

TP C#0

Rendu

Archive

Vous devez rendre un fichier zip avec l'architecture suivante :

```
- rendu-tp-prénom.nom.zip
  |- rendu-tp-prenom.nom/
    |- AUTHORS
    |- README
    |- HelloWorld/
      |- HelloWorld.sln
      |- HelloWorld/
        |- Tout sauf bin/ and obj/
    |- MyCalc/
      |- MyCalc.sln
      |- MyCalc/
        |- Tout sauf bin/ and obj/
    |- ClickerGame/
      |- ClickerGame.sln
      |- ClickerGame/
        |- Tout sauf bin/ and obj/
```

- Il faut bien évidemment remplacer *prenom.nom* par votre nom.
- Le code doit compiler (appuyer sur F5 doit afficher une Windows Form sans erreur de compilation).
- Dans ce TP, vous êtes autorisés à utiliser des images et des sons comme boni, toutefois vous devez garder une taille d'archive légère.

AUTHORS

Ce fichier doit contenir : une étoile (*), un espace, votre login et une nouvelle ligne.
Voici un exemple (où \$ représente une nouvelle ligne) :

```
* prenom.nom$
```

Merci de noter que le nom du fichier est AUTHORS SANS extension.

1 Introduction

1.1 Objectifs

Durant ce TP, nous allons aborder des nouvelles notions comme :

- La découverte de MonoDevelop
- Le langage C#
- Les Windows Forms

1.2 Visual Studio

Microsoft Visual Studio est un "integrated development environment" (IDE) de Microsoft. Il est utilisé pour développer des programmes pour Microsoft Windows comme beaucoup d'autres choses sur le Web. Il n'est disponible que sur Windows.

Visual Studio inclut un éditeur de code qui supporte une complétion du code et un puissant débbugger intégré. Visual Studio supporte différents langages comme le C, C++, C#, VB.NET ou encore F#.

Vous pouvez télécharger Visual Studio Community gratuitement sur www.visualstudio.com ou Visual Studio Enterprise sur Msdn qui est sur intra-bocal.epitech.eu.

1.3 MonoDevelop

Monodevelop est un IDE libre disponible sur Windows, OS X et Linux.

MonoDevelop, comme Visual Studio, inclut un éditeur de code qui supporte une complétion du code et un débbugger intégré. Il supporte différents langages comme le C# et les autres langages .NET.

Vous pouvez télécharger MonoDevelop gratuitement sur www.monodevelop.com

1.4 Le langage C#

Le C# est un langage orienté objet créé par Microsoft et développé par une équipe dirigée par le créateur de Delphi, Anders Hejlsberg.

Le C# est un langage dérivé du C et du C++ et ressemble au Java. C# est utilisé pour les applications web, les applications de bureau ou encore les services du web par exemple.

Le C# est différent de ce que vous avez utilisé jusqu'à maintenant. Vous aurez l'occasion de le découvrir et de vous y habituer pendant le reste de l'année. N'hésitez pas à aller sur MSDN.com pour mieux comprendre son fonctionnement.

2 Cours

2.1 Les bases du C#

Voici quelques bases dont vous aurez besoin pour les exercices de ce TP. Vous verrez les notions abordées ici plus en détails dans les semaines à venir, mais n'hésitez pas à découvrir le langage par vous-mêmes. Vous pouvez en apprendre plus sur le C# en allant sur MSDN. Si jamais vous avez besoin de mieux comprendre quelque chose en C#, pensez à vous référer à MSDN !

- Chaque instruction se termine par un point-virgule
- Les instructions sont contenues dans des blocs délimités par des accolades

- Une variable a un type et une valeur qui peut être changée grâce à l'opérateur =
- Une fonction a un type de retour et peut prendre des paramètres

Pour ce TP, vous n'aurez besoin que d'entiers (type *int*), mais vous pouvez utiliser d'autres types de variables si vous le souhaitez (MSDN est votre ami). Vous n'aurez pas besoin de déclarer de fonctions mais vous travaillerez sur des fonctions d'une forme similaire à celle-ci :

```
1 private void nom_de_fonction(type1 arg1, type2 arg2, . . .)
2 {
3     // On est ici dans le bloc d'instructions de la fonction
4     /*
5     Cette fonction ne
6     retourne rien car son
7     type de retour est void.
8     */
9 }
```

Ignorez le mot-clé *private* et les arguments de la fonction pour l'instant, vous n'en aurez pas besoin pour cette semaine.

Dans l'exemple précédent, des commentaires sont écrits dans le corps de la fonction. La syntaxe des commentaires sur une seule ligne est donc (*// commentaire*) et (*/* commentaire */*) pour les commentaires multi-lignes. À part pour la syntaxe, les commentaires fonctionnent de la même façon qu'en Caml.

Vous pouvez déclarer une variable d'entier de cette façon :

```
1 int nom_de_variable; // la valeur de nom_de_variable est 0
*
1 if (condition) // une condition peut être par exemple a == b
2 {
3     /* Le code exécuté si la condition est vraie */
4 }
5 else
6 {
7     /* Le code exécuté si la condition est fausse */
8 }
```

Pour comparer 2 valeurs, vous pouvez utiliser les opérateurs suivants :

- *a == b*, pour vérifier si *a* est égal à *b* (comme *=* en Caml)
- *a != b*, pour vérifier si *a* est différent de *b* (comme *<>* en Caml)
- *>*, *>=*, *<* and *<=* fonctionnent comme en Caml

Les expressions booléennes fonctionnent aussi, vous pouvez en apprendre plus sur MSDN.

2.2 Les bases du Windows Form

Les Windows Forms sont un moyen simple de créer une interface utilisateur sur Windows. Ils fonctionnent via des événements. Cela signifie que lorsque l'utilisateur interagit avec un élément, une fonction associée à cet élément est appelée.

Pour placer un élément dans votre form, vous pouvez simplement glisser-déposer un objet depuis la Toolbox de Visual Studio. Vous pouvez ensuite modifier votre élément via l'onglet de propriétés (couleur, text, nom, ...).

Pour créer une fonction qui sera appelée lorsque l'utilisateur cliquera sur un élément, double-cliquez sur ce dernier.

2.3 Les bases de MonoDevelop

Pour ouvrir un projet :

Fichier -> Ouvrir -> Choisir un fichier SLN

Vous trouverez l'explorateur de solution sur la gauche de l'écran. Cet onglet affiche les fichiers et dossiers de chaque projet dans la solution.

Solution WindowsFormsApplication (1 projet)

- WindowsFormsApplication : application
 - Properties : Ressources du projet
 - Reference : Bibliothèques utilisées par l'application
 - Form1.cs : Contient votre code et affiche l'éditeur graphique
 - Form1.Designer.cs : Le code généré par l'écriture
 - Form1.resx : Contient votre code
 - Program.cs : Point d'entrée du programme

Vous pouvez compiler et lancer votre projet en appuyant sur F5.

3 Exercice 1 : Hello World

Ouvrez le projet "HelloWorld". Si vous essayez de compiler avec F5, vous verrez une fenêtre Windows Form avec un bouton et une zone de texte. Ce sont, ce qu'on appelle en C#, des objets. En C#, chaque objet a des attributs ; ici le bouton a différents attributs comme le texte qu'il affiche (*Text*), la couleur de la police (*ForeColor*) ou encore la couleur de fond (*BackColor*). Fermez la fenêtre et commençons cet exercice.

Premièrement, double-cliquez sur le fichier Form1.cs situé dans la fenêtre "Solution" sur la gauche de MonoDevelop. Vous vous trouvez maintenant devant, peut-être, votre premier code en C#. Dans ce fichier, seules 2 fonctions vont nous intéresser. Ne vous préoccupez pour l'instant pas du reste du code, vous en saurez plus, plus tard.

La première fonction qui nous intéresse est :

```
1 public Form1()
2 {
3     InitializeComponent();
4     //On écrit le code ici.
5 }
```

Cette fonction est appelée au lancement de votre fenêtre. C'est donc dans cette fonction que vous pourrez faire vos initialisations de départ (les initialisations de l'environnement ainsi que les initialisations de chacun de vos objets). Dans cette fonction, une autre fonction est appelée, n'enlevez jamais cet appel de fonction sinon votre Windows Form ne fonctionnera plus.

La deuxième fonction est liée au bouton de notre fenêtre. Celle-ci va être appelée lors d'un clic sur ce bouton.

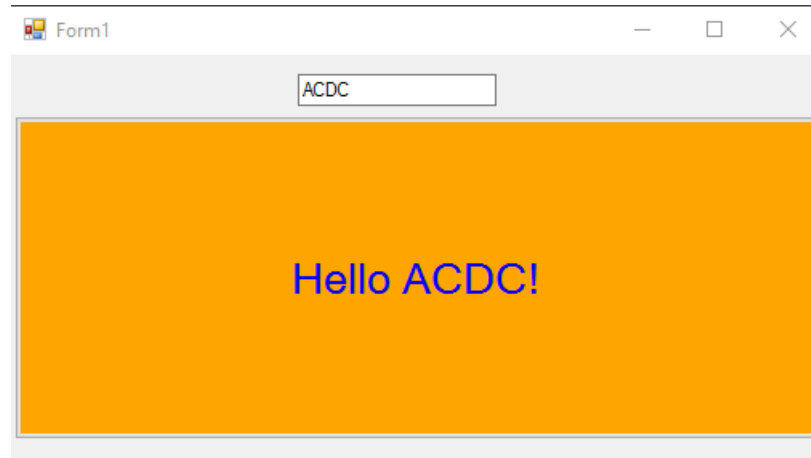
```
1 private void button1_Click(object sender, EventArgs e)
2 {
3     //On écrit le code ici.
4 }
```

Sur le bouton, initialiser le texte à « Hello World! » avec un fond vert et une couleur de texte rouge. Lorsqu'on clique sur ce bouton, le texte, la couleur du fond et la couleur du texte doivent changer pour ce que vous voulez. En revanche, lorsque l'on clique une deuxième fois sur ce bouton, il doit revenir à son état initial.

Informations sur les objets : Le nom du bouton est "button1" et celui de la zone de texte (pour les bonus) est "textBox1".

Bonus :

- Créez une *Textbox* afin d'afficher "Hello World!" sur le bouton si la zone de texte est vide et "Hello ACDC!" (par exemple) si "ACDC" est écrit dans la zone de texte.
- Faites parler votre imagination (plus de couleurs, de fonctionnalités, de boutons tant que ce qui est demandé au départ est respecté).



4 Exercice 2 : MyCalc

Ouvrez le projet MyCalc.

Dans cet exercice, vous devez créer une calculatrice simple. Vous devrez implémenter cinq opérations basiques : l'addition, la soustraction, la multiplication, la division et le modulo.

Votre calculatrice se composera de cinq objets : deux objets *NumericUpDown* qui représentent les deux termes de l'opération, une *ComboBox* pour les différents opérateurs, un *Button* pour soumettre l'opération et un *Label* pour afficher le résultat de l'opération.

L'utilisateur rentrera un nombre dans chaque *NumericUpDown*, choisira une des cinq opérations disponibles et cliquera sur le bouton "Result" afin d'afficher le résultat. Les objets *NumericUpDown* n'acceptent que des nombres, il n'y a donc pas à gérer le cas où l'utilisateur ne rentrerait pas des nombres au programme.

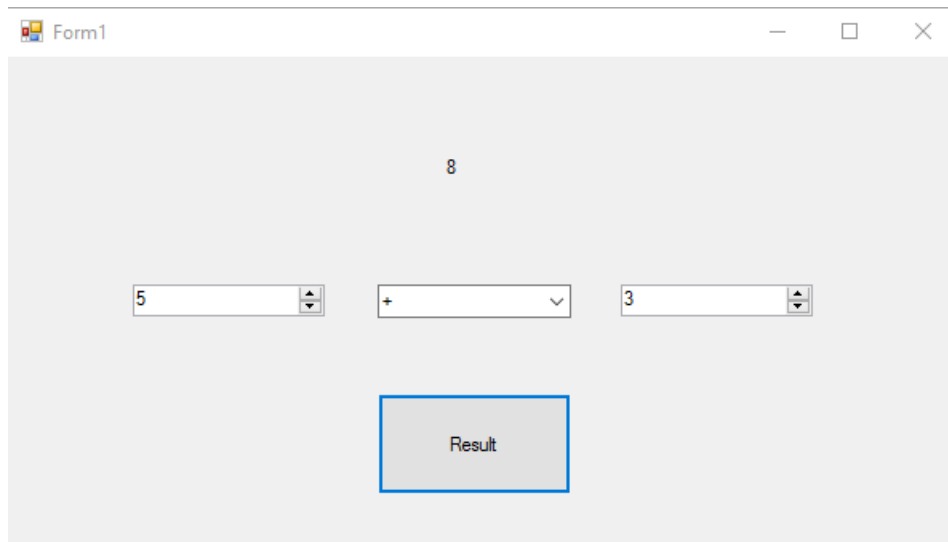
À l'aide des instructions *if* et *else*, vérifier quel opérateur est choisi dans le *ComboBox* par l'utilisateur après un clic sur le bouton "Result", récupérer les valeurs des objets *NumericUpDown* et affichez le résultat correspondant dans l'objet *Label*.

N'oubliez pas de gérer les cas spéciaux.

Informations sur les objets : les noms des *NumericUpDown* sont "nb1" et "nb2", le nom du *ComboBox* est "comboBox1", le nom du *Button* est "button1" et enfin, le nom du *Label* est "label1". Les items du *ComboBox* sont : "+", "-", "*", "/" et "%" dans cet ordre.

Conseil 1 : Pour écrire dans le label, on utilisera la méthode "ToString()" qui convertit un nombre en une chaîne de caractère.

Conseil 2 : Regardez MSDN si vous êtes perdus par rapport aux *ComboBox* ou aux objets *NumericUpDown*.



Bonus : Ajoutez de la couleur, des sons et encore une fois, laissez parler votre imagination.

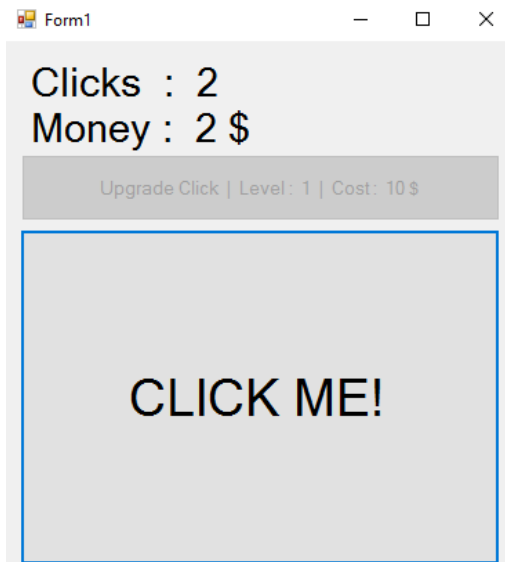
5 Exercice 3 : Clicker Game

Ouvrez le projet ClickerGame.

Dans cet exercice, vous créerez un jeu de clics (comme Cookie Clicker mais votre jeu à vous). Vous êtes libres de personnaliser votre jeu une fois que vous l'aurez fini (images, sons, ... soyez créatifs!).

5.1 Partie A

A la fin de cette partie, votre jeu ressemblera à ceci :



Vous aurez besoin de :

- Un bouton principal sur lequel cliquer : `b_clicker`
- Un label pour afficher le nombre de clics effectués sur le bouton principal : `l_clicks`
- Un label pour afficher l'argent du joueur : `l_money`
- Un bouton pour améliorer le clic du joueur : `b_upgrade`

Fonctionnalités :

- Cliquer sur le bouton principal doit augmenter le compte des clics de 1 (en haut de la fenêtre sur l'exemple).
- Cliquer sur le bouton principal doit augmenter l'argent du joueur.
- La quantité d'argent par clic doit commencer à 1 et augmenter de 1 à chaque amélioration achetée par le joueur (*Level* sur les exemples).
- Le prix de l'amélioration doit doubler à chaque fois qu'elle est achetée. Le prix commence à 10.
- Le bouton d'amélioration doit afficher le "niveau" actuel (qui est l'argent par clic) et le coût du niveau d'amélioration suivant.
- Le bouton d'amélioration ne doit être *Enabled* (regardez MSDN) que lorsque le joueur a assez d'argent pour l'acheter.

Indice 1 : pour utiliser une variable entre plusieurs appels de fonction, il faut la déclarer dans la classe, avant la ligne `public void Form1()`

Indice 2 : pour changer la valeur de départ de ces variables, vous devez changer leur valeur juste en-dessous de la ligne `InitializeComponents()` ; dans la même fonction.

5.2 Partie B

Vous avez maintenant un jeu de clics simple qui fonctionne. Toutefois, il devient vite répétitif avec un seul type d'amélioration. Dans cette partie, vous ajouterez de l'aléatoire au jeu via un objet *Random*.

Voici comment l'utiliser :

```

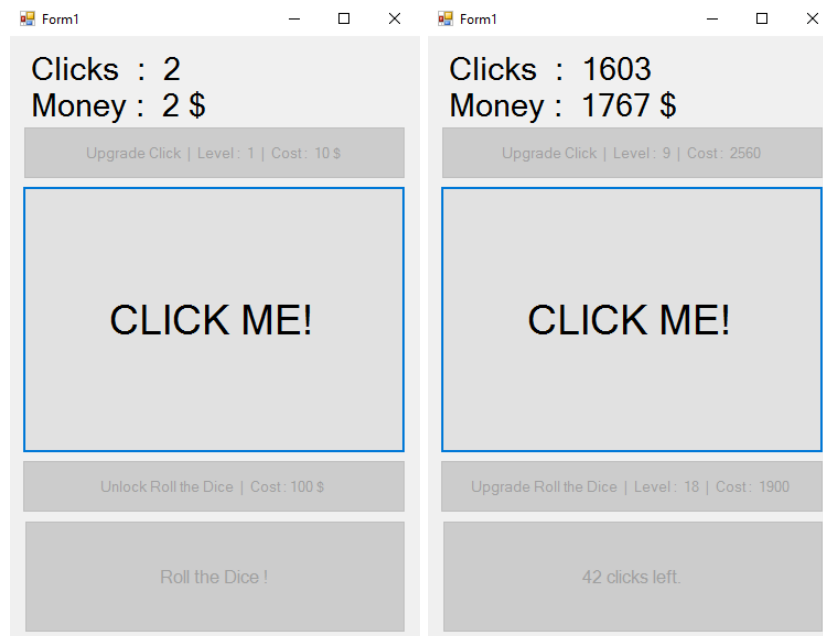
1 Random rand; // on le déclare
2
3 public Form1()
4 {
5     InitializeComponent();
6     rand = new Random(); // on l'initialise
7 }
8 /*
9 On peut maintenant utiliser rand.Next()
10 pour obtenir un entier aléatoire.
11 Allez sur MSDN pour en apprendre plus.
12 */

```

Vous aurez besoin de :

- Un bouton pour la nouvelle mécanique de jeu "Lancer les dés!" : `b_rng`
- Un bouton pour débloquer et améliorer cette nouvelle fonctionnalité : `b_rng_upgrade`

Après cette partie, votre jeu ressemblera à ceci :



Fonctionnalités :

- Le nouveau bouton doit être verrouillé tant que la nouvelle amélioration n'a pas été achetée au moins une fois. Le coût initial est de 100.
- A chaque fois que l'amélioration est achetée, son coût doit augmenter de 100.
- Une fois que la fonctionnalité est débloquée, un compte à rebours de clics doit apparaître sur le bouton pour lancer les dés.
- Après 100 clics, le bouton devient actif.
- Cliquer sur le nouveau bouton doit ajouter une quantité aléatoire d'argent au joueur, dans le champ $[-100; \text{coût}]$ où coût est le coût du prochain niveau d'amélioration. Par exemple, au niveau 1, la quantité aléatoire ajoutée sera entre -100 et 200.
- Cliquer sur le bouton "lancer les dés" le verrouille à nouveau pendant 100 clics du bouton principal.
- L'argent du joueur ne peut pas descendre en dessous de 0.

Améliorer la nouvelle fonctionnalité signifie soit la débloquent (la première fois) soit augmenter de 100 son coût (et donc la borne supérieure du champ de "Lancer les dés!").

À nouveau, le bouton d'amélioration ne doit être activé que si le joueur a suffisamment d'argent pour l'acheter.

N'oubliez pas que vous pouvez personnaliser votre jeu comme vous le souhaitez (couleurs, images, sons, ...). Soyez créatifs !

The code is the law.