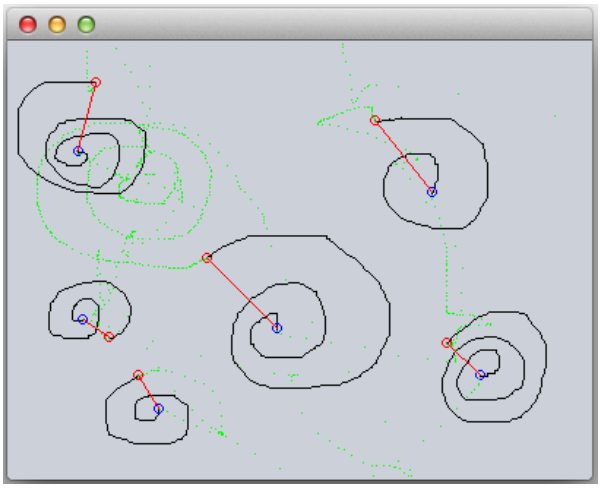
Série E : Les événements de la souris

Table des matières

Exercice E.1: Premiers essais



Pour apprendre à connaître les événements de la souris, faisons quelques essais simples directement sur le canevas de la fiche. Plus tard, nous allons intégrer ces événements dans un projet plus grand et dessiner sur un panneau en suivant le schéma MVC.

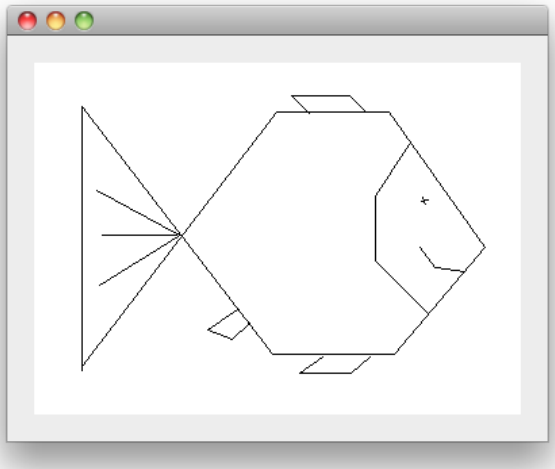
Remarques :

- Il est clair que nos dessins dans cet exercice disparaissent dès que la fenêtre est minimisée ou recouverte par une autre fenêtre. Pour garder les dessins, il faudrait définir quelques classes pour mémoriser la liste des points intervenant dans le dessin. C'est ce que nous allons faire seulement dans un prochain exercice.
- Comme nous ne mémorisons pas nos figures, nous dessinons directement lors des événements de la souris (et non pas dans la méthode **`paintComponent`**).
- Pour avoir accès au canevas, il faut employer **`getGraphics()`**. Vous pouvez employer **`getGraphics()`** dans chaque instruction de dessin. Pour simplifier, vous pouvez aussi définir et initialiser une variable **`g`** au début de chaque événement de la souris par : **`Graphics g = getGraphics();`**

1. Dessinez un point rouge à la position où un bouton de la souris est enfoncé et un point bleu à la position où un bouton de la souris est relâché.
2. Dessinez un cercle rouge (diamètre 6 pixels) autour de la position où un bouton de la souris est enfoncé et un cercle bleu autour de la position où un bouton de la souris est relâché.
3. Lorsque la souris bouge, dessinez un point noir à chaque position par laquelle passe la souris lorsque le bouton de la souris est enfoncé.
4. Lorsque la souris bouge, dessinez un point vert à chaque position par laquelle passe la souris et aucun bouton de la souris n'est enfoncé.
5. Quelle est la différence entre **`mousePressed`** et **`mouseClicked`** ?[1]
6. Mode '*Scribble*' : avec le bouton de la souris, il doit maintenant être possible de gribouiller (DE : kritzeln) sur le canevas.
   C.-à-d : Lorsque la souris bouge et le bouton de la souris est enfoncé, au lieu de dessiner des points noirs, tracez une ligne noire entre la position actuelle de la souris et la position précédente. Pour ce faire, il faut mémoriser la position actuelle de la souris dans un attribut (p.ex. **`private Point lastDragPosition;`**). (Chaque gribouillage est terminé des deux côtés par les cercles bleus et rouges.)
7. Mode '*Draw*: avec le bouton de la souris, il doit maintenant être possible de tracer une ligne rouge entre la position où le bouton a été enfoncé et la position où il est relâché. La ligne devient visible lorsqu'on relâche le bouton.
   C.-à-d : Pour ce faire, il faut mémoriser la position où le bouton a été enfoncé. (p.ex. **`private Point mousePressedPosition;`**).

1 **mouseClicked** ne figure pas au programme et nous n'en aurons pas besoin pas dans nos exercices. Cette question sert à mieux détecter l'erreur si un jour votre programme ne réagit pas de façon prévue parce que vous avez utilisé **mouseClicked** par malveillance.

Exercice E.2: Traceur de lignes 1



Sachant que :

- l'événement **mousePressed** intervient lorsque l'utilisateur enfonce le bouton de la souris,
- l'événement **mouseDragged** intervient lorsque l'utilisateur bouge la souris **avec un bouton maintenu enfoncé**,
- l'événement **mouseReleased** intervient lorsque l'utilisateur relâche un bouton de la souris,
- un objet du type **MouseEvent** possède une méthode **getPoint()** qui retourne la position actuelle de la souris. Le résultat est du type **Point**.

Créez une application permettant de tracer à l'aide de la souris des lignes sur un composant du type **JPanel**.

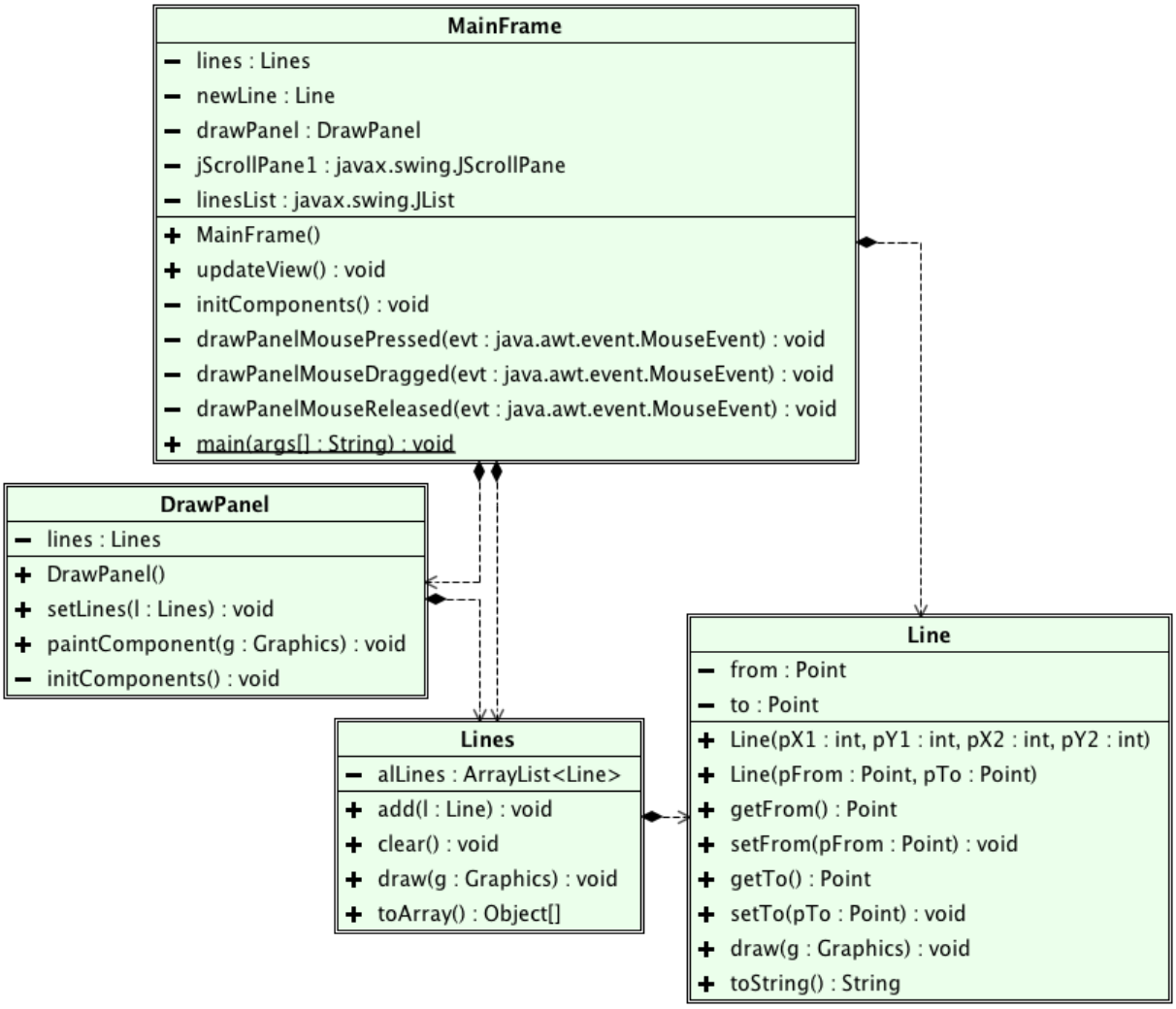
Principe de fonctionnement (Voir aussi le diagramme UML à la prochaine page)

Procédez de façon analogue à l'exercice **Turtles** :

- La classe **Line** sait représenter une ligne. La classe **Line** possède une méthode **draw(Graphics g)** pour se dessiner sur un canevas **g**.
- La classe **Lines** sait gérer une liste de lignes dans une **ArrayList** nommée **alLines.** La classe **Lines** possède une méthode **draw(Graphics g)** pour dessiner toutes les lignes sur un canevas **g**.
- La fiche principale **MainFrame** est le contrôleur : elle possède un attribut du type **Lines** qu'elle initialise au démarrage. Elle possède aussi un objet **drawPanel** du type **DrawPanel**. **MainFrame** réagit aux événements de la souris qui sont effectués sur **drawPanel**. Les lignes sont créées dans **MainFrame** qui les ajoute immédiatement à l'objet **lines**. Après chaque modification dans **lines**, elle fait redessiner **drawPanel**.
- **DrawPanel** est la vue : elle possède un attribut du type **Lines** dont elle a besoin pour dessiner les lignes. **DrawPanel** obtient les lignes de la fiche principale par un modificateur **setLines**. **DrawPanel** dessine toutes les lignes dans sa méthode **paintComponent(Graphics g)**.
- Une nouvelle ligne est créée lorsque le bouton est enfoncé. Les coordonnées de la ligne sont modifiées aussi longtemps que le bouton reste enfoncé et elle est redessinée chaque fois que ses coordonnées changent. (→ De cette façon, nous voyons la ligne déjà lorsque nous sommes en train de la tracer avec la souris)

On a donc deux fois un attribut **lines** : une fois dans **DrawPanel**, l'autre dans **MainFrame**, mais les deux doivent référencer (pointer sur) le même objet ! **MainFrame** effectue la gestion et le contrôle de **lines**, tandis que **DrawPanel** fait dessiner les lignes sur son canevas.

Amélioration (déjà intégrée dans le diagramme UML) :



Lorsque vous savez dessiner des lignes, ajoutez une **`JList`** à **`MainFrame`** (nommée **`linesList`**) pour afficher les coordonnées des lignes contenues dans **`lines`**. Pour cela, vous devez définir :

`Line.toString()`,

`Lines.toArray()`.

En plus, c'est pratique de définir dans **`MainFrame`** une méthode **`updateView`** qui redessine **`drawPanel`** et ré-affiche le contenu de **`lines`** dans **`linesList`**. Cette méthode **`updateView`** est alors appelée après chaque changement de **`lines`** (au lieu de **`repaint`**).

Classes requises : Point, JPanel, Graphics, ArrayList

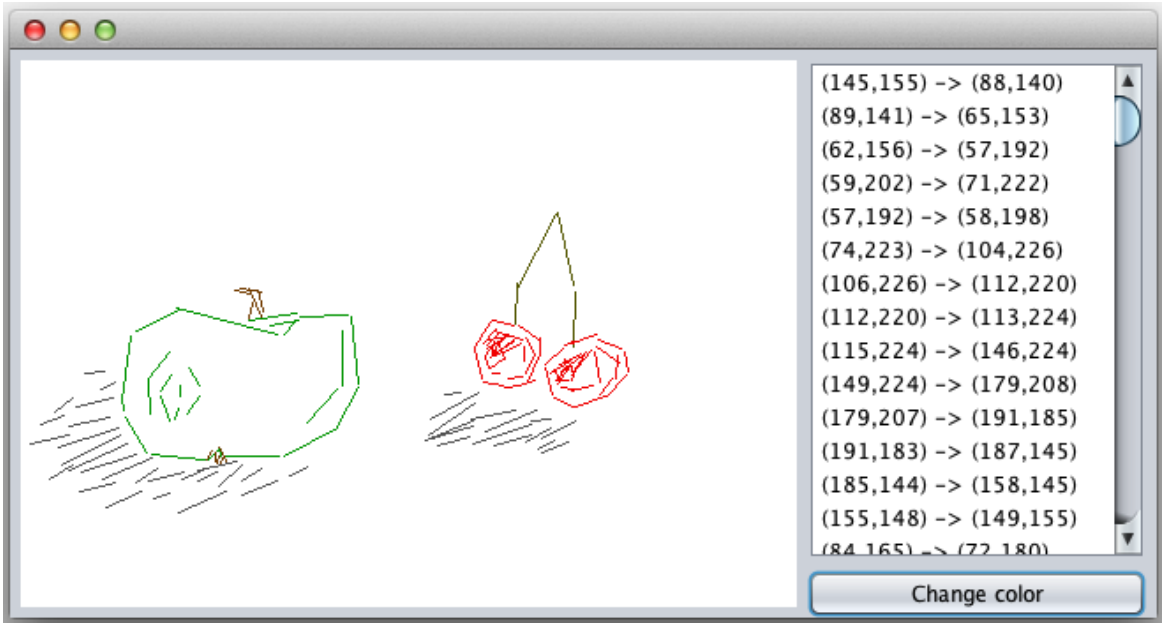
Exercice E.3: Traceur de lignes en couleur

Effectuer les actions suivantes :

- Ajoutez à la classe **`Line`** la propriété **`color`** (du type **`Color`**) permettant de stocker la couleur de la ligne ! Initialisez la couleur de la ligne lors de sa création. Faites dessiner chaque ligne dans ***sa*** propre couleur !
- Ajoutez à la classe **`MainFrame`** la propriété **`drawColor`** (du type **`Color`**) permettant de mémoriser la couleur utilisée pour la création des nouvelles lignes.
- La ligne de code suivante affiche un dialogue permettant de choisir une nouvelle couleur et de sauvegarder cette dernière dans une variable **`newColor`** :

```
Color newColor = JColorChooser.showDialog(this,"Choix d'une couleur", oldColor)
```

- Ajoutez un bouton '*Change color*' pour modifier la couleur de **`drawColor`**.



Pour avancés : Un double clic sur la surface de dessin efface toutes les lignes de la liste.

Classes requises : Point, JPanel, Graphics, ArrayList, Color

Notions requises : ajout de propriétés, intégration d'appels à des méthodes inconnues

Exercice E.4: Traceur de figures

Copiez l'exercice 'Traceur de lignes en couleur' et améliorez-le de la manière suivante:

- le bouton gauche de la souris permet de dessiner des lignes
- le bouton droit de la souris permet de dessiner des rectangles

Quels problèmes constatez-vous?

Notions requises : définition et réarrangement de classes

Exercice E.5: Checkers

Reprenez l'exercice "Damier" (**Checkers** - cours 2e, chapitre dessin) et permettez le déplacement des pions à l'aide de la souris (d'abord sans tester la validité des mouvements).

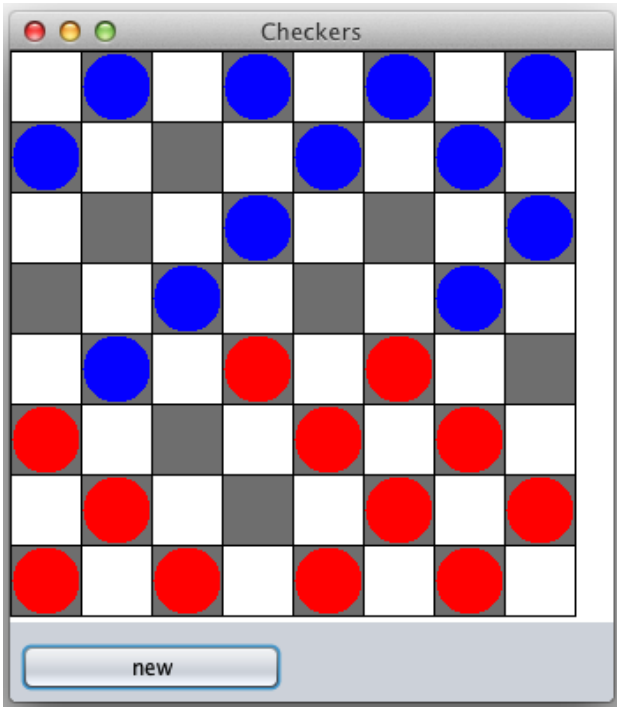
- Utilisez la version du damier qui n'est pas centré sur le canevas.
- Dans **Checkers**, remplacez la méthode init par un constructeur qui a la même fonctionnalité. Dans MainFrame, remplacez les appels de **init** par des appels du constructeur (et n'oubliez pas de passer le nouvel objet à **DrawPanel**).
- Les événements de la souris sont évalués dans **MainFrame**. Pour que **MainFrame** puisse calculer facilement les coordonnées du champ (colonne 0-7, ligne 0-7) à partir des coordonnées (x, y) de la souris, il est utile de recalculer dans **DrawPanel.paintComponent** le côté d'une case et d'ajouter la méthode :

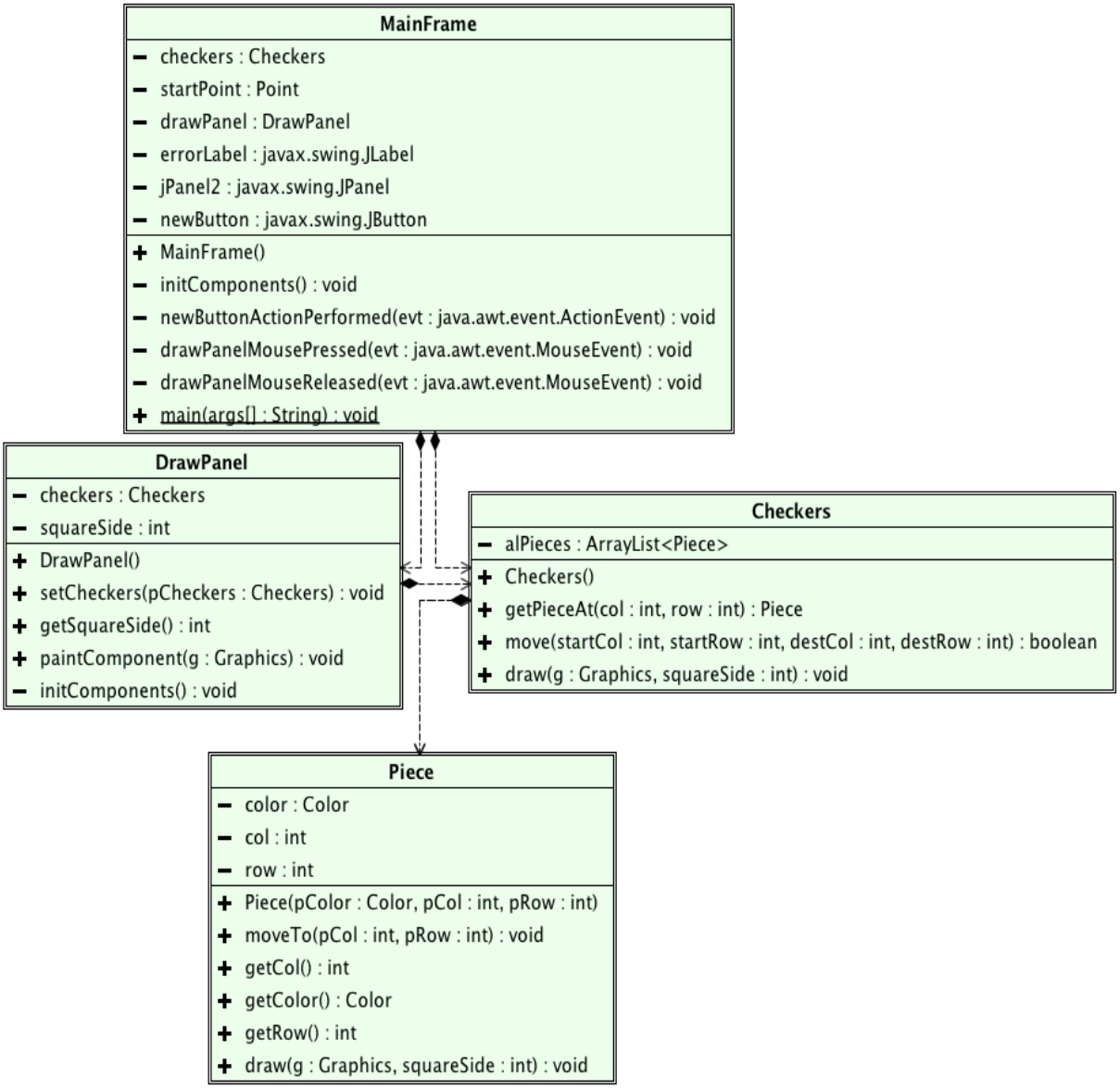
  `getSquareSide()` qui retourne le côté d'une case en pixels

Conseil :

Mémorisez les coordonnées où le bouton de la souris a été enfoncé dans un attribut **startPoint**. Lorsque le bouton est relâché, vous pouvez calculer les coordonnées des deux cases et effectuer le déplacement si possible. (Le déplacement du pion s'effectue seulement après que vous aurez relâché le bouton de la souris. Avant, vous ne voyez pas de changement.)

→ voir diagramme UML à la page suivante !





Améliorez le programme autant que vous le savez. :-)

Classes requises : Point, JPanel, Graphics, ArrayList, Color

Notions requises : calculs sur pixels, proportionnalité