Ecrivez à la souris ! - La solution du vainqueur : tiki06.

par Equipe DELPHI

Date de publication : 4 décembre 2006

Dernière mise à jour :

tiki06 (Thierry Laborde) nous propose de découvrir la solution qu'il a envisagé pour résoudre ce défi.

I - Introduction
II - Déroulement
    II-A - Idée initiale
    II-B - Distance de Levensthein et reconnaissance de caractères
    II-C - Unité UgestionOCR
    II-D - Unité UGestionDico
    II-E - Unité UGestionSkin
    II-F - Unité UMain
        II-F-1 - Problème majuscules/minuscules
        II-F-2 - Principe de fonctionement
III - Autres fonctionalités
IV - Conclusion
V - Téléchargement
    V-A - Téléchargement des sources
    V-B - Téléchargement de l'executable

I - Introduction

Par manque de temps je n'avais pas pu participer au premier défi, alors quand le second s'est présenté j'ai absolument voulu le relever. Pourtant au début, à la lecture du défi, j'ai cru que cela serait super difficile et que je n'y arriverai pas. Mais waskol précisant qu'il y avait une solution simple, je me suis lancé dans l'aventure.

II - Déroulement

II-A - Idée initiale

Au début, je suis parti sur une idée où l'utilisateur ne pouvait dessiner que des droites sur la zone de dessin. Je pensais ainsi que cela me faciliterait le travail et que j'aurais du coup une reconnaissance à 100%. Mais cela m'obligeait à prendre en compte beaucoup de cas et de directions pour pouvoir reconnaitre tous les caractères. Puis waskol nous a aidé en nous fournissant quelques informations :

http://delphi.developpez.com/defi/ecriture/topic/#p121

Du coup j'ai découvert le principe de la Distance de Levensthein que je ne connaissais pas du tout.

II-B - Distance de Levensthein et reconnaissance de caractères

Je ne rentrerai pas dans le détail de la distance de Levensthein car vous trouverez toutes les informations sur le post ci-dessus de waskol.

En bref l'utilisateur desssine en allant dans 8 directions possible :

A droite --> 1

En haut à droite --> 2

En haut --> 3

En Haut à gauche --> 4

A gauche --> 5

En bas à gauche --> 6

En bas --> 7

En bas à droite --> 8

Entre 2 mouvements de souris une droite est formée entre 2 points. On détermine l'angle formé par cette droite que l'on met dans une chaine de caractères. Ceci nous donne donc pour l'ensemble du tracé (le tracé débute quand l'utilisateur appuie sur le bouton de la souris puis dessine et se termine quand il relâche le bouton de la souris) quelque chose sous la forme suivante : **"878181232223"** (Car notre tracé n'est jamais parfait et il est difficile de faire des droites impeccables avec la souris).

Si on détermine ensuite un dictionnaire représentant les différents tracés de chaque caractère, on aurait par exemple pour un **V** : **"82"**, soit un tracé en bas à droite puis en haut à droite.

Ensuite la distance de Levensthein nous permet de déterminer quel est le caractère du dictionnaire qui se rapproche le plus de notre tracé en comparant les 2 chaines de caractères.

II-C - Unité UgestionOCR

J'ai donc modifié mon programme afin d'utiliser ce principe. Pour cela, j'ai découpé le programme (afin de le rendre plus clair et plus lisible) en différentes unités.

Pour la reconnaissance des caractères, j'ai donc créé l'unité : UGestionOCR.

Celle-ci comporte différentes fonctions. La première (CalculAngle) permet de déterminer l'angle d'une droite entre 2 points sur le canvas.

Ceci permet de déterminer l'angle que prend le tracé de l'utilisateur :

fonction CalculAngle

```
{*-------------------------------------------------------------------------------
  Fonction pour calculer un angle entre 2 points sur un canvas

  @param    X1 : Coordonnées X du 1er point
  @param    Y1 : Coordonnées Y du 1er point
  @param    X2 : Coordonnées X du 2ème point
  @param    Y2 : Coordonnées Y du 2ème point
  @Return   Renvoi l'angle en degré
-------------------------------------------------------------------------------}
function CalculAngle(x1,y1,x2,y2:double):double;
var
  a,ang,x,y:double;
begin
  Ang:=0;
  x:=x2-x1;
  y:=y1-y2;
  if x=0 then
  begin
   if y>0 then
     ang:=Pi/2
   else
     if y<0 then
       ang:=3*Pi/2
     else
       ang:=0;
  end
  else
  begin
    a:=arctan(y/x);
    if x>0 then
      if y>=0 then
        ang:=a
      else
        ang:=a+2*Pi;
    if x<0 then ang:=a+pi;
  end;
  Ang:=RadToDeg(Ang);
  result:=ang;
end;
```

Ensuite la fonction DistanceEntreCourbe permet de calculer la distance entre 2 directions :

fonction DistanceEntreCourbe

```
{*-------------------------------------------------------------------------------
  Fonction pour calculer la distance entre 2 points de 2 Courbes

  @param    Courbe1  String de la 1ère courbe
  @param    Courbe2  String de la 2ème courbe
  @Param    i        Numéro du point de la 1ère Courbe
  @Param    j        Numéro du point de la 2ème Courbe
  @return   Renvoi la distance entre les 2 points des 2 courbes
-------------------------------------------------------------------------------}
function DistanceEntreCourbe(Courbe1,Courbe2:string;i,j:integer):integer;
var
  a,b,c:integer;
begin
  a:=StrToInt(Courbe1[i]);
  b:=StrToInt(Courbe2[j]);
  c:=abs(a-b);
  if c>4 then c:=8-c;
  result:=c;
```

fonction DistanceEntreCourbe
```
  end;
```

Puis la dernière fonction permet de calculer la distance de Levensthein entre 2 tracés :

fonction DistanceDeLevensthein
```
  {*-------------------------------------------------------------------------------
    Fonction pour calculer de Levenshtein entre 2 courbes

    @param    Courbe1  String de la 1ère courbe
    @param    Courbe2  String de la 2ème courbe
    @return   Renvoi la distance de Levenshtein entre les 2 courbes
  -------------------------------------------------------------------------------}
  function DistanceDeLevenshtein(Courbe1,Courbe2:String):integer;
  var
    d  : array of array of integer;
    DTW: array of array of integer;
    i,j,n,m,cout : integer;
  begin
    n:=Length(Courbe1);
    m:=Length(Courbe2);
    SetLength(d,n+1,m+1);
    for i:=1 to n do
      for j:=1 to m do
        d[i,j]:=DistanceEntreCourbe(Courbe1,Courbe2,i,j);
    SetLength(DTW,n+1,m+1);
    for i:=1 to m do
      DTW[0,i]:=100000;
    for i:=1 to n do
      DTW[i,0]:=100000;
    DTW[0,0]:=0;
    for i:=1 to n do
    begin
      for j:=1 to m do
      begin
        cout:=d[i,j];
        DTW[i,j]:=Min(Min(DTW[i-1,j]+cout, DTW[i,j-1]+cout),DTW[i-1,j-1]+cout);
      end;
    end;
    result:=DTW[n,m];
  end;
```

II-D - Unité UGestionDico

J'ai ensuité créé l'unité UGestionDico qui va me permettre de gérer mon dictionnaire. Pour cela j'ai créé dans l'unité un Type **TDico** contenant 2 chaînes de caractères : Le caractère concerné et le tracé de ce caractère :

Type TDico
```
  type
  TDico = class  // Class contenant les lettres et le dessin correspondant
    Lettre:String;
    Dessin:String;
  end;
```

Puis une variable **ListeDico** de type Tlist contenant toutes les lignes du dictionnaire.

Ensuite, j'ai mis une fonction pour charger le dictionnaire qui est un simple fichier texte contenant sur chaque ligne le caractère et le tracé :

Fonction ChargeDico
```
  {*-------------------------------------------------------------------------------
    Fonction pour charger le fichier dictionnaire

    @return   Renvoi True si le chargement c'est bien passé, sino False
  -------------------------------------------------------------------------------}
  function ChargeDico: Boolean;
```

Fonction ChargeDico

```
var
  FichierDico:textfile;
  Ligne:String;
  Fichier:String;
  LigneDico:Tdico;
  iPos:integer;
begin
  Result:=True;
  Fichier:=Application.Title+'.dico';
  if FileExists(Fichier) and (ListeDico<>nil) then
  begin
    ListeDico.Clear;
    assignfile(FichierDico,Fichier);
    Try
      reset(FichierDico);
      while not eof(FichierDico) do
      begin
        readln(FichierDico,Ligne);
        if pos('=',Ligne)<>0 then
        begin
          LigneDico:=TDico.Create;
          iPos := Pos('=',Ligne);
          LigneDico.Lettre := LeftStr(Ligne,iPos - 1);
          LigneDico.Dessin := RightStr(Ligne,Length(Ligne) - iPos);
          ListeDico.Add(LigneDico);
        end;
      end;
    except
      Result:=False;
    end;
    closefile(FichierDico);
  end
  else
  begin
    Result:=False;
    Showmessage('Problème lors du chargement du dictionnaire, vérifez votre installation !!!');
  end;
end;
```

II-E - Unité UGestionSkin

En développement, un de mes chevaux de bataille est l'ergonomie, et j'essaie toujours de rendre mon application agréable et simple à utiliser. J'ai donc voulu faire de même pour ce programme. Pour cela j'ai décidé de skinner l'application et de lui donner l'apparence d'un PDA. J'ai donc créé l'unité UGestionSkin qui permet de skinner une fenêtre à partir d'un bitmap. Elle ne contient qu'une seule fonction (SkinFenetre) qui a été faite à partir d'exemples de sources trouvé sur le forum et sur le net :

Fonction SkinFenetre

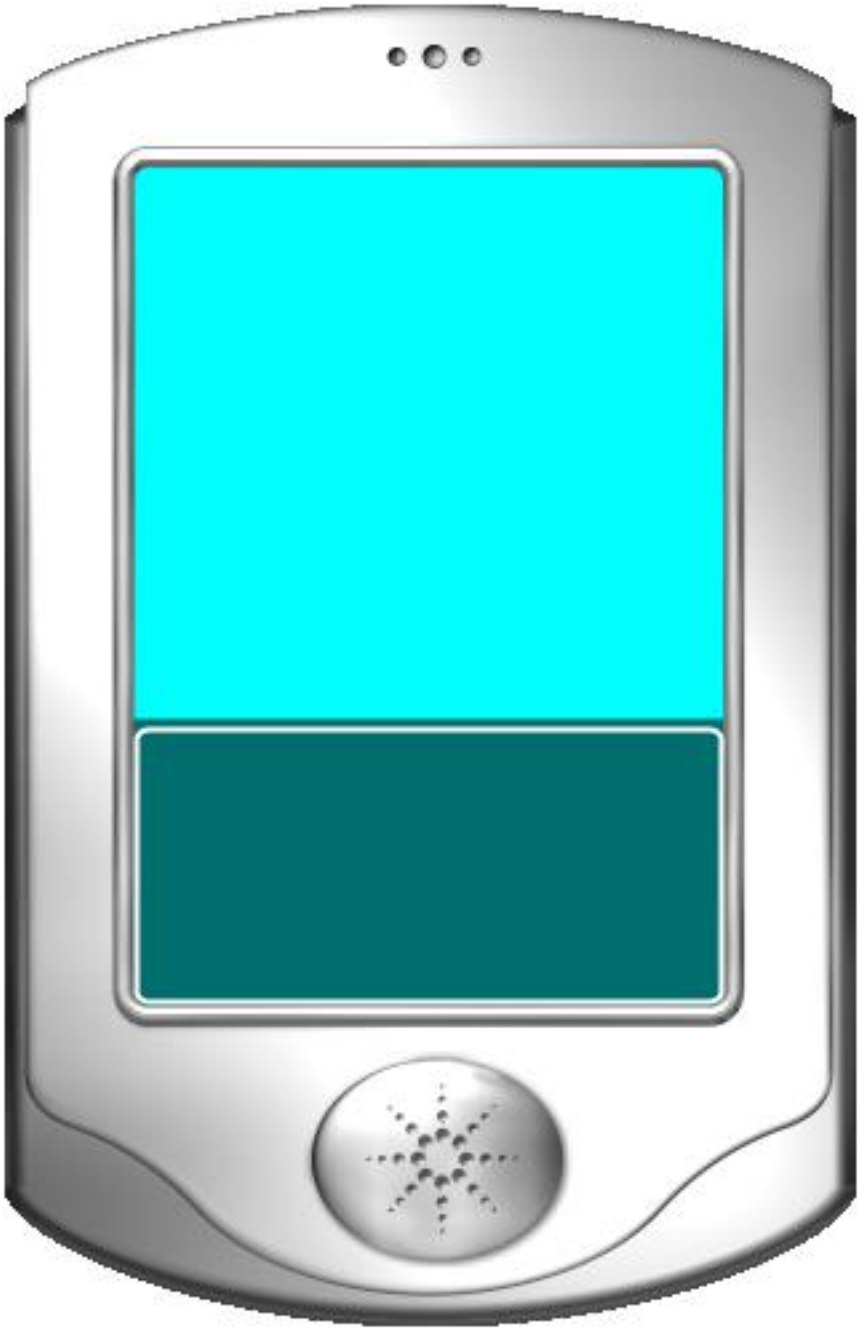
```
{*-------------------------------------------------------------------------------
  Procedure pour skiner une fenêtre à partir d'une image Bitmap

  @param    ImageBmp : Image de la skin
-------------------------------------------------------------------------------}
procedure SkinFenetre(ImageBmp:Tbitmap);
var
  x,y,reg,regtemp,debut,fin: integer;
  etaitblanc,first: boolean;
begin
  Application.MainForm.height := ImageBmp.height;   //on adapte les
  Application.MainForm.width  := ImageBmp.width;    //dimensions de la fiche
  first:=true;
  Reg:=0;
  for y:=1 to ImageBmp.height do
  begin
    debut:=1;
    etaitblanc:=true;
    for x:=1 to ImageBmp.width do
    begin
      if ImageBmp.Canvas.Pixels[x,y]=ImageBmp.Canvas.Pixels[0,0] then //Vérifie si le pixel est de
la couleur transparente
      begin
        if etaitblanc=false then  //et que le précédent ne l'etait pas
        begin
```

Fonction SkinFenetre

```
            fin:=x-1;   //le dernier pixel non transparent etait le precedent
            if first=true then //si c'est la 1°region qu'on crée:
            begin
              reg:=CreateRectRgn(debut,y,fin+1,y+1); //on la crée
              first:=false;  //la prochaine ne sera plus la premiere
            end
            else
            begin//si c'est pas la premiere region
              regtemp:=createrectrgn(debut,y,fin+1,y+1);//on en crée une temporaire
              CombineRgn(reg, reg, regtemp, rgn_or);//on l'ajoute à reg qui sera la region finale avec
  rgn_or (voir autres possibilités dans l'aide)
              deleteobject(regtemp);//on supprime la region temporaire pour rester propres (si on ne
  le fait pas, ca se fait autom. qd on ferme l'application)
            end;
          end;
          etaitblanc:=true;
        end
        else      //le pixel n'est pas transparent
        begin
          if etaitblanc=true then debut:=x;//ben oui rien que ca
          etaitblanc:=false;
          if x=ImageBmp.width-1 then //on arrive au dernier point de la ligne
            if first=true then //si c'est la 1°region qu'on crée:
            begin
              reg:=CreateRectRgn(debut,y,x,y+1); //on la crée
              first:=false;  //la prochaine ne sera plus la premiere
            end
            else
            begin//si c'est pas la premiere region
              regtemp:=createrectrgn(debut,y,x,y+1);//on en crée une temporaire
              CombineRgn(reg, reg, regtemp, rgn_or);//on l'ajoute à reg qui sera la region finale avec
  rgn_or (voir autres possibilités dans l'aide)
              deleteobject(regtemp);//on supprime la region temporaire pour rester propres (si on ne
  le fait pas, ca se fait autom. qd on ferme l'application)
            end;
        end;
      end;
    end;
    SetWindowRgn(Application.MainForm.handle, reg,true);//on applique la region
    application.ProcessMessages;//on le laisse un peu souffler (pas obligatoire, c'est juste par
  compassion pour le processeur)
  end;
```

La fonction part du principe que la couleur transparente du bitmap est le pixel en haut à gauche (X=0,Y=0). Ensuite j'ai créé un bitmap correspondant à mon PDA pour le fond de ma fenêtre :



II-F - Unité UMain

II-F-1 - Problème majuscules/minuscules

Je me suis posé la question de savoir comment gérer les majuscules et les minuscules. J'aurais pu rajouter au dictionnaire les tracés des lettres majuscules et minuscules. Mais je trouvais que cela augmentait le risque d'erreur lors de la reconnaissance et demandait d'avoir beaucoup plus de tracés différents. J'ai voulu également inclure à un moment un bouton permettant de passer de majuscule en minuscule et vice-versa. Mas cela rendait moins pratique l'utilisation du programme.

J'ai donc finalement opté pour une solution ou j'ai mis 2 zones de dessin sur ma fenêtre. De cette manière si

l'utilisateur dessine sur la zone de gauche la lettre sera forcément une majuscule et sur la partie de droite ce sera forcément une minuscule. Je trouve cette solution plus rapide pour l'utilisateur et de plus cela lui permet d'avoir toujours à faire les mêmes tracés pour les majuscules et minuscules. Bien entendu les chiffres et commandes peuvent être dessinés sur n'importe quelle zone.

II-F-2 - Principe de fonctionement

J'ai donc mis sur ma fenêtre 2 PaintBox qui vont être mes zones de dessin. Puis j'ai rajouté un mémo qui va contenir le texte représentant mes tracés. Ensuite, j'ai simplement rajouté 2 boutons pour pouvoir sortir de l'application et la réduire (car vu que ma fenêtre est skinnée, je n'ai pas les boutons par défaut). A l'initialisation, je charge mon bitmap (pour pouvoir skinner ma fenêtre, j'initalise mes couleurs, je charge un curseur personalisé pour la zone de dessin. Puis je créé mon dico et je le charge, et j'initalise également mes 8 directions avec les angles correspondants :

```
{*-------------------------------------------------------------------------------
  Procédure d'initialisation des paramètres
-------------------------------------------------------------------------------}
Procedure TFrmMain.Initialisation;
begin
  Try
    ImgFond.Picture.LoadFromFile('Fond.bmp');
  except
    Showmessage('Problème lors du chargement de l''image de fond, vérifez votre installation !!!');
    Application.terminate;
    exit;
  end;
  DoubleBuffered:=True;
  CaractereDessin:='';
  PbDessinMaj.Canvas.Pen.Width:=2;
  PbDessinMin.Canvas.Pen.Width:=2;
  FrmMain.Color:=COLOR_PAINTBOX;
  Screen.Cursors[CURSEUR_STYLO] := LoadCursor(HInstance, 'STYLO');
  PbDessinMaj.Cursor := CURSEUR_STYLO;
  PbDessinMin.Cursor := CURSEUR_STYLO;
  ListeDico:=TList.Create;
  // définitions des angles de références
  AngleRef[1]:=90;
  AngleRef[2]:=45;
  AngleRef[3]:=0;
  AngleRef[4]:=315;
  AngleRef[5]:=270;
  AngleRef[6]:=225;
  AngleRef[7]:=180;
  AngleRef[8]:=135;
  if Not(ChargeDico) then Application.Terminate;
end;
```

Ennsuite lorsque l'utilisateur clique sur une des zones de dessin j'initialise mon tracé à la position de la souris et je masque l'autre zone de dessin qui ne sert pas pour le moment :

```
{*-------------------------------------------------------------------------------
  Procédure déclenchée lors de l'appui sur le bouton de la souris sur la zone
  de dessin pour initialiser la position de départ du dessin
-------------------------------------------------------------------------------}
procedure TFrmMain.PbDessinMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  (Sender As TPaintBox).Canvas.MoveTo(X,Y);
  // Si on commence un dessin sur une des PaintBox on cache l'autre tant qu'on
  // a pas fini le dessin
  if (Sender As TPaintBox).Name='PbDessinMin' then
    PbDessinMaj.Visible:=False
  else
    PbDessinMin.Visible:=False;
end;
```

Puis lorsque l'utilisateur fait son tracé je calcule régulièrement la direction du tracé pour construire ma chaine de caractère correspondant au tracé. Pour cela je récupère l'angle en dégré du tracé de l'utilisateur puis je vérifie quel est l'angle de mes 8 directions le plus proche de celui qui est tracé. J'ajoute ensuite à ma chaine représentant le tracé la direction prise par le tracé de l'utilisateur. (Pour éviter d'avoir trop d'erreurs de reconnaissance, je fais cette vérification que tous les 10 pixels de déplacement par rapport à la dernière position de la souris) :

```
{*-----------------------------------------------------------------------------
  Procédure déclenchée lors du dessin du caractère pour enregistrer les angles
  et dessiner le caractère
-----------------------------------------------------------------------------}
procedure TFrmMain.PbDessinMouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
var
  i,index:integer;
  Distance,Ang:extended;
begin
  if Shift=[ssLeft] then
  begin
    // On ne refait le calcul que tous les 10 pixels de déplacement pour éviter
    // d'avoir trop d'erreurs de reconnaissance
    if (Abs(X-(Sender As TPaintBox).Canvas.PenPos.X)>10) or (Abs(Y-(Sender As
TPaintBox).Canvas.PenPos.Y)>10) then
    begin
      Index:=0;
      Ang:=CalculAngle((Sender As TPaintBox).Canvas.PenPos.X,(Sender As
TPaintBox).Canvas.PenPos.Y,X,Y);
      (Sender As TPaintBox).Canvas.LineTo(X,Y);
      Distance:=100000;
      for i:=1 to High(AngleRef) do
      begin
        if (Distance>sqrt(sqr(Ang-AngleRef[i]))) then
        begin
          Distance:=sqrt(sqr(Ang-AngleRef[i]));
          Index:=i;
        end;
      end;
      if (Index<>0) and (rightStr(CaractereDessin,1)<>inttostr(Index)) then
        CaractereDessin:=CaractereDessin+inttostr(Index);
    end;
  end;
end;
```

Ensuite lorsque l'utilisateur relâche la souris, je lance la vérification avec la distance de Levensthein pour trouver le caractère correspondant à son tracé. Puis j'affiche le caractère reconnu (ou je demande confirmation de sortie de l'application au cas ou l'utilisateur aurait tracé la forme de sortie), j'efface la zone de tracé et je réaffiche la zone de dessin masqué.

```
{*-----------------------------------------------------------------------------
  Procédure déclenchée lors du relachement du bouton de la souris sur la zone de
  dessin pour lancer la reconnaissance du carctère
-----------------------------------------------------------------------------}
procedure TFrmMain.PbDessinMouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
  i,index:integer;
  Distance:extended;
begin
  Index:=-1;
  Distance:=100000;
  for i:=0 to ListeDico.Count-1 do
  begin
    if (Distance>DistanceDeLevenshtein(CaractereDessin,Tdico(ListeDico[i]).Dessin)) then
    begin
      Distance:=DistanceDeLevenshtein(CaractereDessin,Tdico(ListeDico[i]).Dessin);
      Index:=i;
    end;
  end;
  if index<>-1 then
  begin
    if Length(Tdico(ListeDico[Index]).Lettre)>1 then
    begin
      if strtoint(Tdico(ListeDico[Index]).Lettre)=99 then  // cas de la commande pour sortir de
l'application
        ImgFermer.OnClick(Sender)
```

```
        else
          keybd_event(strtoint(Tdico(ListeDico[Index]).Lettre),0,WM_KEYDOWN,0);
      end
      else
      begin
        if (Sender As TPaintBox).Name='PbDessinMaj' then
          MmText.Text:=MmText.Text+UpperCase(Tdico(ListeDico[Index]).Lettre)
        else
          MmText.Text:=MmText.Text+LowerCase(Tdico(ListeDico[Index]).Lettre);
      end;
      MmText.SelStart:=Length(MmText.Text);
    end;
    CaractereDessin:='';
    (Sender As TPaintBox).Canvas.FillRect((Sender As TPaintBox).Canvas.ClipRect);
    // le dessin est fini on réaffiche les 2 PaintBox
    PbDessinMaj.Visible:=True;
    PbDessinMin.Visible:=True;
  end;
```

Puis le cycle peut recommencer pour un autre tracé et une autre reconnaissance.

III - Autres fonctionalités

Si j'avais eu le temps voici d'autres fonctionalités que j'aurais bien aimé rajouter au programme (certains des autres participants au défi en ont inclues quelques-unes) :

- Ajout de la possibilité d'inclure des nouvelles lettres et/ou tracés au dictionnaire.
- Ajout de fonction par dessin. Exemple : Copier, coller....etc.
- Possibilité de déssiner des mots entiers et pas lettre par lettre.
- Pouvoir dessiner n'importe où sur le bureau Windows pour lancer des fonctions.
- ...etc

IV - Conclusion

Hormis l'aspect technique, j'ai beaucoup appris de ce défi. Mais je crois qu'avant tout ce que ce défi m'a montré c'est que d'un problème posé qui pouvait paraître compliqué au départ (la reconnaissance de caractères) il a été possible de le résoudre simplement. Bien entendu, nous ne sommes pas allés aussi loin que le font les logiciels professionnels d'OCR. Mais cela nous permet de se souvenir d'un des principes de base du développement : rendre simple quelque chose qui à la base paraît compliqué. Et cela a été possible grace aux idées que nous à fournies waskol, mais également aux autres idées, questions, réponses des autres forumeurs. Et c'est justement dans ces moments là que prend toute la valeur de la communauté **developpez.com**, quand nous ne voyons pas de solution, quand on se retrouve bloqué sur un point...etc. Car nous sommes toujours plus performants en partageant et en se faisant aider des autres que tout seul dans notre coin. Voila ce qui à mon avis est le point le plus important de tous ces défis Delphi.

V - Téléchargement

V-A - Téléchargement des sources

V-B - Téléchargement de l'executable