Développement Logiciel

Machine à sous évoluée

IDSOUGOU Yassir - KASTNER Damien

Promo 2019

22/02/2018

Introduction

Ce TP a pour but de développer une application reproduisant le comportement d'une machine à sous sur l'environnement de développement Microsoft Visual Studio$_{2008}$.

La partie métier du logiciel est constituée des classes développées lors de la séance "Composition d'objets" des TPs de C++. De même, les données permettant d'initialiser la machine à sous proviennent des fichiers textes d'extensions respectives `.smb` et `.cmb`.

Un certain nombre de fichiers sont fournis dans les archives `ClassesTP`, `FichiersInit` et `Images` accessibles via Moodle (cours "Outils infos.").

L'interface sera quant à elle constituée de deux formulaires Windows élaborés en C++/CLI à l'aide de l'environnement de développement intégré Visual Studio .NET.

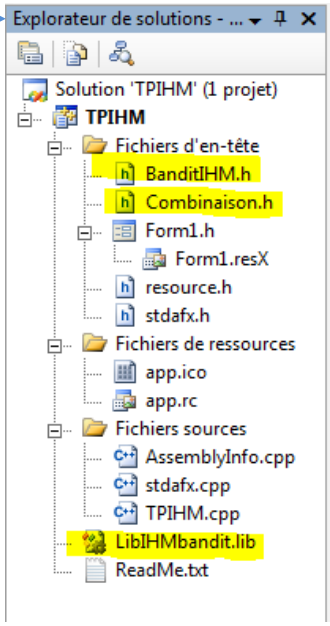
1. Analyse des fichiers disponibles

Les fichiers contenus dans `ClassesTP` correspondent aux fichiers header (.h) des classes `BanditIHM` et `Combinaison` ainsi qu'une librairie `LibIHMbandit` contenant les définitions des méthodes de ces classes précompilées.

Ce répertoire `ClassesTP` contient les classes avec leurs différentes méthodes qui définissent le comportement de notre machine à sous.

Pour utiliser ces fichiers, il faut les mettre dans le répertoire `TPIHM` qui correspond à notre projet, ceci sera visible sur l'`Explorateur de solutions` ci-dessous.

`Explorateur de solutions` →



Tout d'abord, on a inclut les fichiers .h au début du script `Form1.h`, qui sera le formulaire de jeu, afin de pouvoir utiliser les classes `BanditIHM` et `Combinaison`, ainsi que leurs méthodes associées:

```
#include"../ClassesTP/BanditIHM.h"
#include"../ClassesTP/Combinaison.h"
```

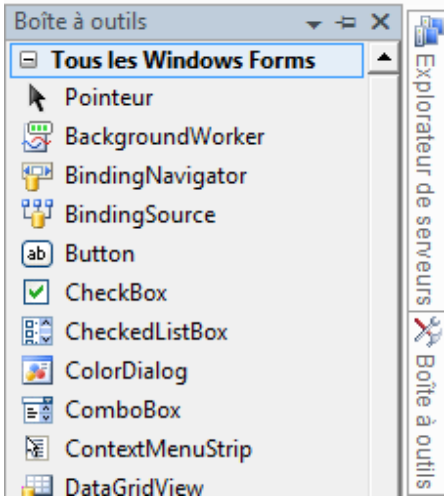
Pour un premier test, on fait l'instance d'un objet `BanditIHM` dans le constructeur du formulaire

2. Conception du formulaire de jeu

2.1 Instanciation de la fenêtre de jeu

Dans `#pragma region Windows Form Designer generated code` , on créé l'interface graphique. Le code est généré automatiquement lorsque nous mettons en place les différents éléments de l'interface à partir de `Form1.h[Design]`.

La disposition, la taille et autres caractéristiques de l'interface graphique sont manipulables de manière dynamique et sont codés automatiquement sans que nous ayons à taper cette partie du code.



La boîte à outils permet d'ajouter des éléments avec lesquels l'utilisateur va interagir (Boutons, Menu Contextuel, Checkbox, Listes, Labels, etc...).

Lors de l'ajout d'un élément, on peut choisir l'action que l'utilisateur devra faire pour générer un évènement (passage du curseur, click droit/gauche, etc...). Le code est généré automatiquement dans `Form1.h` . Cependant, il est nécessaire de coder l'action que cet évènement va engendrer.

Voici un exemple du code généré automatiquement dans `Form1.h` lors de l'ajout d'un click bouton sur `Form1.h[Design]` (Sans l'ajout du code de l'action):

```
private:System::VoidClickButton1(System::Object^sender,
                    System::EventArgs^e) {

        }
```

Voici un exemple du code généré automatiquement dans Form1.h lors de l'ajout d'un click sur un `menuStrip` dans `Form1.h[Design]` (Avec l'ajout du code de l'action):

```
private: System::Void ToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^  e) {
     cagnotte = 0;
     label3->Text = cagnotte.ToString();
                }
```

Il s'agit ici du code du bouton ***RAZ*** de l'interface, qui permet la remise à zéro de la partie.

2.2 Instanciation de la machine à sous

Dans `namespace TPIHM`, on trouve le constructeur de la classe `Form1` dans laquelle on définit un objet de la classe `BanditIHM` (nommé `bandit1`) qui prend 2 arguments : un fichier `.smb` comprenant les différents symboles de la machines et un fichier `.cmb` comprenant les combinaisons gagnantes et le facteur de gain qui leur est associé.

```
bandit1 = new BanditIHM("!","!");
```

Cet objet `bandit1` doit être d'un droit d'accès `public` pour pouvoir être modifié si l'utilisateur veut changer de paramètres.

Le destructeur de la classe `Form1` est généré automatiquement. L'objet `bandit1` est détruit en même temps que l'objet `Form1` car il est un composant de cet objet. Ils sont donc détruits à la fermeture de l'application.

2.3 Instanciation des événements de jeu

Lors du jeu, trois évènements sont mis à la disposition de l'utilisateur :

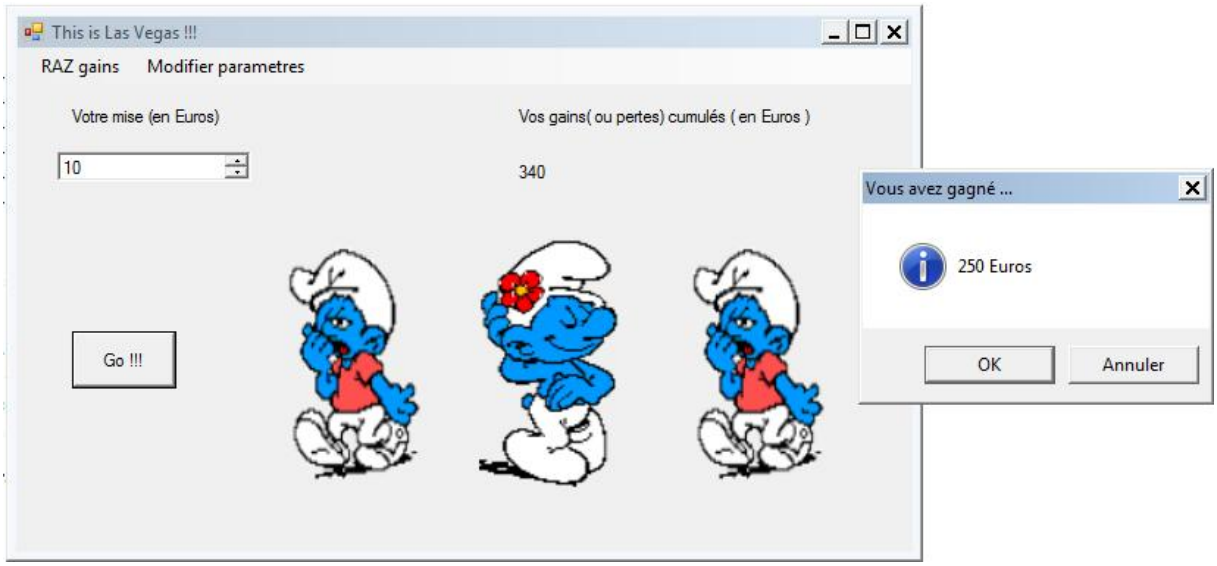
- ✓ Déterminer la mise, via un `numericUpDown`
- ✓ Lancer le jeu, via un `Button`
- ✓ Remettre les gains à zéro, via un `menuStrip`

L'interface graphique contient des éléments de visualisation non modifiables par l'utilisateur:

- ✓ Des `label` (zones de texte) contenant le gain cumulé en euros et des commentaires indicatifs sous forme de texte
  - « `Votre mise (en Euros)` »
  - « `Vos gains (ou pertes) cumulés(en Euros)` »
- ✓ Des `pictureBox` contenant des images associées aux symboles et servant à afficher les combinaisons.
- ✓ Une `MessageBox` qui s'ouvre lorsque la combinaison est gagnante et précisant le gain obtenu.

2.4 Modification des listes d'images

Les images associées aux symboles peuvent être modifiées rapidement par le concepteur. Elles sont contenues dans `imageList` où chaque image est indexée par un numéro correspondant à un symbole. Le concepteur peut alors facilement choisir de changer les images (Schtroumpfs, chats, etc).



Ce Gain correspond à la combinaison 7 1 7 avec un rapport de gain de 25.

3. Conception du formulaire de paramétrage

3.1 Instanciation de la fenêtre de paramétrage

L'instanciation du formulaire de paramétrage est réalisée dans le constructeur du formulaire de jeu, afin qu'il apparaisse au démarrage de l'application. La destruction est réalisée automatiquement à la sortie du formulaire de jeu et est simplement caché lors de la partie.

```
this->monFormParam = (gcnew FormParam());
```

Cet objet `FormParam` doit être d'un droit d'accès `private`. De plus, il faut rajouter le fichier .h au début du script `Form1.h` :

```
#include "FormParam.h"
```

Le formulaire de paramétrage est déclaré dans les variables nécessaires au concepteur de la manière suivante :

```
private: FormParam^ monFormParam;
```

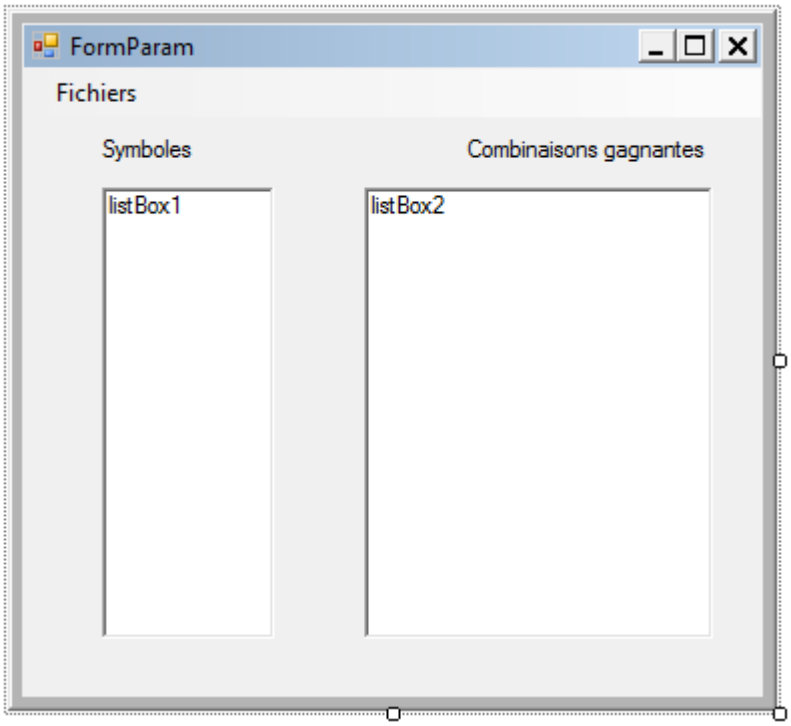
3.2 Configuration de la fenêtre de paramétrage

L'utilisateur doit pouvoir changer les paramètres de la machine à sous via une interface graphique. On génère donc une autre fenêtre `FormParam.h` contenant ce paramétrage. Dans `#pragma region Windows Form Designer generated code` du fichier `FormParam.h` on créé l'interface graphique.

Le code est généré automatiquement lorsque nous mettons en place les différents éléments de l'interface à partir de `FormParam.h[Design]`.

Cette fenêtre `FormParam.h` contient les éléments suivants :

- ✓ Un `menuStrip` permettant de choisir et modifier les symboles ou les combinaisons
- ✓ Un `openFileDialog` qui est une fenêtre qui s'ouvre après sélection du `menuStrip` et permettant de sélectionner un fichier `.smb` ou `.cmb` et d'initialiser l'objet pointé par bandit1.
- ✓ Deux `listBoxs` affichant les combinaisons et symboles sélectionnés après initialisation



Avant tout, on a inclut les fichiers .h au début du script `FormParam.h` pour pouvoir utiliser les différentes méthodes et les objets des classes `BanditIHM` et `Combinaison` dans des évènements :

```
#include"../ClassesTP/BanditIHM.h"
#include"../ClassesTP/Combinaison.h"
```

On utilise par la suite deux pointeurs de `BanditIHM` dans `Form1.h` et `FormParam.h` afin de faire le pont entre les deux formulaires. En effet, l'instanciation de notre bandit est réalisée une fois les symboles et combinaisons chargés correctement dans le formulaire de paramétrage lors de l'appui sur le bouton d'initialisation. Avant le paramétrage, à l'aide de la méthode `getBandit()` dans `FormParam.h`, on fait pointer les pointeurs sur la même adresse afin de récupérer l'instance du `BanditIHM`.

```
Form1(void)
        {
            InitializeComponent();
            //
            //TODO : ajoutez ici le code du constructeur
            cagnotte = 0;
            mise = 0;
            bandit1 = new BanditIHM("!","!");
            this->monFormParam = (gcnew FormParam());
            monFormParam->getBandit(bandit1);
            this->monFormParam->ShowDialog();
            //
        }
```

- `getBandit()` est une méthode `public` qui se trouve dans `FormParam.h`

```
public:  void getBandit(BanditIHM* bandit){
                bandit2 = bandit;
         }
```

La fenêtre de paramétrage ne peut être quittée uniquement si le `BanditIHM` a été correctement initialisé.

4. Problèmes rencontrés

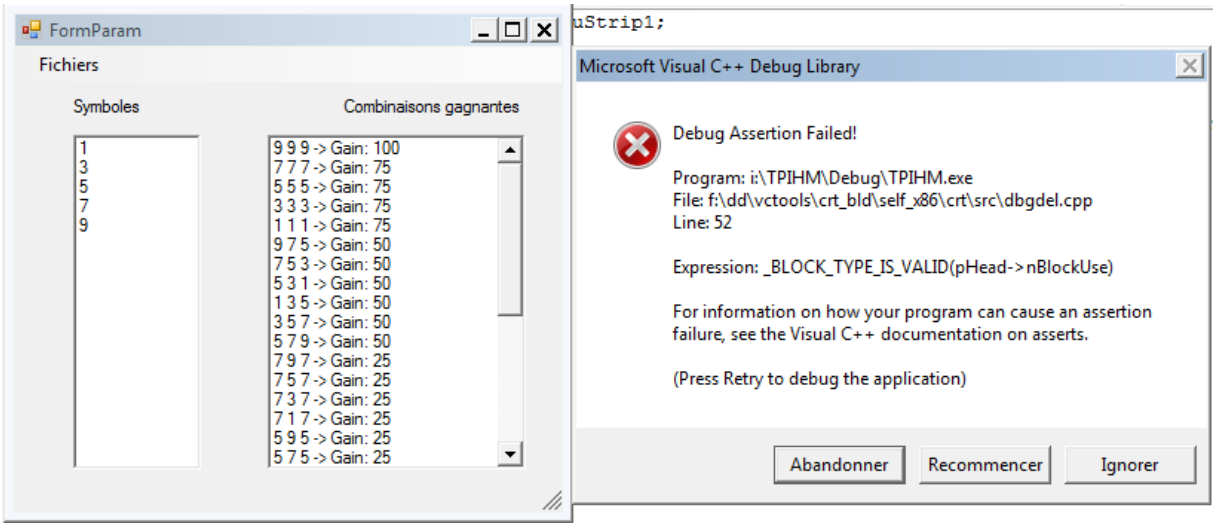
Nous sommes arrivés au point où l'application fonctionne sous certaines conditions, mais rencontre des bugs.

4.1 Premier Bug

Lors du premier paramétrage, lorsqu'on choisit un seul fichier sur les deux, et qu'on lance l'initialisation, l'application s'arrête et affiche un message d'erreur. Nous pensons qu'il faut prendre en compte les mauvaises manipulations de l'utilisateur et veiller à ce que l'application soit robuste vis à vis des erreurs d'initialisation. Il est donc important de s'assurer que l'initialisation peut être lancée uniquement si les deux fichier sont correctement chargés.

4.2 Deuxième Bug

Après le succès du premier paramétrage, l'utilisateur peut jouer sans soucis, mais lorsqu'il souhaite changer les paramètres, l'initialisation des nouveaux paramètres est impossible et affiche le message d'erreur suivant:



Nous pensons qu'il y a des défauts sur la gestion des tests des différents chargements de fichiers. Une solution pourrait être de réaliser une nouvelle initialisation des variables de tests lorsque le formulaire de paramétrage passe de l'état caché à l'état affiché.

5. Conclusion

Ce TP nous a permis de découvrir un environnement de développement permettant de réaliser des applications graphiques utiles aux interactions qu'il peut exister entre une application et son utilisateur. De plus, nous avons mis en application les classes que nous avions créées lors des TPs de C++.