` qui dessine le quadrillage du damier blanc sur fond noir de `largeurDamier` × `hauteurDamier` cases. Cette méthode possède l'en-tête suivant :

```
public void paint(Graphics g)
```

2 La fenêtre principale

2) Créez une classe `Jeu` héritière de `Frame`. Dans le constructeur de `Jeu`, créez un `Damier` de dimension 15 × 15. Appelez les méthodes `pack` et `setVisible` de `Jeu` pour afficher votre fenêtre aux dimensions du damier.

3) Ajouter à la classe `Jeu` la méthode `main` qui crée un objet de type `Jeu`. Testez votre programme.

4) Dans la classe `Jeu`, placez au-dessus de la fenêtre une barre de menus qui contient le menu *Gestion*. Dans le premier menu, ajoutez l'entrée *Recommencer* et *Quitter*. Associez à cette dernière entrée l'action d'achèvement de l'application (*i.e.* `System.exit`).

3 Les cases Obstables, Arrivée

Maintenant, nous allons modifier l'application pour afficher dans le damier les cases obstables et arrivée. Ces cases seront représentées par la classe `Case` et distinguées par leur couleur : jaune pour les obstacles et bleu pour la case d'arrivée.

5) Déclarez la classe `Case`. Ajoutez l'attribut `fond` de type `Color` qui définit la couleur

de la case, l'attribut `orig` de type `Point` associé au coin nord-ouest de la case et les attribut de type entier `largeur` et `longueur`.

6) Définissez le constructeur qui initialise les attributs et ajoutez les méthodes :

```
public Point origine() {...}
public void dessiner(Graphics g);
```

qui retourne le point origine de la case, et permet de dessiner la case dans le contexte graphique `g` selon la couleur `fond`. La classe `Point` est celle que vous avez déjà utilisée dans le TD sur les figures.

7) Dans la classe `Damier`, ajoutez les attributs suivants :

```
private static int NBOBSTACLES = 20;   // nombre maximum d'obstacles
private Case[] lesObstacles;           // tableau des obstacles
```

8) Dans le constructeur de `Damier`, créez le tableau `lesObstacles` de dimension `NBOBSTACLES`. Créez le tableau de cases obstacles, leurs positions sont choisies aléatoirement.

9) Créez la case d'arrivée.

10) Dans la méthode `paint` de `Damier`, parcourez le du tableau `lesObstacles` et, pour chaque case, appelle la méthode `dessiner`. Compilez votre programme et vérifiez que l'affichage des cases obstacles et arrivée est correct.

4 La balle

La balle est un mobile que l'on doit déplacer sur le damier. Elle est représentée par un cercle de couleur verte. Son point de départ est la case nord-ouest.

La balle est représentée par la classe `Balle` qui possède trois attributs, `fond` la couleur de la balle, `orig` le point d'origine du cercle, et `diamètre` le diamètre du cercle.

Puis, définissez les méthodes suivantes :

```
// déplace le point d'origine de la balle en (x,y)
public void déplacer(int x, int y) {...}
// teste si le point (x,y) est dans la balle ou non
public boolean appartient(int x, int y) {...}
// retourne la distance entre le point p et
// le point d'orgine de la balle
public double distance(Point p) {...}
// dessine la balle dans le Graphics g
public void dessiner(Graphics g);
```

5 Les événements souris

Maintenant, nous voulons déplacer la balle dans le damier à l'aide de la souris. Pour cela, nous devons :

– sélectionner la balle à déplacer lorsque l'utilisateur appuie sur le bouton gauche de la souris ;

– déplacer la balle avec le pointeur de la souris tant que l'utilisateur maintient le bouton enfoncé ;
– libérer la balle quand l'utilisateur relâche le bouton de la souris.

Comme les événements de type « bouton sélectionné », les événements « souris » sont interceptés par des auditeurs. Il existe deux types d'auditeurs pour les événements souris. L'interface `MouseListener` intercepte les événements de bouton appuyé ou relâché, tandis que l'interface `MouseMotionListener` s'intéresse aux mouvements du pointeur de la souris.

`MouseListener`
`MouseMotionLis`

Complétez la classe `Damier` avec les méthodes `mousePressed` et `mouseDragged` pour sélectionner et déplacer la balle sur le damier. Ces deux méthodes possèdent comme paramètre un événement de type `MouseEvent` à partir duquel on récupère les coordonnées du pointeur de souris avec les méthodes `getX` et `getY`.

La méthode `mousePressed` doit déterminer si la position du pointeur de la souris est sur la balle pour la sélectionner. Vous utiliserez la méthode `appartient` définie précédemment.

la méthode `mouseDragged` utilise la méthode `déplacer` pour déplacer la balle. C'est aussi cette méthode qui devra tester si le mobile heurte un obstable ou atteint la case d'arrivée. Dans un cas comme dans l'autre l'application devra signaler l'événement.

6 En savoir plus sur l'AWT

L'AWT est très riche et nous n'avons eu le temps que d'en découvrir une petite partie. Les composants graphiques que nous avons utilisés héritent tous de la classe `Component`. Ils héritent ainsi des caractéristiques générales des fenêtres, comme la possibilité de changer leur couleur (`setForeground` et `setBackground`), leur dimension (`setSize`), *etc*.

`setForeground`
`setBackground`
`setSize`

Il est bien entendu possible de manipuler plusieurs fenêtres. Les méthodes de la classe `Window` permettent de manipuler les fenêtres principales (`toFront` ou `toBack` pour faire passer la fenêtre en avant-plan ou en arrière-plan, `dispose` pour se débarrasser d'une fenêtre, *etc*). L'auditeur `WindowListener` permet de définir le comportement d'une fenêtre lorsqu'elle reçoit des événements de réduction en icône, de fermeture, *etc*.

`toFront`
`toBack`
`dispose`
`WindowListener`

Les classes `TextArea` et `TextField` héritières de `TextComponent` permettent à l'utilisateur de saisir un texte ou une ligne de texte. La classe `Dialog` est utile pour créer des fenêtres de type « confirmation », telles que « voulez-vous sauvegarder avant de quitter ? » et attendre la réponse de l'utilisateur.

`TextArea`
`TextField`
`Dialog`

Il existe une classe d'agencement plus riche (mais aussi plus complexe à utiliser) que celles que nous avons vues jusqu'à présent. La classe `GridBagLayout` permet d'agencer des composants selon un damier irrégulier.

`GridBagLayout`

Pour plus de détails, vous êtes invités à consulter la documentation de l'API Java.

7 Les *applets*

Les applets applets sont des petites applications graphiques JAVA intimement liées au World Wild Web[1]. Contrairement aux applications graphiques précédentes qui s'exécutent de façon autonome, les applets nécessitent une autre application graphique pour leur exécution, un visualisateur spécialisé (appletviewer) ou un navigateur WWW. Leur contexte d'exécution est celui d'un document HTML interprété par le visualisateur chargé de l'exécution de l'applet. Le chargement de l'applet dans le document HTML se fait avec la commande `applet`. Chaque applet a les moyens d'accéder à des informations sur son contexte d'exécution, en particulier sur ses paramètres de chargement ou sur les autres applets du document.

Les composants graphiques habituels (boutons, labels, canevas, *etc.*) d'AWT ou Swing peuvent être utilisés pour la construction des applets, et réagissent normalement aux événements (clics de souris, *etc.*). Les applets peuvent également traiter des images (généralement au format *gif*) et des documents sonores (généralement au format *au*), évidemment si l'environnement d'exécution dispose d'un dispositif de reproduction sonore.

Généralement placées sur des serveurs WWW, les applets s'exécutent localement dans le navigateur de utilisateur après leur téléchargement à travers le réseau. Ce type de fonctionnement impose bien sûr des règles de sécurité pour garantir l'intégrité de la machine de l'utilisateur. Par exemple, une applet ne peut, par défaut[2], lire ou écrire des fichiers ou exécuter des programmes de la machine client. L'action d'une applet est limitée par un composant spécifique du navigateur (SecurityManager) qui en assure le contrôle.

Il est très facile de transformer votre programme en une applet. Comme une applet n'est pas un programme indépendant, la procédure `main` devient inutile. Une classe qui veut être une applet doit hériter de la classe `Applet` et redéfinir la méthode `start`. Lorsqu'un programme lit une page HTML contenant une applet, il provoque un appel à `start` pour débuter son exécution. Plus précisément, la classe héritière de `Applet` peut redéfinir quatre méthodes : Applet

- `init` est appelée lorsque la page HTML est lue pour la première fois ; `init`
- `start` est appelée juste après `init`, ou chaque fois que la page HTML est relue ; `start`
- `stop` est appelée lorsque l'utilisateur quitte la page HTML (en chargeant une nouvelle page par exemple) ; `stop`
- `destroy` est appelée lorsque l'utilisateur quitte le programme qui exécute l'applet. `destroy`

La méthode `main` devient donc inutile. En revanche, la fenêtre principale de l'appli-

[1] Également appelé WWW, W3, Web ou encore la Toile. L'idée fondamentale de WWW est de fournir à la consultation, par l'intermédiaire de serveurs spécialisés, des documents de type *hypertexte*, c'est-à-dire contenant des références, appelées *liens*, à d'autres documents, eux-mêmes consultés de la même manière. Il suffit de cliquer sur un lien pour voir apparaître la page vers laquelle pointait ce lien. Ces documents sont des fichiers rédigés dans le langage de commande HTML (HyperText Markup Language). Ce langage est formé d'un ensemble de commandes dispersées dans un texte ordinaire, qui permettent d'une part de placer des indications de présentation, d'autre part de placer des documents graphiques ou sonores, ou encore des programmes à exécuter, enfin de placer des références à d'autres documents. L'accès au WWW se fait à l'aide d'un navigateur (ou butineur), qui en général fournit une interface graphique.

[2] Elle a la possibilité de le faire si elle possède une autorisation.

cation doit hériter de `Applet`.

11) Copiez le fichier `Jeu.java` dans `AppletJeu.java` et renommez la classe `Jeu` en `AppletJeu`.

12) Ajoutez la directive

```
import java.applet.Applet;
```

en tête de `AppletJeu.java`. Faites hériter `AppletJeu` de `Applet` au lieu de `Frame`. Remplacez le constructeur de `AppletJeu` par la méthode public **void** `init()`.

13) Une applet ne peut pas utiliser de barre de menu. Supprimez les menus et remplacez chaque entrée par un bouton qui pourra faire appel au même auditeur que l'entrée associée.

14) Supprimez l'appel à la méthode `pack` qui n'est pas définie pour les applets. Compilez votre programme.

Il faut ensuite écrire un page HTML qui appelle l'applet.

15) Utilisez le modèle suivant pour créer le fichier `AppletJeu.html` :

```
<html>
  <body>
    <h1>Applet Jeu</h1>
    <OBJECT CODE=AppletJeu.class WIDTH=500 HEIGHT=500>
    </OBJECT>
  </body>
</html>
```

16) Visualisez votre applet à l'aide d'un navigateur. Vous pouvez aussi utiliser le programme `appletviewer` livré avec l'environnement JDK. `appletviewer`