TD Microsoft Surface SurfacePaint

Objectif

Afin de se familiariser avec les différents widgets présents sur la table Microsoft Surface, nous allons créer une application pas à pas nous permettant de dessiner sur la table. Le projet utilisé dans ce TP est librement inspiré d'un projet étudiant effectué en 5ème année en 2010-2011.

Thème : Développement d'une application de dessin collaboratif sur la table Surface.

Objectifs : Mettre en pratique des widgets présents sur la table Microsoft Surface - ***ScatterView***, ***TagVisualizer***, ***SurfaceInkCanvas***, ***SurfaceMenu***, ***LibraryContainer***

I. Installation et configuration de l'environnement de développement

Préparation de l'environnement

Afin de développer pour Microsoft Surface, il vous faudra les éléments suivants :

- Un PC ou une machine virtuelle sous Windows 7
- Microsoft Visual Studio 2010 (version Pro, Ultimate ou C# Express) (à télécharger sur le portail MSDN Polytech)
- .Net Framework 4.0 (http://www.microsoft.com/en-us/download/details.aspx?id=17718)
- XNA Framework 4.0 (http://www.microsoft.com/en-us/download/details.aspx?id=20914)
- Surface SDK 2.0 (http://www.microsoft.com/en-us/download/details.aspx?id=26716)

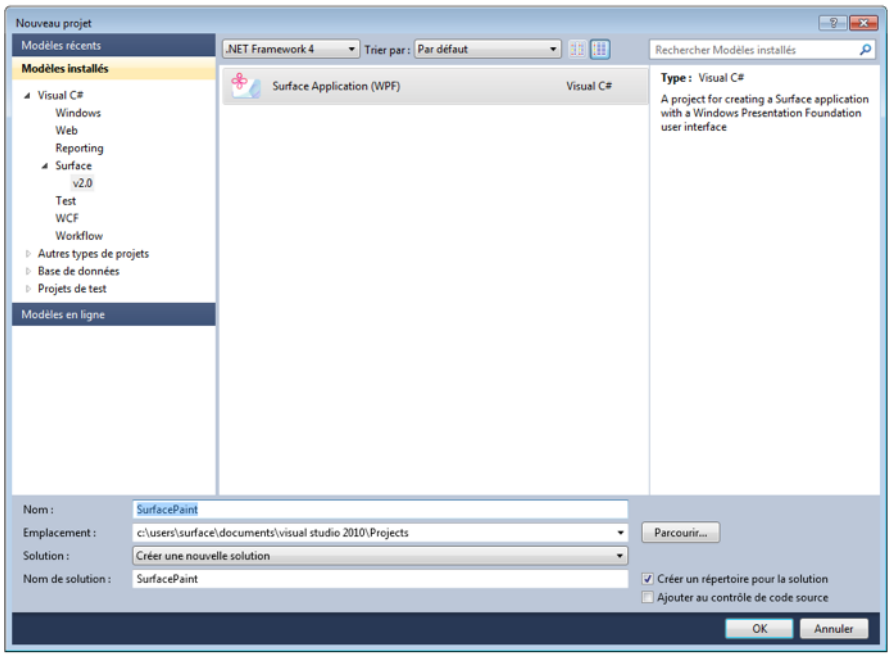
Utilisation de la machine virtuelle

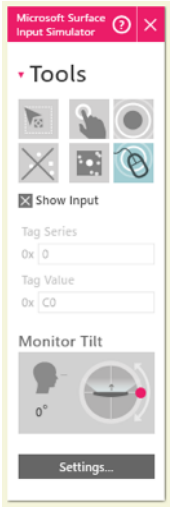
Afin de faciliter le processus de configuration de l'environnement, une machine virtuelle contenant l'ensemble des logiciels nécessaires au développement Surface vous est fournie. Pour pouvoir l'utiliser, installez VirtualBox (https://www.virtualbox.org/wiki/Downloads).

II. Prise de contact

1ère étape : Création d'un projet Surface

Ouvrez Visual Studio et créez un nouveau projet Surface que vous appellerez SurfacePaint.
Menu Fichier -> Nouveau Projet -> Visual C# -> Surface v2.0 -> Surface Application (WPF)





Les éléments importants de la structure du projet ainsi créé sont les suivants :

- Un dossier « Resources » dans lequel vous allez mettre toutes les ressources de votre projet (images, sons …)
- Un fichier App.xaml et le fichier App.xaml.cs associé qui définissent le point d'entrée de l'application. Ces fichiers ne seront jamais modifiés.
- Un fichier SurfacePaint.xml qui est le fichier de description permettant de rajouter l'application dans le launcher de la table.
- Un fichier SurfaceWindow1.xaml et le fichier SurfaceWindow1.xaml.cs associé qui sont les fichiers que nous allons modifier tout au long de ce TP. Ces fichiers représentent l'écran principal de notre application.

Afin de vérifier que notre application nouvellement créée fonctionne correctement, lancer votre application grâce à la flèche verte. Ceci lancera le simulateur.

Afin de simuler une environnement multitouch sur ordinateur, lancer le Surface InputSimulator.

2ème étape : Familiarisation avec le fichier principal de notre application

Le fichier SurfaceWindow1.xaml représente la fenêtre principale de notre application. Celle-ci est la première chose visible de notre application une fois lancée. Afin de vous familiariser avec le contenu de ce fichier, ajoutez une couleur au Background de la ***Grid*** principale grâce à la propriété Background.

III. Les outils du peintre

Maintenant que nous avons mis en place le squelette de notre application, nous allons ajouter une palette qui apparaîtra lors du dépôt sur la table du Tag correspondant. Un tel composant se nomme ***TagVisualizer***, et permet d'afficher une ***TagVisualization*** (un morceau d'interface) à la détection d'un tag sur la table.

1ère étape : Ajout d'une palette de couleur

- Dans la ***Grid*** principale, ajoutez un ***TagVisualizer*** qui réagira au ByteTag ayant pour code 0xC0. Pour cela, ajoutez un élément <s:TagVisualizationDefinition/> correctement configuré au sein de la définition des tags du ***TagVisualizer***.
- Liez ce ***TagVisualizer*** à la ***TagVisualization*** fournie avec le TD (fichiers ColorPalette.xaml et ColorPalette.xaml.cs) afin que la palette apparaisse à la détection du tag. Pour ce faire, utilisez la propriété Source de la TagVisualizationDefinition.
- Lorsque vous poserez le tag 0xC0 sur la table, la palette ci-dessus devra apparaître.

2ème étape : Ajout d'une feuille de dessin

Afin de préparer l'ajout de différents éléments à notre application comme un menu, nous allons tout d'abord ajouté une ***ScatterView*** afin de pouvoir bénéficier de ses propriétés particulières qui sont le fait de pouvoir redimensionner, déplacer et tourner librement ses éléments.

- Ajoutez une ***ScatterView*** au sein du ***TagVisualizer***. Vous pouvez nommer un élément grâce à la propriété x:Name.
- Ajoutez un premier ***ScatterViewItem*** au sein de celle-ci destiné à être la feuille de dessin. Pour cela, ajoutez les bonnes propriétés à cet item pour qu'il soit en dessous de tous les autres, sans aucune orientation, ayant une largeur et une longueur de telle sorte à prendre la place de tout l'écran, non sélectionnable, que l'on ne peut pas bouger ni redimensionner…
- Dans ce ***ScatterViewItem***, ajoutez un ***SurfaceInkCanvas*** avec un fond transparent.
- Afin d'accéder plus facilement à notre SurfaceWindow, implémentez-y le pattern Singleton. De plus, ajoutez un champ de type Color en statique qui contiendra la couleur sélectionnée grâce à la palette.
- Chaque ***SurfaceButton*** de notre palette émet un évènement lorsqu'on lui clique dessus. Complétez le handler *OnClick* de notre palette afin de changer la couleur du trait du ***SurfaceInkCanvas*** définie précédemment dans SurfaceWindow1.xaml en statique grâce à la propriété *DefaultDrawingAttributes* de ***SurfaceInkCanvas***.
- Après avoir testé ceci, nous voulons ajouter un moyen de changer la couleur de fond de notre feuille d'une simple pression. Pour cela, ajoutez une action sur l'événement *TouchExtensions.HoldGesture* de notre ***ScatterViewItem***. Cette action devra changer la couleur de l'arrière plan de la grille principale en fonction de la couleur choisie sur la palette précédemment. L'événement *TouchExtensions.HoldGesture* est déclenché lorsque un contact est maintenu pendant quelques secondes sur un élément.

IV. Les annexes du peintre

Pour le moment nous pouvons dessiner sur la table, mais nous ne pouvons effacer notre beau tableau. Pour effectuer ceci nous allons ajouter un menu à notre application qui nous permettra de changer de mode d'édition : un mode pour dessiner et un mode pour effacer.

1ère étape : Ajout d'un menu

- Ajoutez un ***ScatterViewItem*** de taille 200x200 en désactivant les « effets d'activation » de l'item, ainsi qu'en interdisant son redimensionnement (propriété CanScale).

- Ajoutez-y une ***Grid*** constituée des éléments suivants :
  - Une ***Ellipse*** de taille 200x200, transparente, ayant un rebord (Stroke) noir.
  - Un ***ElementMenu*** de taille 100x100 contenant un premier sous-menu « Mode ». Dans ce sous-menu, ajoutez 2 sous-menus « Dessiner» et « Effacer», chacun d'entre eux contenant une image (crayon.png et eraser.png à ajouter au sein du dossier Resources) grâce à l'élément <s:ElementMenuItem.Icon>.
- Ajoutez un click listener sur les éléments « Dessiner » et « Effacer » de notre menu afin qu'ils réagissent à l'événement *OnClick*. Ce listener permettra de réagir au choix d'un menu et de reporter ce choix sur le mode d'édition du ***SurfaceInkCanvas*** grâce à la propriété *EditingMode*.
- Pour finir, ajoutez une image de taille 200x200 sous l'élément du menu le plus haut niveau. Pour chaque changement de mode, celle-ci devra changer afin de refléter la configuration actuelle (gomme pour le mode « Effacer », crayon pour le mode « Dessiner ». Pour cela, il suffit de changer la Source de cette image afin de pointer sur la bonne ressource.

2ème étape : Ajout d'une bibliothèque d'images

Afin de pouvoir améliorer notre production artistique, nous allons ajouter une bibliothèque d'images à notre application.

- Ajoutez un nouveau ***ScatterViewItem*** ayant un arrière-plan transparent, en désactivant les « effets d'activation » de l'item et en utilisant un ***BorderBursh*** transparent. Cela supprimera complètement le style du ***ScatterViewItem***.
- Ajoutez une ***LibraryBar*** ayant un ***DataTemplate*** contenant une ***Image***. Le ***DataTemplate*** représente la façon dont seront affichés les éléments contenus au sein de la ***LibraryBar***. Ici, nous voulons juste afficher l'image associée à chacun des éléments.

```
<s:LibraryBar.ItemTemplate>
    <DataTemplate>
        <Viewbox Stretch="Uniform">
            <Image Source="{Binding}" />
        </Viewbox>
    </DataTemplate>
</s:LibraryBar.ItemTemplate>
```

- Remplissez cette ***LibraryBar*** au chargement de l'application avec les URI des images présentes par défaut dans le répertoire Images de Windows. Pour cela, utilisez le code suivant au sein de SurfaceWindow1.xaml.cs

```
private void LoadLibraryBarContent()
{
    try
    {
        string publicFoldersPath = Environment.GetEnvironmentVariable("public");
        // These are default OS folders.
        string publicImagesPath = publicFoldersPath + @"\Pictures\Sample Pictures";
```

```
            String[] files = System.IO.Directory.GetFiles(publicImagesPath, "*.jpg");

            items = new System.Collections.ObjectModel.ObservableCollection<String>();

            foreach (String file in files)
            {
                items.Add(file);
            }

            library.ItemsSource = items;

        }
        catch (System.IO.DirectoryNotFoundException)
        {

        }
    }
```

V. Un peu plus d'interactivité

Afin de rendre un peu plus vivante notre application, nous allons rajouter un son lors du changement de mode, ainsi qu'une animation sur le menu. Pour finir, nous essayerons de changer le fond de l'application grâce à un drag'n'drop d'images.

1ère étape : Du son et des animations

- Ajoutez le fichier Sounds.cs que nous vous fournissions à votre projet
- Ajouter les fichiers sons crayon.wav et eraser.wav à votre projet au sein du dossier Resources.
- Jouez les différents sons à chaque changement de menu grâce aux méthodes de la classe Sounds.
- Ajoutez une ***Image*** dans la ***Grid*** constituant le menu sous l'***ElementMenu***. Utilisez les images que nous vous fournissons (« draw.png » et « erase.png ») Afin de déterminer le mode courant (Dessin ou Effacement).
- Nous allons ajouter une animation à cette image afin que celle-ci tourne sur elle-même. Pour cela, ajoutez au sein de cette ***Image***, un ***Image.RenderTransform***, contenant lui-même un ***RotateTransform*** ayant pour centre (100,100). Appliquez à cette transformation, au chargement de notre application, une ***DoubleAnimation*** qui commencera à 0 pour finir à 360, sur une durée de 5 secondes et qui se répètera à l'infini. N'oubliez pas de lancer l'animation.
- A chaque changement de mode, n'oubliez pas de changer la ***Source*** de ***l'Image*** que nous venons de rajouter.

2ère étape : Drag'n'Drop

Maintenant que notre librairie d'images est créée, nous allons ajouter une nouvelle fonctionnalité à notre application : changer le fond de notre feuille de dessin par drag'n'drop d'une image.

- Pour commencer, autorisez la ***ScatterView*** à accepter le drag'n'drop d'autres composants grâce à la propriété AllowDrop
- Ajouter à la ***ScatterView*** un handler sur l'événement *s:SurfaceDragDrop.Drop*. Celui-ci permettra de réagir à l'événement lancé lors du laché d'un élément de la librairie sur notre feuille de dessin.
- Dans cet handler, récupérez la donnée contenu au sein du ***Cursor*** de l'évènement (un ***Cursor*** est une représentation abstraite de l'objet déplacé) sous forme d'une ***String***. Dans notre cas, la donnée est le chemin vers l'image déplacée. (items est la collection de String utilisée pour peupler notre library)

```
String url = event.Cursor.Data as String;
```

- Changer le Background de notre ***ScatterView*** grâce à une ***ImageBrush*** initialisée avec notre image.

```
ImageSource i = new BitmapImage(new Uri(url));
scatterView.Background = new ImageBrush(i);
```

Voila, vous êtes maintenant en possession d'une application de dessin interactive et collaborative.

VI. Pour aller plus loin

Grâce à l'utilisation de tags, créez des tampons ajoutant un nouveau ***ScatterViewItem*** contenant une image à chaque appui.

A l'heure actuelle, plusieurs utilisateurs avec plusieurs palettes interagissent sur le même ***SurfaceInkCanvas***, ce qui peut poser des problèmes. Améliorez l'application actuelle afin qu'elle puisse gérer plusieurs feuilles de dessin liées chacune à sa propre palette.