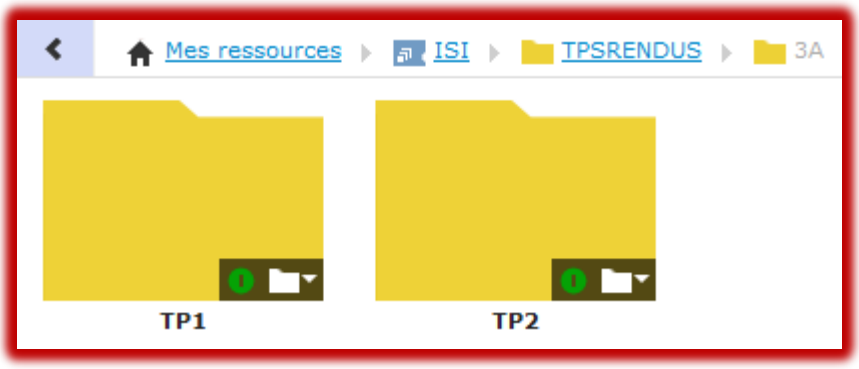
Memory

Le travail peut être réalisé en binôme, il est déposé dans le répertoire sous Spiral



Attention, vous ne déposez que le répertoire de votre solution avec tous les fichiers. Si vous voulez apporter une modification, vous pouvez remplacer votre solution.
Ne touchez pas aux solutions de vos camarades….

Objectifs

- Organiser les contrôles sur un formulaire
- Utiliser un réservoir d'images dans un formulaire
- Utiliser les images dans un formulaire

Conventions

pictogramme indiquant qu'il faut réaliser une action

Objectifs

On souhaite réaliser un jeu de Memory qui distribue aléatoirement des "cartes" sur un "tapis de jeu". Le joueur visualise quelques secondes les cartes. Ensuite il doit retrouver la carte affichée alors que toutes les cartes du tapis sont retournées.

Développement

Mise en place du tapis de jeu

Pour organiser les cartes sur le formulaire, nous nous proposons d'utiliser un conteneur appelé TableLayoutPanel.

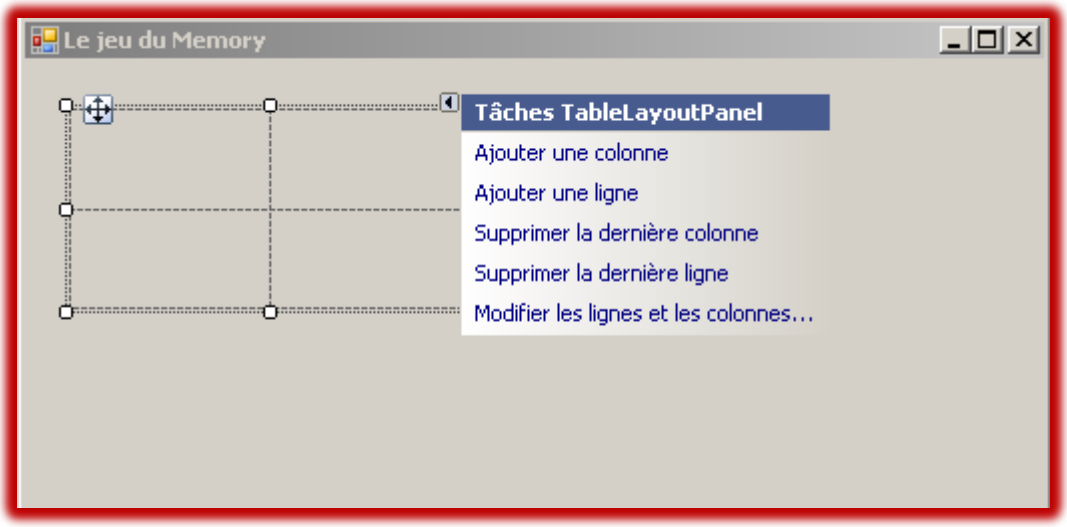
Présentation rapide du contrôle.

La fonction principale de ce contrôle est de permettre de conserver un aspect correct lorsque le contrôle est redimensionné (en conception) ou encore lorsque le formulaire est redimensionnée (en mode exécution).
Dans un premier temps, nous allons plutôt nous en servir pour organiser notre "tapis de jeu", sans nous préoccuper du redimensionnement pour l'instant.
Ce conteneur organise son contenu sous forme d'une grille, similaire à un tableau HTML. Dans chaque cellule peut être placé un seul contrôle.
Pour notre projet, il n'y aura qu'un seul type de contrôle (des images), mais il peut y avoir plusieurs sortes de contrôles. Par exemple : une rangée de TextBox, au-dessous, une rangée de boutons, puis soit une image, soit un label dans les cellules d'une troisième rangée.

Mise en place

Ce contrôle se trouve dans la partie "Conteneurs" de la boîte à outils.

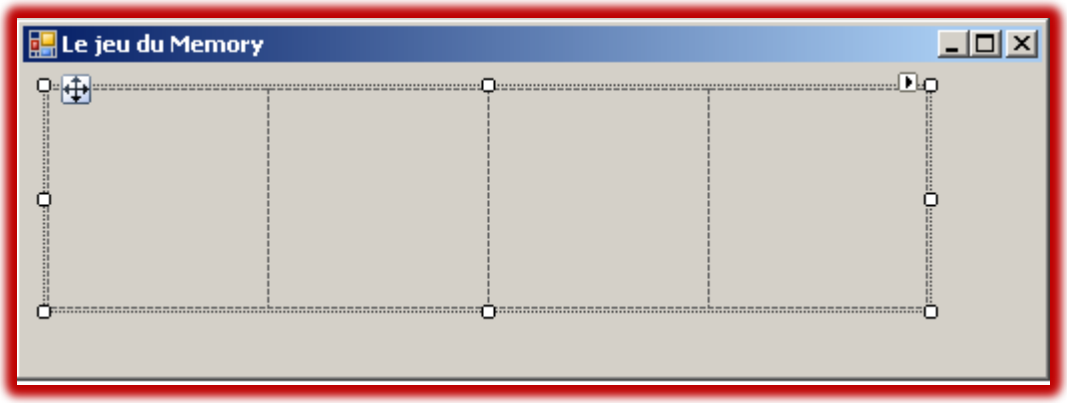
⌨ Déposer un contrôle TableLayoutPanel sur le formulaire et le nommer "tlpTapisDeCartes"



Par défaut, le contrôle est une grille de dimension 2 x 2.

Supprimer la dernière ligne, puis cliquer sur "Modifier les lignes et les colonnes" pour ajouter 2 colonnes et faire en sorte que chaque colonne ait une largeur correspondant à un pourcentage de 25% de la largeur totale,
Modifier ensuite sa taille dans les propriétés du contrôle : 440 en largeur et 110 en hauteur

Votre contrôle doit ensuite ressembler à ceci :



Insérer ensuite dans chaque case un contrôle permettant de réceptionner une image. Il s'agit du contrôle PictureBox.

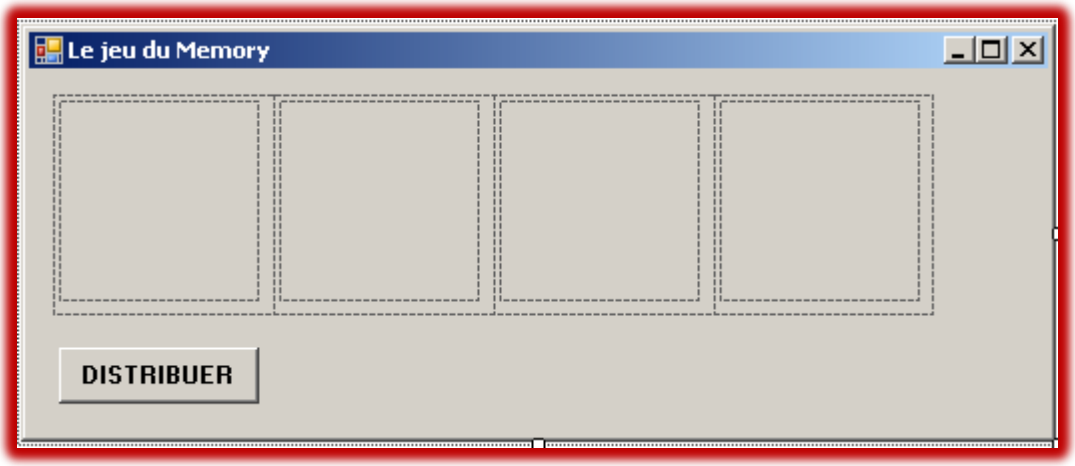
Conseil pour aller vite :

Déposer un premier contrôle PictureBox dans la première case
Modifier sa hauteur en "100" (propriété size/height) pour obtenir un carré.
Tout en maintenant la touche "CTRL" enfoncée, cliquer/glisser ensuite ce contrôle sur la 2ème case, puis la 3ème, puis la 4ème. C'est la façon la plus simple de dupliquer un contrôle.

NB : Noter que l'on ne peut pas mettre 2 contrôles dans la même case ; ce n'est pas parce qu'il n'y a pas assez de place, mais parce que le conteneur TableLayoutPanel est prévu pour héberger un contrôle par case.

Renommer ensuite les contrôles PictureBox comme suit : pb_01, pb_02, pb_03 et pb_04,
Ajouter un bouton que vous nommerez "btn_Distribuer". Attention, faites attention à l'ordre dans lequel vous déposez vos PictureBox

Vous devriez obtenir ceci :

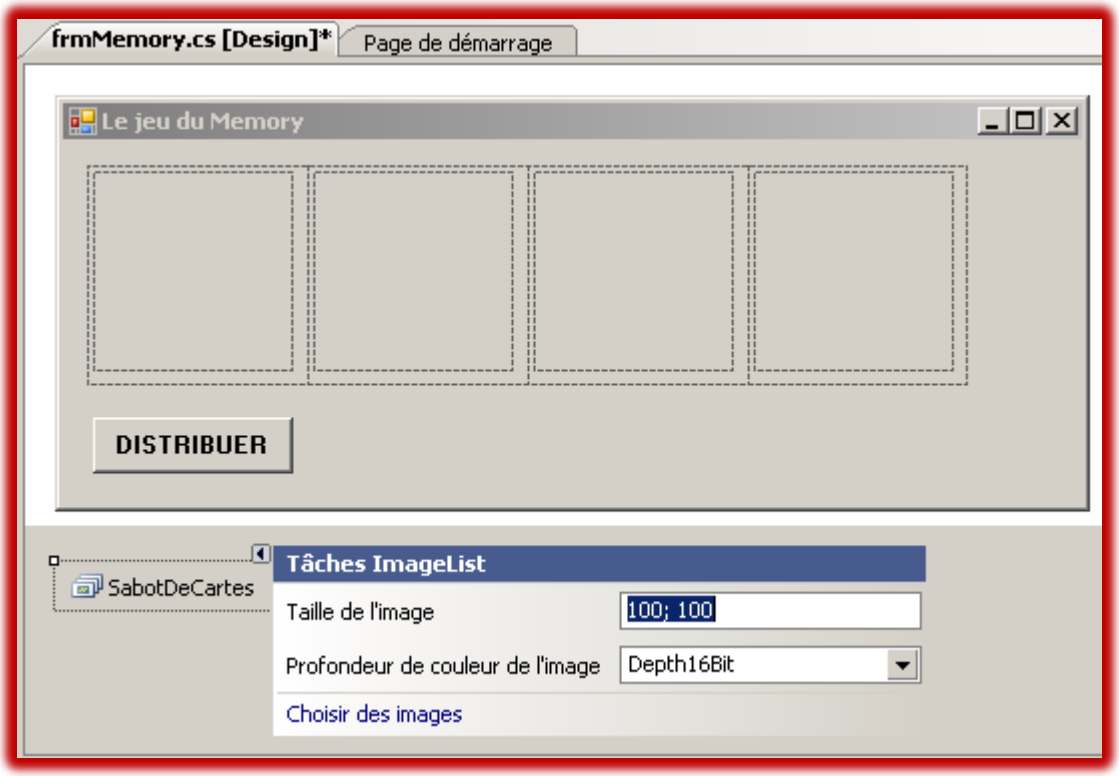


Mise en place du réservoir d'images

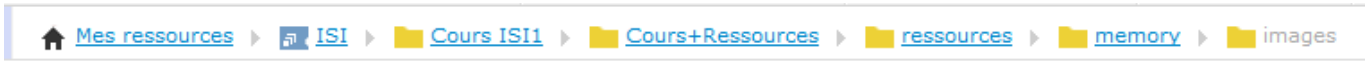
Le contrôle ImageList va nous permettre de constituer un réservoir d'images, et donc d'alimenter ensuite les contrôles PictureBox. Ce contrôle se trouve dans la partie "Composants" de la boîte à outils.

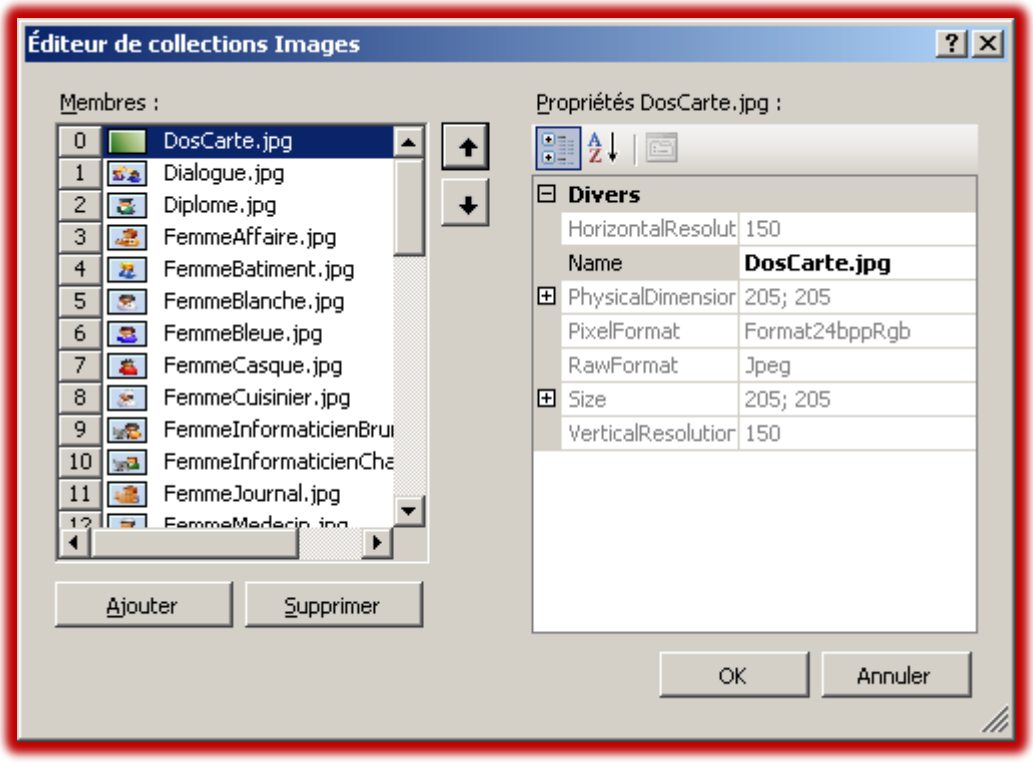
Déposer un contrôle de type ImageList et le nommer ilSabotDeCartes.

Notons que ce composant, comme d'autres que nous avons utilisé (Timer, OpenFileDialog ...) ne se positionne pas sur le formulaire. En effet il n'est pas visible à l'exécution. Visual Studio le place donc en mode conception dans la zone réservée à ce type de contrôles, au-dessous du formulaire.
Ce composant permet de stocker dans l'application les images qui seront utilisées ; attention au fait que s'il contient beaucoup d'images au lancement de l'application, la taille de l'exécutable sera conséquente.



Modifier ses propriétés : Taille de l'image 100 x 100, Profondeur : 16 bits,
Puis cliquer sur Choisir les images, Ajouter les 40 images que vous trouverez dans :





Veillez à ce que l'image DosCarte.jpg soit la première des images, à l'indice 0.

Distribution des cartes

Occupons-nous maintenant de la distribution des cartes. Nous commençons par une distribution simple : on distribue les 4 premières images du "sabot" (les images d'indice 1 à 4 du réservoir) sur notre "tapis de jeu".
Taper la procédure suivante et appelez-la dans la procédure événementielle associée au clic du bouton "Distribuer".

```
private void Distribution_Sequentielle()
{
    PictureBox carte;
    int i_carte=1;

    foreach (Control ctrl in tlpTapisDeCartes.Controls)
    {
        // Je sais que le contrôle est une PictureBox
        // donc je "caste" l'objet (le Contrôle) en PictureBox...
        carte = (PictureBox)ctrl;
        // Ensuite je peux accéder à la propriété Image
        // (je ne pourrais pas si je n'avais pas "casté" le contrôle)
        carte.Image = ilSabotDeCartes.Images[i_carte];
        i_carte++;
    }
}
```

Remarque : Le "CAST" a pour objectif de forcer à considérer un objet comme étant d'un certain type, sans quoi on ne peut pas accéder aux propriétés & méthodes qui lui sont spécifiques. Le "CAST" est une conversion abrupte, à l'emporte-pièce...
ATTENTION ! Toutefois, si l'objet n'est pas réellement du type attendu, une exception est levée et le programme plante si elle n'est pas gérée !

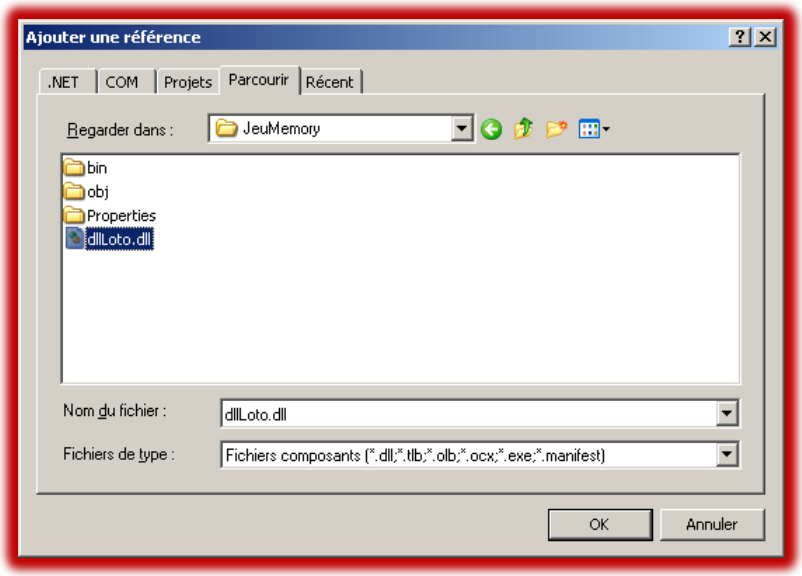
Utilisation de la "LotoMachine"

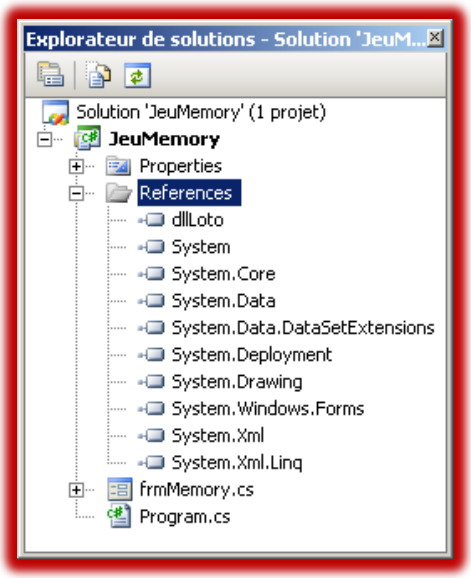
Pour vous faciliter la vie, une dll (autrement dit une classe) est mise à votre disposition pour vous permettre de générer facilement une suite de nombres aléatoire. Cet outil permettra par exemple de sélectionner aléatoirement 4 images dans le réservoir pour les afficher.
On aura alors une simulation de distribution aléatoire des cartes
Remarque : Vous pourrez par la suite écrire votre propre DLL en vous inspirant de l'interface de celle-ci.
Sous Windows, copier la DLL "dllLoto" de :

Mes ressources ▸ ISI ▸ Cours ISI1 ▸ Cours+Ressources ▸ ressources ▸ memory ▸ images ▸ DLL

vers le dossier Bin du projet.
Ensuite il faut la référencer en ajoutant une référence au projet :

Clic droit sur References dans l'explorateur de solution / Ajouter une référence,
Dans l'onglet Parcourir, sélectionner la dll, puis OK,
Vérifier qu'elle apparaît ensuite dans les références.





Il faut ensuite ajouter la clause using adéquate pour pouvoir utiliser facilement la classe "LotoMachine" que contient la dll "dllLoto".

Ajouter la clause en haut de la page de code associée au formulaire :

```
...
using System.Windows.Forms;
using dllLoto;
```

La DLL "dllLoto" contient la classe "LotoMachine". Que contient cette classe ?

- Un constructeur qui admet en paramètre la valeur maximale qui peut être générée par la machine.
- Une méthode TirageAleatoire(nbNumeros, doublons) qui renvoie un tableau de nbNumeros nombres aléatoires, compris entre 1 et la valeur maximale définie à l'instanciation, avec possibilité de doublons ou non (suivant la valeur du booléen "doublons").
- Une méthode NumeroAleatoire() qui renvoie au hasard l'une des valeurs de la série générée au moment de l'appel de Tirage Aleatoire.
  Cette méthode ne peut donc être appelée avant d'avoir invoqué TirageAleatoire(). Si par erreur elle est appelé mal à propos, une exception "Le tirage n'a pas encore été effectué." est levée.
- Des accesseurs, mais que vous n'utiliserez pas à priori...
-

Test de la LotoMachine

Avant d'utiliser la "LotoMachine" pour distribuer les cartes, comprenons son fonctionnement en affichant simplement la liste de nombre générés par cette machine hasardeuse.

Ajouter sur votre formulaire provisoirement un bouton "btn_Test" dont le libellé sera "TEST LotoMachine", Ajouter le code événementiel suivant :

```
private void btn_Test_Click(object sender, EventArgs e)
{
    // On utilise la LotoMachine pour générer une série aléatoire
    // On fixe à 49 le nombre maxi que retourne la machine
    LotoMachine hasard = new LotoMachine(49);
    // On veut une série de 6 numéros distincts parmi 49 (comme quand on joue au
loto)
     int[] tirageLoto = hasard.TirageAleatoire(6, false);
     // false veut dire pas de doublon : une fois qu'une boule est sortie,
     // elle ne peut pas sortir à nouveau ;-)
     // La série d'entiers retournée par la LotoMachine correspond au loto
     // affiché sur votre écran TV ce soir...
    string grilleLoto="* ";
    for (int i = 1; i <= 6; i++)
    {
        grilleLoto = grilleLoto + tirageLoto[i] + " * ";
    }
    MessageBox.Show(grilleLoto,"Tirage du LOTO ce jour !");
}
```

Tester le bon fonctionnement de ce code de test de la classe "LotoMachine".

Distribution aléatoire des cartes (utilisation de la LotoMachine)

Modifier l'appel de procédure dans la procédure événementielle associée au bouton "btn_Distribuer" comme montré dans le code ci-dessous.

```
Distribution_Aleatoire();
```

Deux valeurs sont importantes dans les traitements que nous allons faire maintenant :

- Le nombre d'images dans le "Sabot"
- Le nombre de cartes sur le "Tapis"

Pour une meilleure évolution de notre application, nous allons les déclarer globalement au formulaire, de manière à pouvoir les mettre à jour, si nécessaire, n'importe où dans le code.

Déclarer les variables (des entiers) "nbCartesDansSabot" et "nbCartesSurTapis" tout au début du code de la classe "frmMemory" :

```
public partial class frmMemory : Form
{
    // Déclaration des variables globales du jeu
    int nbCartesDansSabot;  // Nombre de cartes dans le sabot (en fait nombre
                            // d'images dans le réservoir)
    int nbCartesSurTapis;   // Nombre de cartes sur le tapis
```

Coder la procédure Distribution_Aleatoire() comme suit :

```
private void Distribution_Aleatoire()
{
    // On utilise la LotoMachine pour générer une série aléatoire
    LotoMachine hasard = new LotoMachine(nbCartesDansSabot);
    // On veut une série de nbCartesSurTapis cartes parmi celles
    // du réservoir
    int[] tImagesCartes = hasard.TirageAleatoire(nbCartesSurTapis,false);
```

```
    // La série d'entiers retournée par la LotoMachine correspondra
    // aux indices des cartes dans le "sabot"

    // Affectation des images aux picturebox
    PictureBox carte;
    int i_image;
    for (int i_carte = 0; i_carte < nbCartesSurTapis; i_carte++)
    {
        carte = (PictureBox)tlpTapisDeCartes.Controls[i_carte];
        i_image = tImagesCartes[i_carte+1]; // i_carte + 1 à cause
       // des pbs d'indices
        carte.Image = ilSabotDeCartes.Images[i_image];
    }
}
```

Il faut s'occuper d'initialiser les variables nbCartesSurTapis et nbCartesDansSabot. Cela peut se faire lorsque l'on souhaite distribuer les cartes, avant d'appeler la procédure Distribution_Aleatoire().

⌨ Modifier le code de la procédure événementielle.

```
private void btn_Distribuer_Click(object sender, EventArgs e)
{
    // On récupère le nombre d'images dans le réservoir :
    nbCartesDansSabot = ilSabotDeCartes.Images.Count - 1;
    // On enlève 1 car :
    // -> la l'image 0 ne compte pas c'est l'image du dos de carte
    // -> les indices vont de 0 à N-1, donc les indices vont jusqu'à 39
    //    s'il y a 40 images au total *

    // On récupère également le nombre de cartes à distribuées sur la tapis
    // autrement dit le nombre de contrôles présents sur le conteneur
    nbCartesSurTapis = tlpTapisDeCartes.Controls.Count;

    // On effectue la distribution (aléatoire) proprement dite
    Distribution_Aleatoire();
}
```

* Vous pouvez vérifier les indices dans l'éditeur d'images du contrôle "ilSabotDeCartes", de type ImageList.

⌨ Tester le bon fonctionnement de la distribution aléatoire des cartes en cliquant plusieurs fois sur le bouton "DISTRIBUER".

⌨ Ajouter une procédure événementielle qui permet de connaître la carte choisie.

```
private void pb_XX_Click(object sender, EventArgs e)
        {
            PictureBox carte;
            int i_carte, i_image;
            //if (Image_1 == null)
            //    MessageBox.Show("L'image 1 n'est pas référencée");
            //if (Image_2 == null)
            //    MessageBox.Show("L'image 2 n'est pas référencée");

            if (nb_cartes < 2)
            {
                carte = (PictureBox)sender;
                i_carte = Convert.ToInt32(carte.Tag);
                i_image = tapisCARTES[i_carte];
                carte.Image = imgListe.Images[i_image];
                if (i_image == i_hasard)
                {
                    MessageBox.Show("Bravo !");
                }
                else
                {
```

```
                MessageBox.Show("DOMMAGE !");
            }
            if (nb_cartes == 0)
            {
                Image_1 = carte;
            }
            if (nb_cartes == 1)
            {
                Image_2 = carte;
            }
            nb_cartes++;

        }
        else
        {
            MessageBox.Show("Deux cartes sont déjà retournées !");
            RetournerLesCartes();
            nb_cartes = 0;
            Image_1 = null;
            Image_2 = null;
        }

    }
```

TP

Partie à traiter

Ajouter un bouton "btn_Retourner" permettant de "retourner" toutes les cartes. Il s'agit en fait de modifier l'image de toutes les cartes sur le tapis en la remplaçant par l'image DosCarte.

Ajouter une fonction de jeu proprement dite :
Ajouter un contrôle PictureBox en dehors du tapis ; vous le nommerez "pb_Recherche".
Ajouter un bouton "btn_Jouer" qui :

Retourne les cartes
Affiche une des 4 images non visibles (dans le contrôle "pb_Recherche") pour que l'utilisateur désigne la carte identique sur le tapis. (utiliser la méthode NumeroAleatoire() de la classe "LotoMachine".
Ajout d'un Timer pour gérer le délai de choix des cartes

Ecrire une procédure événementielle (une seule commune aux 4 cartes sur le tapis), qui sur clic, la retourne et affiche le MessageBox "BRAVO" si l'image est bien la même que celle recherchée, "DOMMAGE" dans le cas contraire.

Transformer votre formulaire pour qu'il y ait 8 cartes sur le tapis au lieu de 4. Si vous avez bien développé, l'incidence sur le code devrait être très limitée !

Partie à réaliser

On peut imaginer un vrai jeu de memory, dans lequel l'objectif consiste à retrouver les paires (deux cartes identiques) avec le moins de coups possible.

- A chaque coup l'utilisateur retourne 2 cartes.
- Si les images sont identiques, elles restent face visible.
- Si les images sont différentes, elles sont de nouveau retournées, face cachée, soit après une temporisation, soit quand le joueur décide de les retourner.