Conception d'un système interactif en Java

Éléments d'interaction gestuelle

Mathieu Petit, Meriam Horchani

4 juin 2008

Résumé

À la fin du précédent TD, vous étiez amenés à une première utilisation du geste pour interagir avec le système. Ce TD est l'occasion de pousser cette utilisation plus loin dans un cas où l'interaction gestuelle peut sembler plus naturelle que l'interaction par menus classiques, à savoir celui des logiciels de dessin.

Table des matières

1 Interaction gestuelle pour logiciel de dessin

Si les menus[1] ont fait le succès des systèmes interactifs grand-public, tels que les logiciels de traitement de texte, ils n'ont pas semblé appropriés dans les cas où la tâche à réaliser nécessite une précision gestuelle non permise par la souris et difficilement appréhendable par le clavier. C'est notamment le cas des logiciels de dessin.

En effet, même si ces logiciels intègrent un certain nombre de menus, il semble plus naturel de dessiner un cercle, une spirale, un parallélogramme, que de sélectionner grâce à un menu une forme et de déterminer ensuite la position de l'objet et les dimensions de ces coordonnées. Le solution généralement choisie dans les solutions de dessin est donc de permettre à la fois une interaction par menus et boîtes de dialogue et une interaction gestuelle consistant à (re)dimensionner une forme sélectionnée (en utilisant, par exemple, le rayon pour le cercle et la diagonale pour les rectangles).

Une autre possibilité pourrait être de pouvoir dessiner directement la forme désirée sans commencer par sélectionner l'outil correspondant. L'utilisateur pourrait alors dessiner la forme globale du cercle souhaité "à main levée", forme approximative qui serait transformée en cercle parfait par la machine. Toutefois, la reconnaissance gestuelle, si elle a fait des progrès, n'est pas toujours au point et il n'est pas évident techniquement parlant de faire la distinction entre un cercle, un ovale, une patatoïde et une spirale.

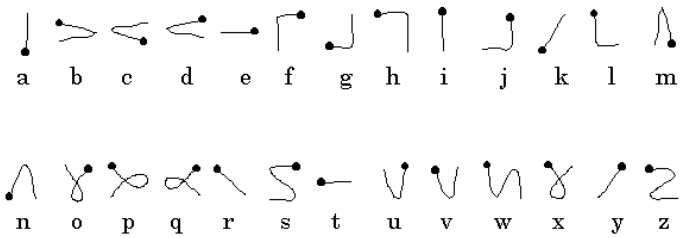

Fig. 1 – L'alphabet unistroke disponible sur les assistants personnels Palm

Une autre solution est donc d'utiliser l'interaction gestuelle pour désigner la forme désirée, avant de déterminer ses dimensions. Cette forme peut ne pas renvoyer exactement à la forme finale (par exemple, un trait horizontal pour dire à la machine qu'on souhaite dessiner un rectangle, un vertical pour un losange et un vertical puis un horizontale - pour un triangle). C'est pour cette solution que vous allez travailler dans ce TD.

Dans le cas de la reconnaissance de l'écriture manuelle, l'alphabet numérique Unistroke de Palm (Fig. 1) est un compromis entre les capacités de reconnaissance de la machine et la déformation de l'écriture naturelle auquel l'utilisateur va devoir se contraindre. Dans les années 90, le pari (réussi) de la société Palm à été de faire porter le poids de l'apprentissage de l'écriture à l'utilisateur, alors que d'autres systèmes (Newton de Apple), qui tentaient la reconnaissance à tout prix produisaient des analyses souvent erronées.

L'interaction gestuelle peut également simplifier le recours à un menu dans les cas où deux "actions" sur le système ou sur l'interface sont nécessaires pour

[1] Rappel : "menu", c'est le *M* de WIMP (cf. cours 1).

réaliser une action du point de vue de la tâche. C'est le cas pour la suppression d'une forme : il faut, classiquement, sélectionner la forme puis indiquer sa suppression (par un clic de souris, par un touche du clavier ou en passant par un menu). Il peut sembler naturel d'agir directement sur l'objet à effacer pour le supprimer, comme cela est le cas avec une gomme réelle.

Question 1 *Identifiez trois formes et trois actions sur ces formes qui vont semblent nécessaires à tout logiciel de dessin. Pour chacun de ces éléments, proposez quatre gestes qui conduiraient à leur réalisation.*

2 Objectif et principe du TD

L'objectif du TD n'est pas de concevoir intégralement un logiciel de dessin. Nous allons nous concentrer sur la création d'une forme et sur sa suppression. La forme considérée est le cercle. Si vous n'avez pas identifié la forme de cercle et l'action de suppression comme éléments nécessaires de tout logiciel de dessin, il est temps d'y songer ...

L'implémentation va se faire en Java, en utilisant entre autres la librairie "MousesGesture". Cette librairie ne permet pas d'identifier n'importe quel geste. Elle ne peut qu'identifier les séquences de direction "Haut", "Bas", "Droite" et "Gauche". Cette limite peut possiblement vous amener à modifier le geste que vous aviez l'intention d'utiliser pour identifier le fait que l'utilisateur veuille dessiner une forme plutôt qu'une autre. Par ailleurs, l'interface "MouseEventsListener" vous permet de réagir au clic de la souris et de récupérer sa position à l'écran.

Même si le TD se concentre sur la forme de cercle, les actions de création et de suppression doivent s'appliquer à n'importe quelle forme (ou du moins en donner l'impression à l'utilisateur). L'action de suppression doit donc être indépendante de la forme. Celle de création doit être pensée de façon à pouvoir être applicable à toute forme. Pour cela, nous considérons qu'un geste spécifique permet à l'utilisateur d'informer au système que la forme qu'il souhaite créer (un cercle, un rectangle ou un triangle par exemple) et d'indiquer ensuite les dimensions de cette forme (le rayon du cercle, la demi-diagonale du rectangle ou le rayon du cercle circonscrit du triangle).

Question 2 *Étant donné les contraintes de la librairie de reconnaissance de geste utilisée et les principes de création et de suppression décrits, quel gestes proposez-vous pour les actions de dessin d'un cercle et celles de sa suppression ?*

3 Implémentation demandée

Sur Moodle, récupérez et importez dans Eclipse en tant que nouveau projet, le fichier "Geste_stage1.zip".

Comme toujours, il contient une classe `Starter`, qui permet de construire un objet de type `drawingBoard` : la fenêtre de dessin (Fig. 2). Tout le code sera implémenté dans `drawingBoard.java`, au niveau de la classe interne `DrawingPane`. Le vecteur `shapes` contiendra toutes les formes que l'on dessinera à l'écran (ici un ensemble de `Circle`). Celles-ci seront disposées dans la `BufferedImage`

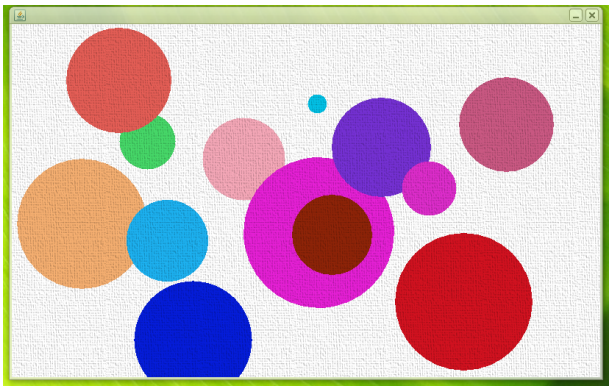

FIG. 2 – logiciel de dessin que vous réaliserez

`painting` au moment de l'appel de la méthode `paintCanvas()`. Dans un premier temps, nous initialisons le dessin de l'interface (les numéros des questions sont reportés dans le code source pour vous indiquer les endroits où il faut coder) :

1. Dans le constructeur, utilisez un appel aux méthodes statiques `load()` et `ImageIO.read()`pour charger la texture de la toile (fichier images/-toile.png),
2. avec `setPreferedSize`, agrandissez la taille du `drawingPane` à la taille de l'image `toile`,
3. dans `paintCanvas()`, initialisez l'image `painting` avec une nouvelle `BufferedImage` de la taille de `toile`, et de type `BufferedImage.TYPE_INT_RGB`,
4. récupérez dans un objet de type `Graphics2D` le contexte graphique de l'image `painting` avec la méthode `createGraphics()`,
5. dessinez dans cet objet un rectangle blanc sur toute la dimension de la `toile`,
6. ajoutez une boucle qui parcourt l'ensemble des formes de `shapes` et qui les dessine avec la couleur de votre choix sur l'objet `Graphics2D` (utilisez la méthode `fill()`),
7. dans le constructeur, ajoutez un objet `Circle` aux dimensions de votre choix dans le vecteur `shapes`,
8. enfin, dans la méthode `paintComponent(Graphics G)`, dessinez dans le contexte graphique `g2d` de l'écran l'image `painting`, puis l'image `toile`.

Exécutez votre code. Vérifiez que la toile s'affiche avec un fond blanc aux bonnes dimensions, et qu'un cercle bleu y est bien dessiné . Nous allons ajouter l'écoute de divers évènements pour que vous puissiez construire la suite de l'algorithme en autonomie :

9. à la déclaration de la classe `DrawingPane`, implémentez les interfaces `MouseGestureListener`, `MouseListener` et `MouseMotionListener`,
10. ajoutez les méthodes non implémentées, qui sont nécessaires (clic sur l'idée solution, puis "*add unimplemented methods*"),
11. dans le constructeur, ajoutez `this` à l'objet `mouseGestures` en tant que `MouseGestureListener`,

12. ajoutez des écouteurs d'évènements `Mouselistener` et `MouseMotionListener` depuis `this` sur `this`,
13. dans la méthode `mousePressed`, initialisez le `Point origin` aux coordonnées du pointeur (avec `arg0.getPoint()`), puis affichez à la console les coordonnées du `Point origin`,
14. dans la méthode `processGesture`, affichez la chaîne `arg0` à l'écran.

Exécutez le code et vérifiez que les coordonnés du début du geste, ainsi que la chaîne de segments `U, D, L, R` représentant le geste effectué s'affichent bien à l'écran. Dans le code, comparer la valeur de cette chaîne avec les chaînes gestuelles que vous avez choisies pour discriminer entre l'action "créer un cercle" et l'action "effacer une forme". Affichez les textes "créer cercle" ou "effacer forme" selon le cas.

Question 3 *Quelle étape supplémentaire est nécessaire pour que le système puisse complètement déterminer le cercle ? Comment demander cette information à l'utilisateur ?*

Ces questions vous amènent à détailler des étapes du dialogue entre l'utilisateur et le système. En théorie, il faudrait ensuite coder votre méthode de dialogue pour la création du cercle.

Comme le temps passe vite, vous partirez d'un squelette de code dans lequel le dialogue avec l'utilisateur est géré par une machine à état. Dans le code que nous proposons, la variable `mode` définit l'état courant, et le comportement, par exemple, de l'affichage graphique à l'écran est fonction de l'état. Cette approche est très commune dans l'implementation de systèmes de dialogue ou plusieurs "sens" sont associés à une même action d'interaction. Les personnes intéressées demanderons plus d'informations ... mais pour les autres :

15. chargez les sources de "Geste_stage2.zip" depuis Moodle dans Eclipse,
16. changez les valeurs des chaînes `ERASEGESTURE` et `CIRCLEGESTURE` à votre convenance,
17. complétez l'état `CIRCLE` de la méthode `paintComponent` : dessinez un cercle provisoire rouge de rayon `origin currentPoint` centré sur `origin`,
18. ajoutez la forme validée par le clic souris de l'utilisateur dans le vecteur `shapes` : initialisez `currentShape` au moment ou vous dessinez le cercle provisoire et ajoutez `currentShape` au vecteur `shapes` dans la méthode `mouseClicked()`,
19. dans la méthode `processGesture()`, codez la réaction a une demande d'effacement. On efface toujours la forme qui se trouve sous le point d'origine du geste. Une méthode de `Shape` permet de savoir si un point est inclut dans la forme,
20. complétez l'état `GESTURE` de la méthode `PaintComponent()` : on dessine en trait épais bleu la trace du geste en cours. Utilisez les valeurs des points `lastPoint` et `currentPoint` comme coordonnées de la ligne à afficher,
21. changez la coloration des cercles en construisant une couleur à partir d'un triplet RGB généré aléatoirement.

Testez votre code, et mesurez bien les actions de dialogue pour effectuer la tâche utilisateur.

Pour finir, vous exécuterez votre code sur un UMPC, ou l'on agit en entrée au moyen d'un écran tactile et d'un stylet :

22. ajoutez le code donné en annexe au constructeur de `DrawingBoard`,
23. générez une archive exécutable `.jar`, copiez là sur l'un des deux UMPC et exécutez le code.

Question 4 *Manipulez l'interface, créez des formes et effacez les. Quelle limitation technique rends le l'interaction difficile avec le logiciel quant il est exécuté sur l'UMPC ? Essayez de proposer des améliorations dans le dialogue pour contourner ces limites d'interaction. Connaissant les deux plateformes d'exécution, existe-t-il selon vous un dialogue qui soit identique sur les deux appareils et faudrait-il pour autant favoriser son adoption lors de l'implementation ?*

4 ANNEXE : Code pour le portage sur l'UMPC

```
setUndecorated(true);
setResizable(false);
GraphicsEnvironment ge=
        GraphicsEnvironment.getLocalGraphicsEnvironment();
GraphicsDevice myDevice=ge.getDefaultScreenDevice();
try {
    myDevice.setFullScreenWindow(this);
} finally {
    myDevice.setFullScreenWindow(null);
}
```