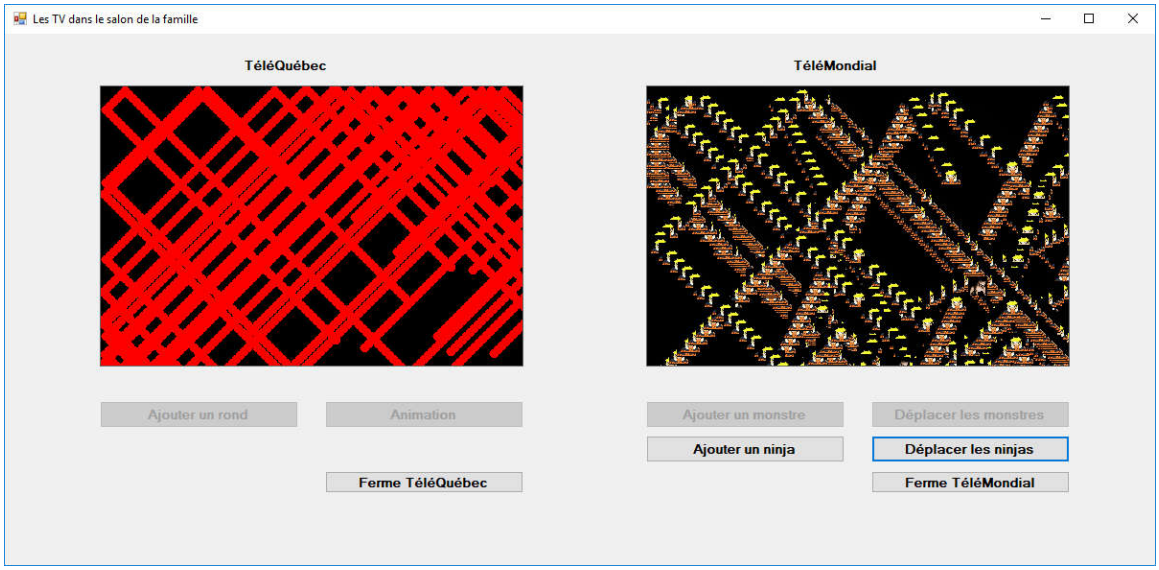
TP3b : Héritage de classes (7%) v. beta

D I R E C T I V E S :

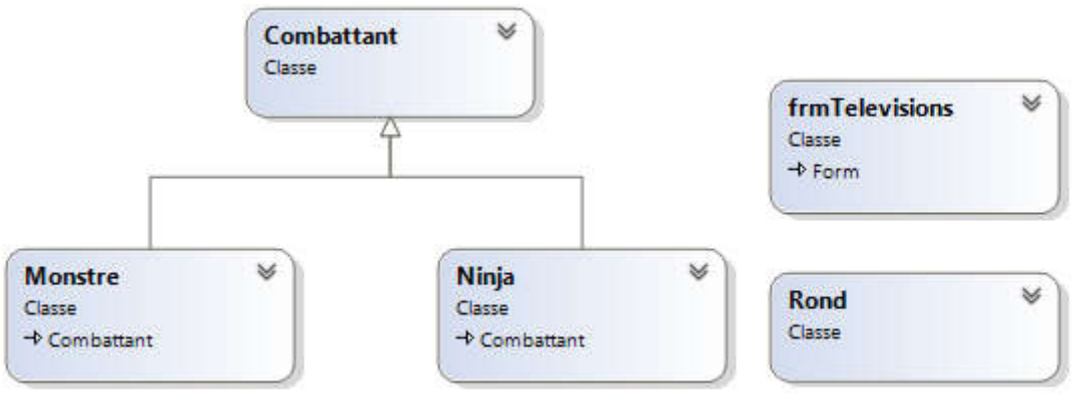
Prenez le projet zippé ***PlusieursTV.zip*** *(à venir) et complétez-le*, en suivant les consignes suivantes, pour obtenir une exécution semblable à ***PlusieursTV*****.exe** (à venir)**. Peut être fait en équipe.**

Visuel du projet :



Les **listes** pour emmagasiner les **monstres et ninjas** sont en erreurs de compilation, il faut créer les **classes** avant pour compiler et les utiliser.

1- **Créer** la hiérarchie de classes **public** suivantes : (.5 point)



y est

2- Ajouter les membres manquant dans les classes de la hiérarchie : (.75)

Rond : // elles y sont (vous pouvez n'en mettre plus dépendant de votre logique)
`Propriétés public: Diametre, Couleur, Vitesse, PositionX, PositionY`

Combattant : (vous pouvez n'en mettre plus dépendant de votre logique)
`Propriétés public:: Force, Avatar, Vitesse, PositionX, PositionY`

Monstre : (vous pouvez n'en mettre plus dépendant de votre logique)
`Propriétés public:: Planete`

Ninja : (vous pouvez n'en mettre plus dépendant de votre logique)
`Propriétés public:: Nom`

Note : toutes les propriétée sont en lecture et écriture

3- Créer des constructeurs dans les classes: (.75 point)

Rond : en surcharge, initialisant ses **membres** et ses **membres hérités** accessibles
Combattant : aucun
Monstre : initialisant ses **membres** et ses **membres hérités** accessibles
Ninja: initialisant ses **membres** et ses **membres hérités** accessibles
(ces constructeurs peuvent initialiser d'autres propriétés ou membres si vous en ajoutez)

4- Coder les 3 méthodes *public void* Bouge() des classes Rond, Monstre et Ninja qui seront délégués aux threads via Threadstart (1 point)

5- Compléter en codant les boutons et timers du projet (4 points)

☺

Remettre, le **projet zippé** à votre ***nom*** par **mio** ou **git,** pour vendredi 29 octobre

Aide pour simplifier le TP3b

Retour sur l'utilisation des **listes** et les notions de programmation objet. Un **objet** à comme modèle une **classe** (qui est son type de donnée) et est instancié (devient existant) avec l'opérateur **new**. En jargon informatique, on dit qu'un **objet** est l'instance d'une classe. (ex : avoir une liste de ronds à la place d'un tableau)

Déclaration d'une liste

Ex : List<Rond> m_lstRond; // Rond étant la classe qui sera le modèle de tous les ronds

Initialisation d'une liste

Ex : m_lstRond = new List<Rond>(); // crée l'objet List qui contiendra tous les ronds

Ajouter un objet à un liste (avec la méthodes **Add**) :

Ex : rond = **new** Rond(/* param du constructeur */);
m_lstRonds.**Add**(rond);

Parcourir (avec l'instruction foreach)

Ex :
```
foreach(Rond rond in m_lstRonds)
{
        MessageBox.Show(rond.Couleur.Tostring()); // Couleur est une propriété
}
```

Explication pour la construction des 10 ronds dans un tableau de dimension 10

Nous aurions pu simplifier par étape notre code comme ceci :

```
Rond rond; //rond de type Rond
…
rond = new Rond(/* dépendant de votre constructeur */);
m_tRonds[m_compteurRond] = rond;
SolidBrush pinceau = new SolidBrush(rond.Couleur);
m_gTeleQuebec.FillEllipse(/* param */); //Dessine le rond dans le panel TéléQuébec
m_compteurRond++; //avance le compteur de rond(s)
```

Retour sur le dessin avec un objet Graphics,

Using `System.Drawing; // les classe Image et SolidBrush viennent de là`

Déclarer un objet Graphics

Ex : `Graphics m_gTeleQuebec;`

Initialiser un objet Graphics associé à un panel avec la méthode `CreateGraphics`

Ex : `m_gTeleQuebec = pnlTeleQuebec.CreateGraphics();`

Dessiner un rond plein avec la méthode `FillEllipse` d'un objet Graphics

```
public void FillEllipse (brush, int x, int y, int width, int height);
```

```
Ex :  SolidBrush pinceau = new SolidBrush(Color.Red);
      m_gTeleQuebec.FillEllipse(pinceau, 50, 60, 20, 20);
```

Dessiner une image plein avec la méthode `DrawImage` d'un objet Graphics

```
public void DrawImage (Image image, int x, int y, int width, int height);
```

```
Ex :   Image img = Image.Fromfile("ninja.png"); // propriété Avatar dans TP
       m_gTeleMondial.DrawImage(img, 10, 10, 20, 20);
```

Les **Timers** étant Enabled à false au début et les démarrage (***start***()) n'étant fait que dans les click des boutons, on peut **simplifier** le code des événements ***tick*** des **Timers.** Seulement le code encadré est nécessaire.

Le if était en erreur dans la version envoyée, il y avait « ! » de trop… dsl

```
            if (!m_TickAutoQC) // pour éviter le premier tick automatique
            {
                m_TickAutoQC = false;
            }
            else
            {
                for (int i = 0; i < m_compteurRond; i++)  // foreach si c'est une liste
                {
                    m_gTeleQuebec.FillEllipse(/* param basés sur des propriétés
                                                 d'un rond */);
                }
            }
```

Résumé de **thread** avec *code d'une classe* autre que le formulaire

Thread : tâche indépendante exécutée en parallèle au thread principal d'un programme.
Espace de noms : `using System.Threading;`

En C#, **Thread** de *System.Threading* est une **classe** que l'on peut **instancier** mais qui **ne peut être hériter** (classe scealed (scellée)).

Étape à suivre pour la création d'un thread **dans une classe** à l'extérieur du formulaire :

1- Créer une **méthode** qui sera la tâche déléguée à un objet Thread. Cette tâche pourra se servir de données passées en paramètres au constructeur et s'en servir via des membres. Code d'une classe recevant des données en paramètres pour les utiliser.

```
private void TâcheThread() // ne reçoit rien et ne retourne rien
{
      …
}
```

2- Lors de la création d'un objet habitant la méthode du thread, lui passer des données par les paramètres (il(s) seront reçus et placer dans des membres de l'objet et pourront être utiliser par la tâche)).

```
public class Auto
{
   public int Autonomie{ get; set; } // propriété membre

    public Ninja(int autonomie)
    {
        Autonomie = autonomie;
    }
    public void Roule() // ne reçoit rien mai peut utiliser des membres
    {
       int km = 0, usure = 0;
       while (km < Autonomie)
       {
            usure++;
            Thread.Sleep(20);
       }
    }
 }
```

Dans un formulaire

3- Créer un objet **ThreadStart** délégant la **méthode** à exécuter

```
Auto F150 = new Auto(450);
ThreadStart codeThread = new ThreadStart(F150.Roule);
```

4- Créer un objet **Thread** en lui spécifiant un objet **ThreadStart**.

```
Thread monThread = new Thread(codeThread);
```

5- Démarrer le thread par un événement du **formulaire** (ex : click d'un bouton)

```
monThread.start();      // exécutant Roule() de l'objet
```