Ecrivez à la souris ! - La solution du défieur.

par Equipe DELPHI

Date de publication : 4 décembre 2006

Dernière mise à jour :

Ecrire à la souris, avec un trackBall, sur une tablette graphique surr un PC ou avec un stylet sur un ordinateur de poche, voilà qui pour bon nombre d'entre nous relevait de la science fiction il y a encore une dizaines d'année !

Quel défi ! Double puisqu'il s'agissait aussi pour ma part de démontrer que les algorithmes permettant ce genre d'interaction avec l'utilisateur étaient facilement accessible à tous.

A travers cet page et un article rédigé pour l'occasion je vous propose de découvrir la mise en oeuvre de cette technologie, qui vous ouvrira, je le souhaite, la voie vers de nombreuses réalisation telles que des corecteurs orthographique, logiciel intégrant la reconnaissance de la parole, etc...

I - Comment l'idée m'est-elle venue ?

Je venais tout juste de répondre à un fil de discussion "Utiliser un écran tactile" en envoyant les sources d'un exemple préparées spécifiquement pour ce dernier.

Un membre de l'équipe du Défi Delphi, Giovanny Temgoua, fut le seul à passer, par bonheur, sur le fil de discussion et trouva à juste titre que nous tenions là le sujet du défi n°2.

Les sources de la toute première version ont donc été accessibles à toutes et à tous pendant quelques heures, le temps que nous le déplacions dans notre coffre-fort privé. :wink:

C'est cette source qui à servi de base au défi.

Un peu plus tard, pendant le défi, un autre membre de l'équipe, Pascal Jankowski à soumis une source issue d'un fil de discussion du forum, dont je m'en suis immédiatement inspiré, cela concernait un algorithme connu sous le nom de **Distance de Levenshtein**. J'en ai aussitôt profité pour guider les participants du défi, vers cet algorithme assez simple (sur lequel je vais revenir plus en détail) à mettre en place afin de leur permettre d'avancer plus facilement dans leur développement.

Je remercie donc _cqu_ de m'avoir inspiré si brillamment.

Voici le fil de discussion concerné : [Recherche Algo] Distance levenshtein

A partir de là, j'ai pu mettre au point , une version améliorée, mettant en oeuvre la gestion d'un dictionnaire.

II - La Reconnaissance Gestuelle, qu'est-ce donc ?

C'est un procédé qui consiste à interpréter les mouvements de la souris afin de reconnaître le tracé d'une lettre (comme sur les ordinateurs de poche), ou d'un symbole afin d'executer des commandes simples (exemple de plugin d'aide à la navigation pour le navigateur Firefox : https://addons.mozilla.org/firefox/39/), ou encore de reconnaître le tracé de formes pour des applications aussi innatendues que spectaculaires (http://www.koreus.com/media/mit-sketching.html)

III - Deux programmes, deux approches ?

III-A - Le dénominateur commun

Un geste est une succession de mouvements de souris, avec ou sans levé(s) de " crayon ".

Dans mon approche et pour ne pas faire trop compliqué, j'ai choisi l'option sans levé de crayon.

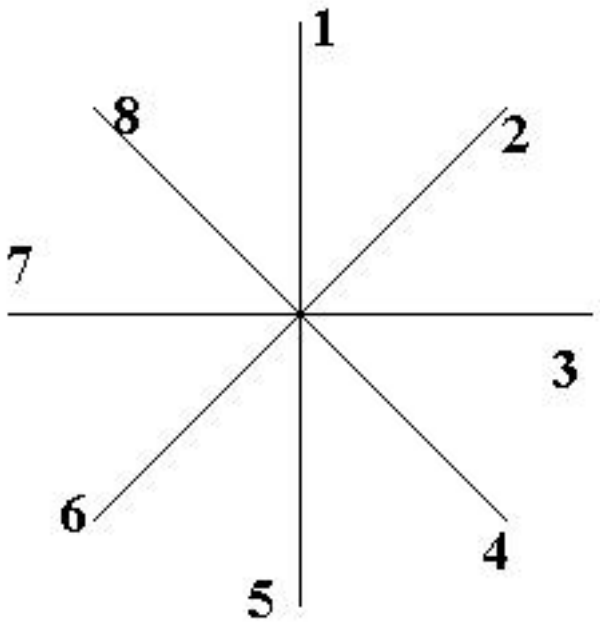
Un geste de l'utilisateur consiste donc en trois étapes :

- Poser le crayon (enfoncer le bouton gauche de la souris) => le programme récupère les coordonnées de ce premier point lors d'un évènement **OnMouseDown** d'un **TPaintBox** sur lequel s'effectue le tracé.
- Tracer la lettre (en maintenant le bouton de souris enfoncé) => le programme enregistre la succession des points dans un tableau ou une liste lors d'un événement **OnMouseMove**.
- Lever de crayon (l'utilisateur relâche le bouton de la souris) => le programme tente de reconnaître le tracé en le comparant à un dictionnaire lor d'un évènement **OnMouseUp**.

III-B - L'enregistrement du tracé

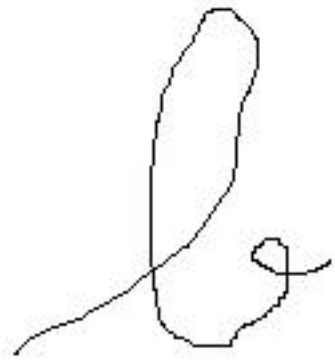
J'ai utilisé ici deux approche différentes :

- Dans un cas j'ai enregistré une succession de " directions " codées ainsi :



Ainsi, si l'utilisateur dessine un **A** (une sorte de V à l'envers), le codage dans une chaîne de caractère sera : '**24**'.

Pour un **b** minuscule manuscrit, le codage ressemblera à ceci : **218765432186543**. Ce qui inclut non seulement la grande boucle d'un **l** minuscule, mais aussi la petite boucle en fin de tracé :



Cette approche permet un stockage simple de chaque lettre ou symbole du dictionnaire, ainsi que du tracé de l'utilisateur. Par contre, ce système a ses limites, comment distinguer en effet un **p** minuscule d'imprimerie et un **D** majuscule d'imprimerie, le codage étant pour tout deux **134567** ? De même, comment distinguer un cercle, d'une spirale, symboles qui peuvent tout deux être codés **1234567812345678** ?

D'une part, cette approche impose un système d'écriture particulier à l'utilisateur qui doit s'adapter au dictionnaire du logiciel (pour distinguer le **p** du **D**), en proposant en plus un choix limité de symboles à reconnaître (pour éviter un conflit entre la reconnaissance d'un cercle et d'une spirale).

Stockage dans le dictionnaire : chaque enregistrement du dictionnaire type **TGeste** est simplement stocké dans un tableau nommé **Gestes**.

- Dans le programme plus évolué, c'est la succession des points (en rouge sur le dessin) qui est enregistrée dans une liste de points (une classe encapsulant **Tlist** a été écrite à cet effet : classe **TGeste** dans **UnitGeste.pas**)

Cette approche nécessite un système de stockage du dictionnaire de références plus complexe, néanmoins

elle a l'énorme avantage de permettre au logiciel de s'adapter au besoin de l'utilisateur, pour peu qu'elle offre un système d'apprentissage.

De plus, avec cette approche, la différenciation entre un **p** et un **D**, ou un cercle et une spirale devient possible.

Stockage dans le dictionnaire : Chaque enregistrement de type **Tgestestocké** du Dictionnaire ou de la base de données des exemples est stockée dans une classe dérivée de **TobjectList** (classe **Tdictionnaire** dans **UnitDico.pas**).

IV - La reconnaissance gestuelle elle même

IV-A - L'algorithme principal

Le point commun entre les deux programmes est l'algorithme suivant :

```
Initialiser Note ;
Caractère_candidat :=aucun ;
POUR toutes les Entrées du Dictionnaire FAIRE
        NouvelleNote :=Comparer(Geste de l'utilisateur, Une Entrée du dictionnaire) ;
        SI NouvelleNote est meilleure que Note ALORS
                Note :=NouvelleNote ;
                Caractère_candidat :=Une Entrée du dictionnaire ;
        FIN SI
FIN POUR
```

Dans les deux programmes, cet algorithme est traduit par l'implémentation de la méthode **TSourisTexte.Reconnaitre** (dans **Unit2.pas** pour le 1er programme, et dans **UnitReconnaissance.pas** pour le 2nd).

La différence entre les deux implémentations tient dans la façon dont la comparaison est effectuée.

IV-B - La fonction de comparaison

Dans le premier programme, il s'agit d'un algorithme de filtrage maison : **plus la note attribuée sera élevée** plus le geste de l'utilisateur sera suceptible d'être le candidat.

Dans le deuxième cas, le programme calcule une **Distance de Levenshtein**. Cette dernière, sur laquelle je reviens plus loin, revient à calculer la somme des distances entre chaque point d'un geste de référence et chaque point du geste tracé. **Plus cette somme sera petite**, plus le tracé de l'utilisateur sera " proche " du geste de référence.

IV-C - La " Normalisation " d'un tracé

Il faut noter encore une différence notable entre les deux implémentations.

Dans le premier programme, seul une succession de " directions " est prise en compte, la taille du tracé n'influe donc que très peu.

Dans le deuxième programme, en revanche, il faut pouvoir comparer une lettre tracée plus petite ou plus grande que celle contenue dans le dictionnaire. Pour ce faire, à chaque fois qu'un geste sera tracé, il sera nécessaire de prendre les limites du rectangle dans lequel est contenu le symbole, et de le rapporter à un rectangle d'une taille prédéfinie.

Dans le dictionnaire du deuxième programme, tous les tracés gestes (caractères, symboles) de référence tiennent dans un rectangle de 1000 par 1000 : le point du tracé le plus à gauche aura pour coordonnées X=0, le point le plus en Haut : Y=0, le point le plus à droite : X=1000 et le point le plus en bas Y=1000.

Admettons que l'utilisateur ait tracé une lettre (b par exemple). Grâce aux différents points de ce tracé, nous pouvons déterminer, les limites dans lequel celui-ci à été effectué dans la PaintBox et nous obtenons : XMin=50, YMin=25, XMax=150, YMax=250.

Avant la reconnaissance de caractère, pour pouvoir comparer le tracé de l'utilisateur à celui du dictionnaire, il nous faudra effectuer la transformation suivante, pour chaque point tracé par l'utilisateur :

- x'=(x-XMin)*1000/(XMax-XMin)
- y'=(y-YMin)*1000/(YMax-YMin)

Le nouvel ensemble de points ainsi obtenu, pourra correctement être comparé au geste du dictionnaire, un peu comme si nous superposions deux calques l'un sur l'autre.

V - La Distance de Levenshtein

Voir tutoriel : **La Distance de Levenshtein : Applications en Delphi**. (en cours d'écriture...)

VI - Conclusion

Encore un défi plein de rebondissements, qui je l'espère vous aura ouvert les portes vers de nombreux autres types d'applications utilisant la **Distance de Levenshtein** (correcteur orthographique, moteur de recherche) ou ses dérivés comme la **déformation dynamique temporelle** ou **Dynamic Time Warping (DTW)** que nous avons ici utilisée ici (Reconnaissance gestuelle, reconnaissance de la parole, reconnaissance de formes, biométrie : empreintes digitales,...).

Sur ce, mes concurrents ont été brillants, comme d'habitude. Je ne peux qu'applaudir encore une fois les trésors d'ingéniosité qu'ils ont déployés et la ténacité dont ils ont fait preuve.

Etonnemment, la partie de développement qui m'a été la plus difficile à mettre en #uvre à été le codage des méthodes **Reconnaître** et **ReduireGeste** du premier programme : n'ayant à ce moment aucune connaissance d'aucun algorithme, sa mise au point à été complètement empirique.