Université de la Rochelle INFO-12605C - IHM TP10 : Contrôles utilisateurs avec .NET et C#

© B. Besserer, R. Péteri

Année universitaire 2017-2018

— Version 2 - Séance de TP du 30 mars 9h30-12h40 —

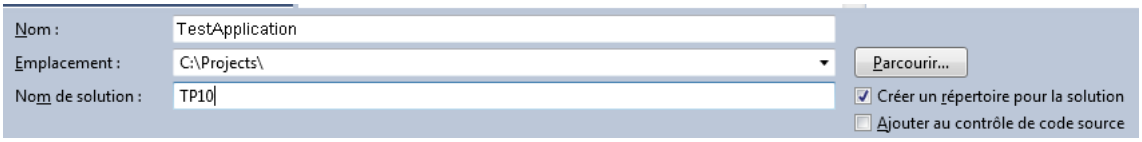
Le but de ce TP est de développer des contrôles IHM spécifiques (*user-defined controls*) qui peuvent ensuite être utilisés, comme les contrôles standard, pour tous les projets. On utilisera aussi les composants développés par d'autres.

La dernière partie du TP (slot machine) constitue un mini-projet, dont un court compte-rendu sera à déposer sous Moodle à la fin de la séance : lignes essentielles du code avec commentaires, propriétés exposées à l'utilisateur, copies d'écrans, ...

1 Configuration du projet et utilisation de contrôles existants

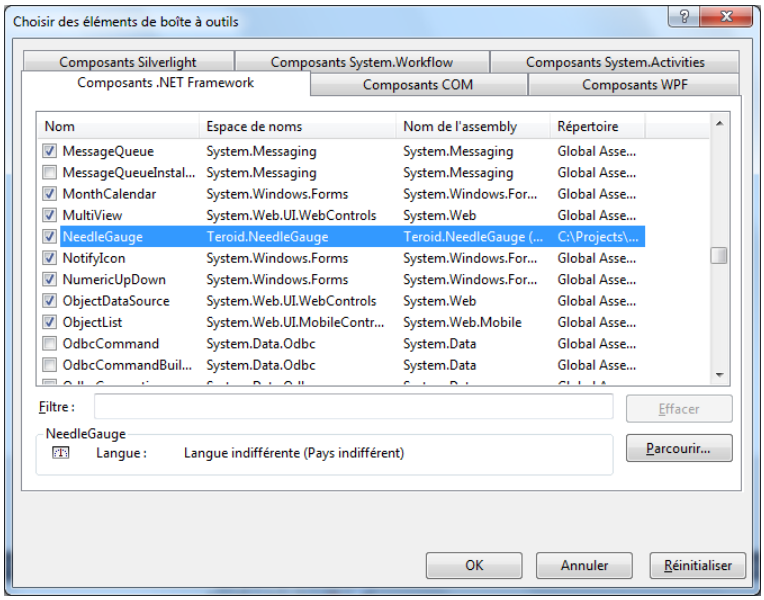
Attention : lors de ce TP, la solution comportera au minimum 2 projets : une application de test (ou programme hôte) et le contrôle que vous allez développer. A partir de la section 2 de ce TP, vous avez d'autres contrôles à développer, cette même solution contiendra donc potentiellement deux projets supplémentaires.

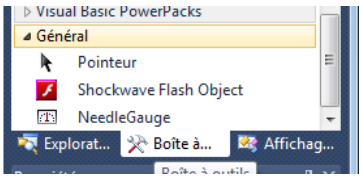
Commencez par créer un nouveau projet de type Windows Forms, que vous baptiserez **TestApplication**, et qui sera créé dans une solution nommée **TP10**.



Cette application de test disposera d'un contrôle de type timer, et d'une variable membre `m_rand` de la classe `Random`. A chaque Tick du timer, on récupère une nouvelle valeur de ce générateur de nombre aléatoire (`m_rand.Next()`)
Vous trouverez sous Moodle une archive nommée Needle Gauge contenant une DLL et un fichier d'aide .CHM. La DLL contient le binaire nécessaire à la création et l'utilisation d'afficheurs à aiguille (un type d'afficheur non disponible parmi les contrôles standards Microsoft). Jetez un oeil sur le fichier d'aide (double-cliquez le CHM après l'avoir extrait de l'archive). Si vous n'arrivez pas à ouvrir celui-ci ou en voir le contenu, vous pouvez ouvrir le fichier (CHM = Compressed HTML) avec 7Zip ou un utilitaire équivalent et extraire les fichiers HTML et l'arborescence nécessaire pour visualiser l'aide avec un navigateur. Si les contrôles sont bien conçus, la documentation est superflue...
Pour utiliser ces contrôles, il faut copier la DLL — n'importe où (contrairement aux objets COM, les composants .NET n'ont pas besoin d'inscription dans le registre système, et l'emplacement de la DLL est moins critique). Allez sur la vue [Design] de votre application de test. Ouvrez la boite à outils (CTL+ALT+X). Faites un clic droit dans la boite à outil, et sélectionner Choisir les éléments...

Dans la boite de dialogue qui s'affiche, sélectionnez l'onglet *composants .NET framework*, puis cliquez sur *Parcourir...*, et sélectionnez la DLL contenant les contrôles d'affichage à aiguilles. Vous devez obtenir une vue similaire à celle ci-après ; le ou les contrôle(s) apparaissent alors dans la boite à outils.





Mettez en place, dans votre application test, au moins deux instances d'un contrôle de type affichage à aiguille, que vous personnaliserez (graphismes différents, angle et valeurs différentes), et les aiguilles seront animés pas le générateur de nombre aléatoire.

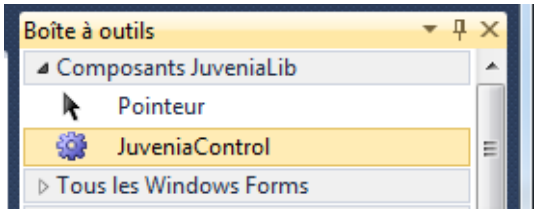
2 Créez votre propre contrôle personnalisé

Ajoutez un nouveau projet de type Bibliothèque de contrôles WindowsForms à votre solution, projet que vous nommerez JuveniaLib. L'assistant vous créera normalement un fichier nommé `UserControl1.cs`, fichier que vous renommerez `JuveniaControl.cs` (dans l'explorateur de solution, clic-droit sur le fichier, choisir renommer).

Dans un premier temps et pour vérifier le bon comportement de votre contrôle personnalisé, celui-ci disposera de deux contrôles standard : un TextBox et un Timer. Vous pouvez ajouter ces éléments par drag & drop sur la surface [Design] de votre contrôle personnalisé. Ensuite :

- Marquez le TextBox en lecture seule.
- Réglez le timer sur 1000 ms.
- Écrivez la routine de gestion de l'événement Tick pour ce timer (sélectionnez le timer, affichez les message, ...), ajoutez la ligne : `textBox1.Text = DateTime.Now.ToString();`
- Dans le constructeur de `JuveniaControl.cs`, apres la ligne `InitializeComponent();`, écrivez `timer1.Start();`

Testez votre contrôle personnalisé. Soit la bibliothèque de contrôles est considérée comme "projet de démarrage", votre contrôle personnalisé sera alors affiché dans un conteneur spécifique (de test). Vous pouvez aussi sélectionner votre projet TestApplication comme projet de démarrage (clic-droit sur le projet dans l'explorateur de solution, choisir Definir comme projet de démarrage). Dans ce cas, votre application de test contiendra un contrôle de type JuveniaControl. Sachez que votre contrôle personnalisé sera visible dans la boite à outils (comme ci-dessous).



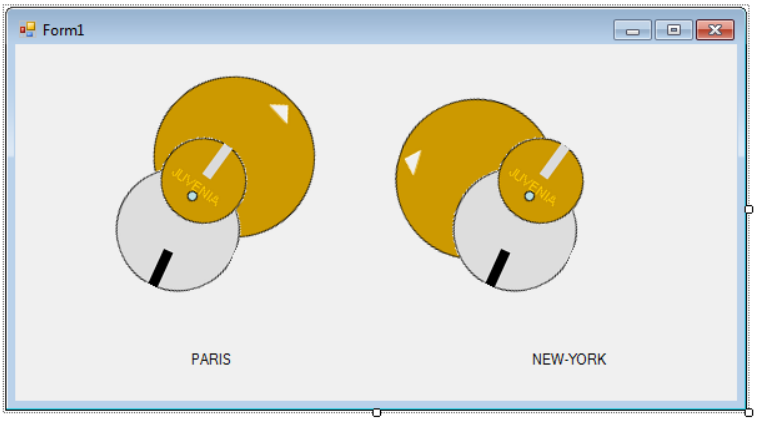
Si votre "horloge textuelle" fonctionne, vous allez maintenant modifier le code et implémenter comme contrôle utilisateur l'afficheur de la montre Juvenia que nous avons vu en TD.

Commencez par importer les bitmaps comme ressources : pour cela, effectuez un clic-droit sur le projet et demandez les propriétés du projet. Sélectionnez 'ressource' et insérez les images. C'est parfois un peu délicat à réaliser du premier coup, mais au final, votre projet doit comporter un répertoire 'ressources', contenant vos images, et vous devez pouvoir les charger de la façon suivante, dans le constructeur du

contrôle (`m_SecondBitmap` est une donnée membre de type Bitmap) ;

```
m_SecondBitmap = new Bitmap(JuveniaLib.Properties.Resources.SecondDial);
m_SecondBitmap.MakeTransparent(Color.White);  // le blanc est considéré comme transparent
```

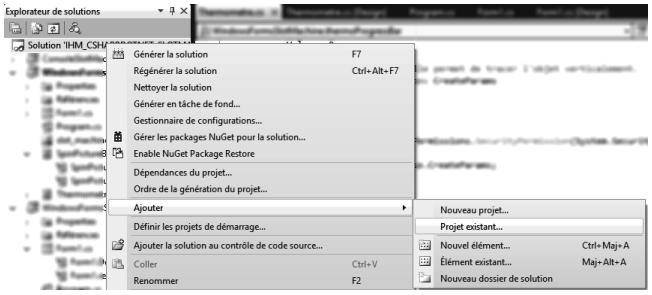
Lors du Tick, appelez uniquement la méthode Invalidate(). La méthode Paint du conteneur du controle fera l'affichage (voir TD).
Ajoutez une propriété permettant un offset au niveau de l'heure, permettant l'affichage dans une autre zone horaire. Testez le fonctionnement de votre contrôle Juvenia dans votre application de Test (2 instances du contrôle, affichage temps réel de l'heure locale et de l'heure à New-York (-6heures))



2

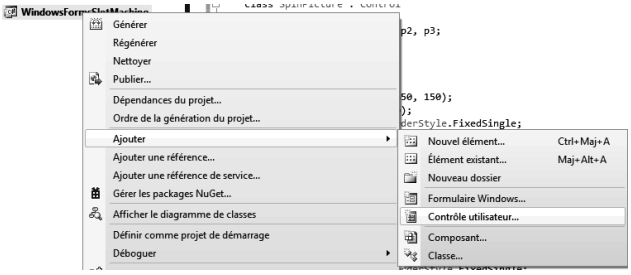
3 Las Vegas...

Pour cette partie, vous ferez un court compte-rendu que vos déposerez sous Moodle. Le compte-rendu sera bref et contiendra les parties importantes de votre code et des captures de l'exécution de l'application.



Récupérez l'archive TP10_part2 ; c'est un projet que vous ajouterez à votre solution (décompressez, copier le répertoire dans le répertoire contenant la solution que vous avez créé pour l'exercice précédent. Ensuite, , clic-droit sur la solution dans l'Explorateur de solution, ajouter -> projet existant).
Marquez ce projet comme projet de démarrage. Testez le programme. Examinez le code. Il s'agit d'une implémentation rudimentaire d'une machine à sous (slot machine). En partant de ce projet, on vous demande :

1. De créer un contrôle affichant non pas une chaîne de caractères, mais des images :
   - Créez un nouveau contrôle utilisateur (voir image ci-contre) nommé SpinPictureBox. La vue [design] de ce nouveau contrôle montre un conteneur vide. Redimensionnez celui-ci, placez-y 3 contrôles de type PictureBox cote à cote. Modifiez les *properties* des pictureBox : Nommez les pictureBox p1 p2 et p3, SizeMode = Zoom, et rendez les cadres visibles autour de ces pictureBox (borderStyle = borderStyle.Simple)



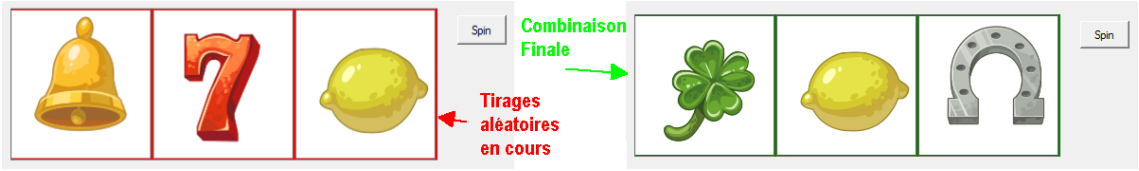
   - Dimensionnez les pictureBox de manière à voir un peu du conteneur autour et entre les pictureBox.

   Complétez le code de ce contrôle utilisateur :
   - Dans le constructeur de ce contrôle, Chargez la bitmap symbols.png (ou encore mieux, ajoutez cette bitmap aux ressources).
   - ajouter la méthode `Spin(int duration)` en vous inspirant de la méthode Spin de la version fournie. Au lieu récupérer un caractère dans un tableau de caractères, vous allez récupérer aléatoirement un objet de type rectangle depuis un tableau d'objets de type rectangle, chaque rectangle

correspondant à une portion de l'image chargée. Ensuite, utilisez la méthode
`Bitmap cloneBitmap = myBitmap.Clone(cloneRect, format);`
par exemple : `smallBitmap = FullBitmap.Clone(myRectTab[i], FullBitmal.PixelFormat);`
ou i serait une valeur aléatoire.

> Pour les images (comme pour les fichiers audio), si vous ne précisez pas le chemin et mettez un code sous la forme : `myBitmap = new Bitmap("myImage.png");` Alors ce fichier image doit être placé dans le même répertoire que l'exécutable (bin/debug si vous travaillez en mode debug)

- Modifiez la couleur de fond (conteneur) : rouge quand les tirages aléatoires sont en cours, vert quand la combinaison finale est affichée.



☑3

2. Nous allons maintenant utiliser un contrôle orienté à la verticale. Attention, sous WindowsForms, il est plus facile de manipuler des TrackBar et ProgressBar dans leur orientation horizontale que verticale, il y des limitations dans ce dernier cas.
   On demande de dériver une classe myProgressBar de la classe ProgressBar (voir code fourni). La ProgressBar sert d'habitude à afficher l'avancement d'une action en dessinant des blocs, nous allons nous en servir pour représenter des pièces (mise de départ, condition nécessaire pour jouer).

   - Dans le constructeur de votre classe dérivée, positionnez la valeur maximale à 3. Chargez la Bitmap "coins.png" ou mieux, ajouter cette image aux ressources et chargez la Bitmap depuis les ressources.
   - Déterminez la largeur du contrôle afin que celui-ci puisse représenter l'image de 3 pièces superposées.
   - Ajoutez cette ligne à la fin du constructeur : `SetStyle(ControlStyles.UserPaint, true);` indiquant que la méthode de dessin sera celle que vous allez implémenter et non celle de la classe de base.

   Ensuite surchargez la methode OnPaint et ajoutez-y votre code de dessin, vous devez dessinez autant de pièces que le contrôle a reçu de clics.

```
protected override void OnPaint(PaintEventArgs e)
{
  Rectangle rect = new Rectangle(0, 0, this.Width, this.Height);
  if (ProgressBarRenderer.IsSupported)
    ProgressBarRenderer.DrawVerticalBar(e.Graphics, rect);    // ceci dessine le rectangle standard de la progressBar

// ajoutez ici votre code de dessin personalisé
}
```

   Important : Il faudra associez à l'événement "clic souris" dans ce contrôle l'incrémentation de la valeur du progressBar (attacher une méthode à l'événement OnClick). Il vous faudra rajouter des variables membres à la classe myProgressBar (éventuellement accessible via des *properties*), peut être une méthode ou une autre technique pour limiter le nombre de pièce à 3 (même si l'on continue à cliquer). La présence du nombre maximum de pièces est nécessaire pour jouer, et les pièces disparaissent alors.

   Attention, les contrôles sont indépendants. Il ne faut pas d'interaction directes **entre les contrôles**, les interactions passent par le programme principal.

☑4 Pour eviter un clignotement de vos bitmaps dans le contrôle, mettez la ligne `Application.EnableVisualStyles();` en commentaire dans le fichier `Program.cs` du programme principal.

3. Pour finir, après l'image, le son...
   Pour jouer un son :

```
System.Media.SoundPlayer player;
player.SoundLocation = "ding.wav";    // emplacement du fichier sonore
player.Load();                        // chargement du fichier sonore en mémoire
player.Play();                        // on joue le son
```

   - Pour chaque changement d'image **dans le dernier pictureBox (celui qui tourne le plus longtemps)**, jouez le son "ding.wav". Réfléchissez au placement de chaque bout de code.
   - En cas de victoire, jouer le son "win.wav". Pour augmenter les chances de gagner, remplacez la condition pour gagner qui est : *[trois symboles identiques]* par la condition *[au moins deux symboles identiques]*.

☑5