



TD - Découverte de Visual Studio

Partie 1

Ressources Célène : [DI4.S8.PLATEFORMES .NET / « Lg.Net »](#)

Machine virtuel disponible : VMWARE Windows 10 - .Net - Nicolas DAGNAS

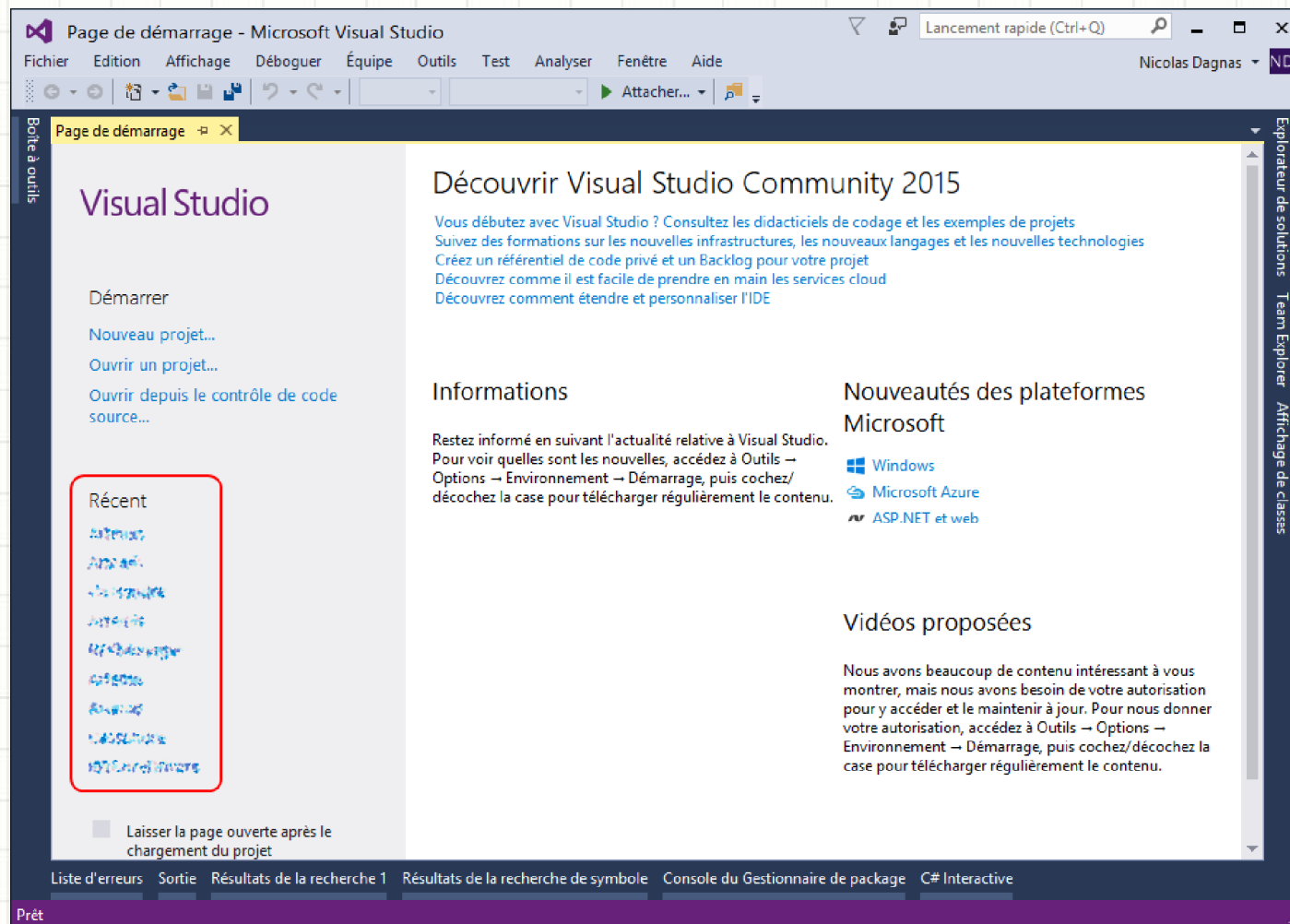
Intervenant : Nicolas DAGNAS

Pour commencer

- Le TD peut être effectué avec Microsoft Visual Studio de 2015 à 2019, voir les précédents (Visual Studio Code n'est pas abordé ici).
- Pour ceux n'ayant pas de Visual Studio à disposition, une machine virtuelle est disponible avec Visual Studio 2019 :
 - VMWARE Windows 10 - .Net - Nicolas DAGNAS
- Pour ceux qui l'installeront, faites une installation basique avec uniquement C# Desktop et sans la mobilité.
- Ce TD ayant pour but la découverte de l'environnement, n'hésitez pas à poser des questions si quelque chose n'est pas clair.

VS – Page de démarrage

- Premier démarrage de « Microsoft Visual Studio ».

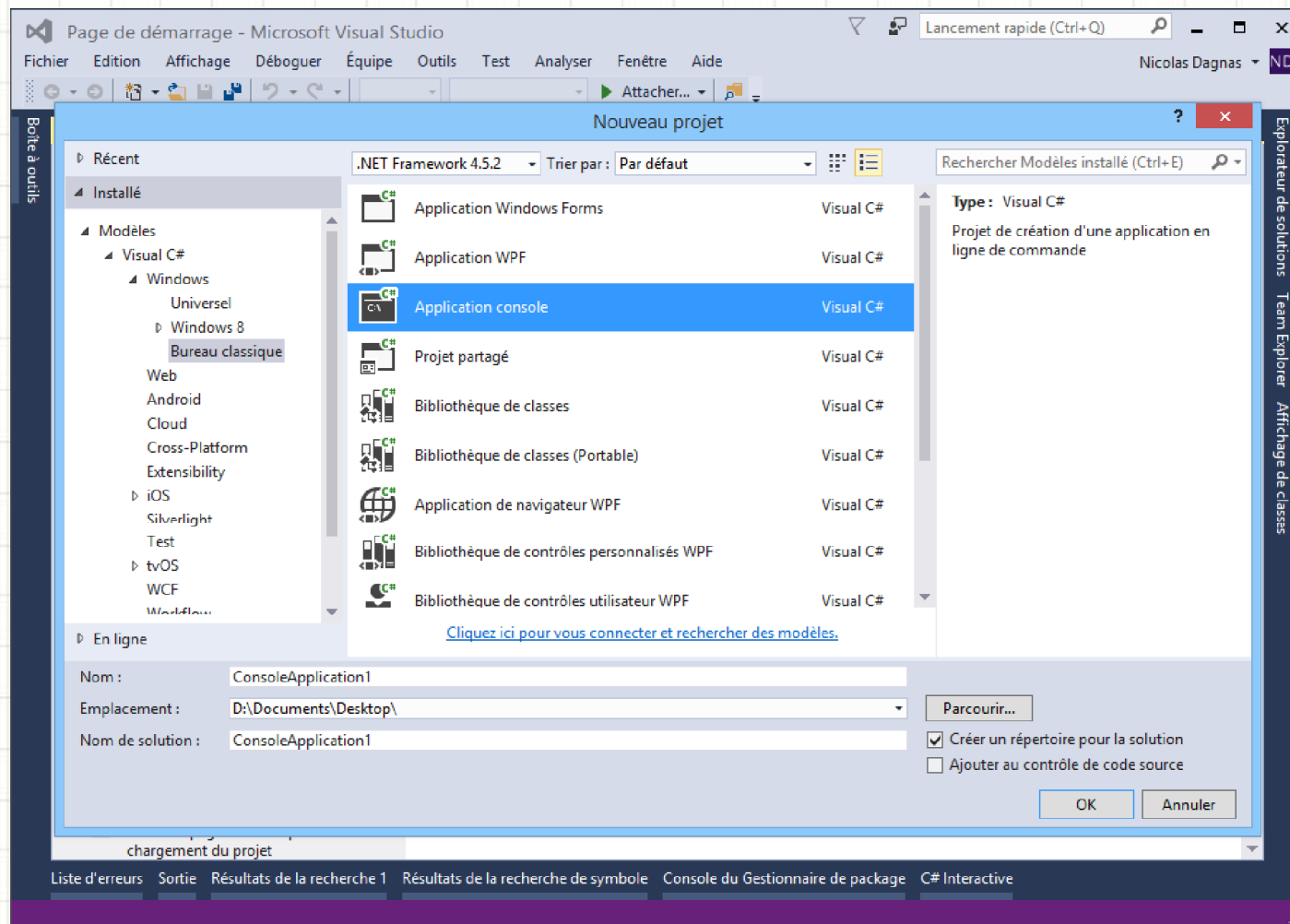


VS – Types de Projets

- Pour le TD, nous nous concentrerons sur ces deux types de projets :
 - Windows > Application console (.NET Framework)
 - Fenêtre DOS, interactions minimalistes avec l'utilisateur
 - Windows > Application Windows Form (.NET Framework)
 - Fenêtre Windows, conceptions d'écrans complexes type IHM

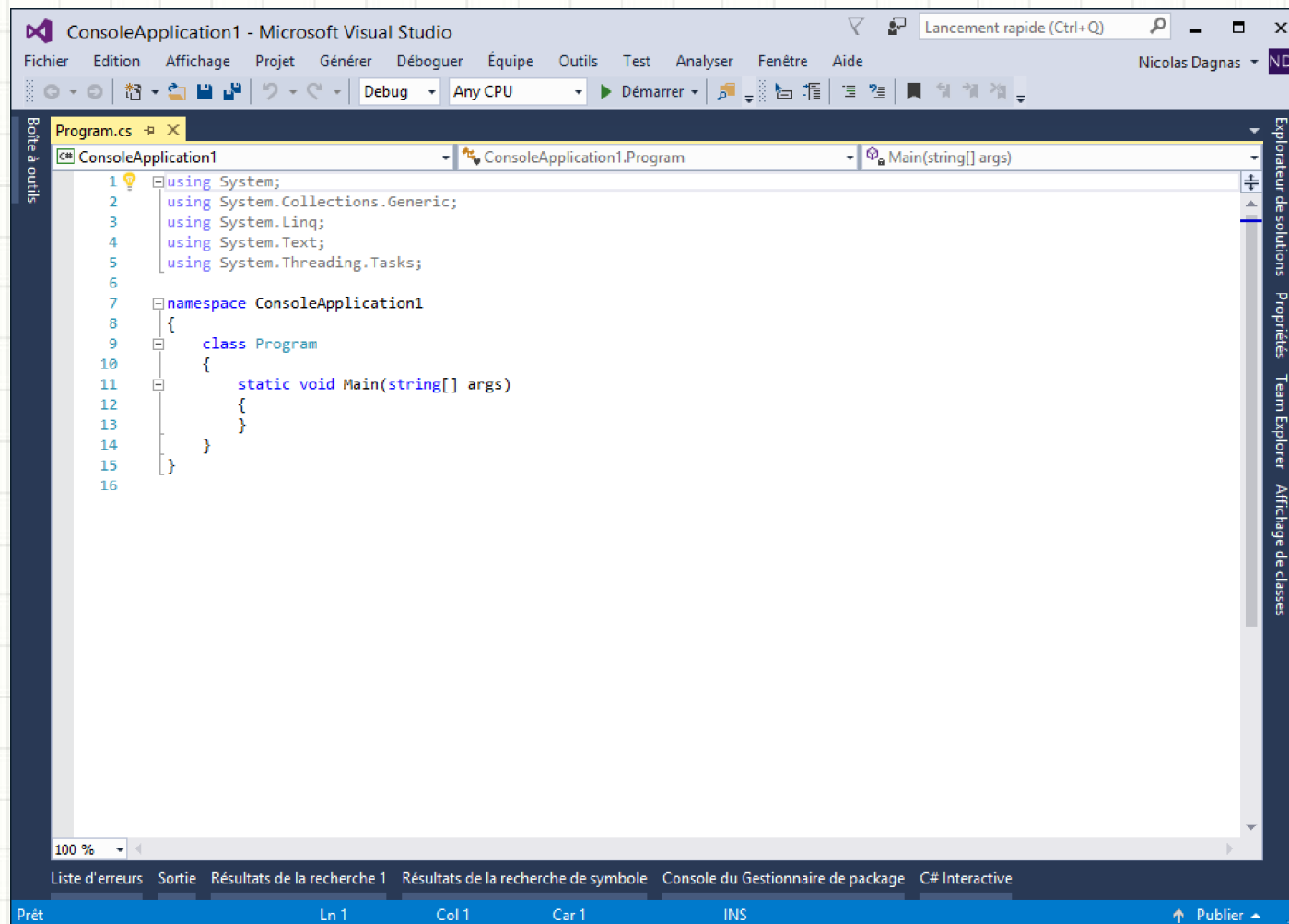
VS – Projet Console

- Nouveau projet « Application console (.NET Framework) » - « C# »



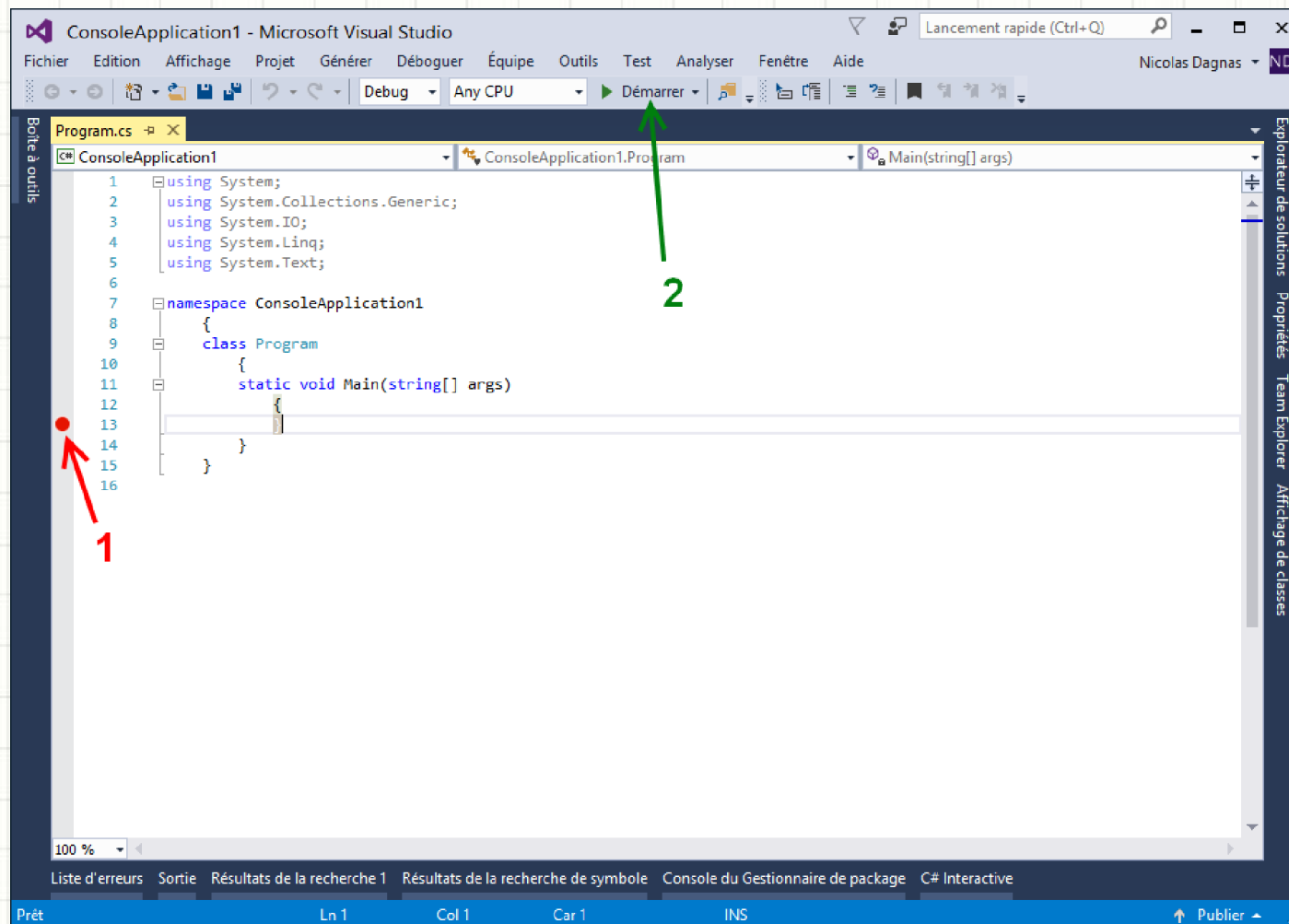
VS – Projet Console

- Mon nouveau projet.



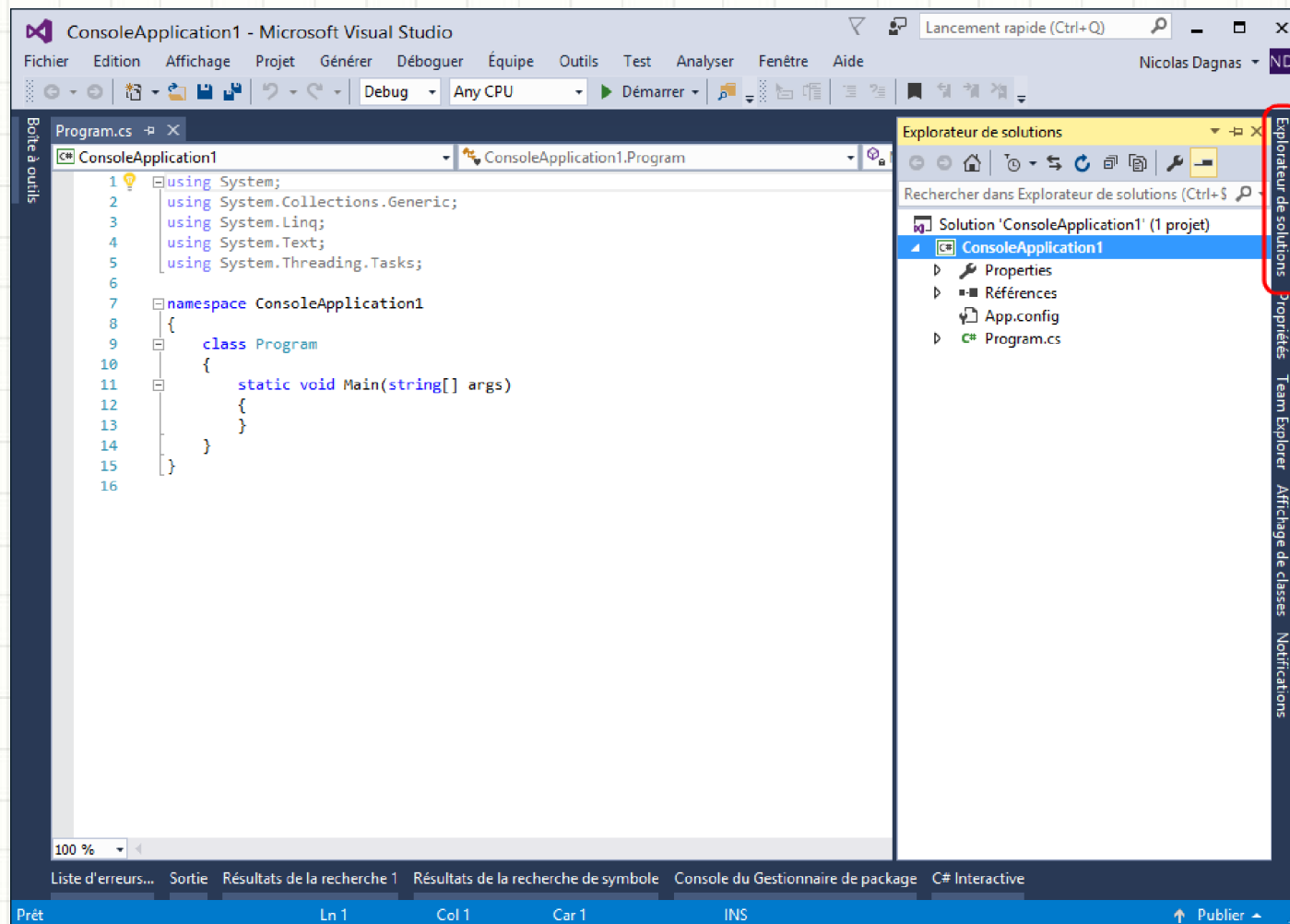
VS – Projet Console

- Un point d'arrêt et contenu de « args ».



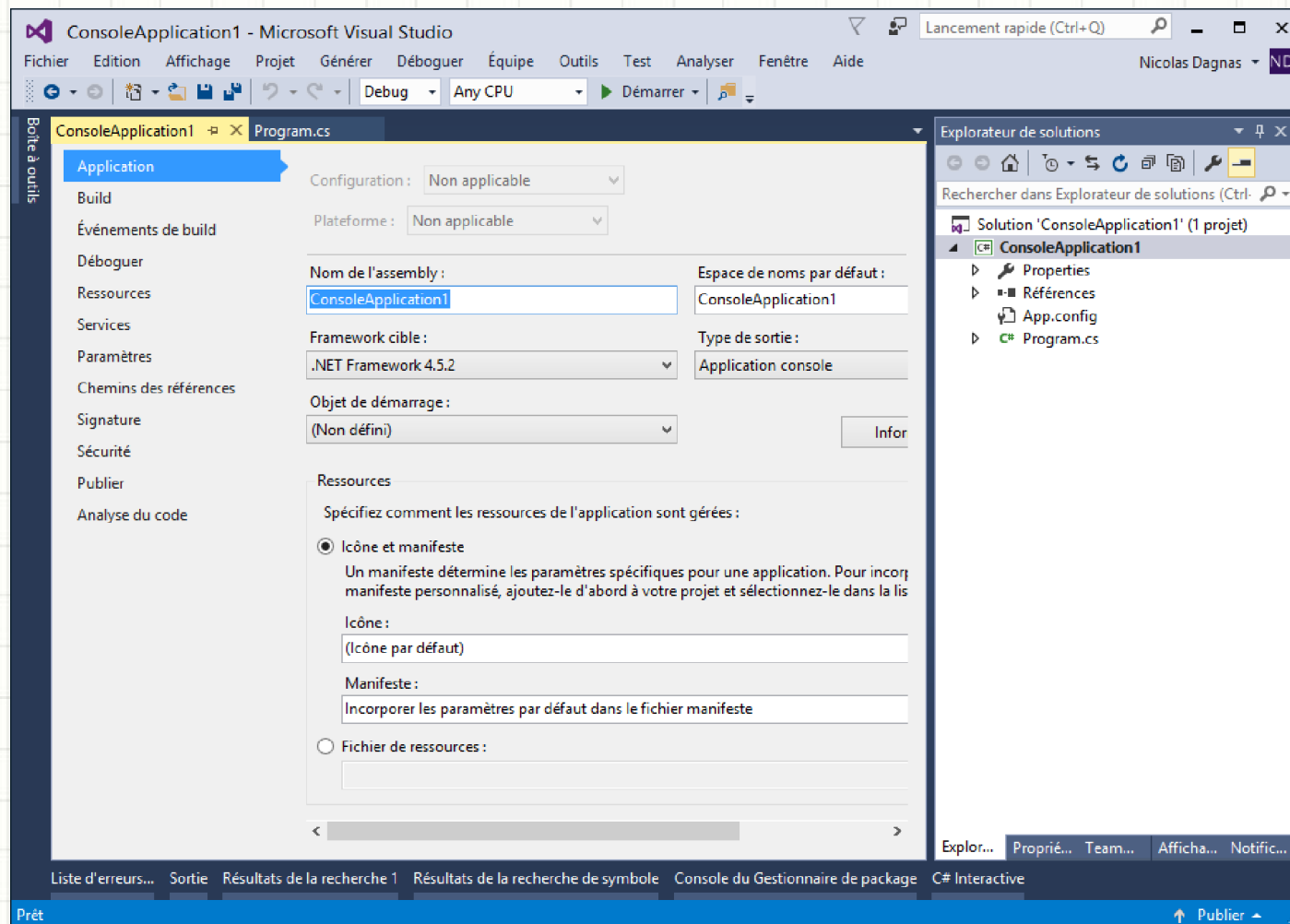
VS – Propriétés du projet

- Variable « args », comment la renseigner ?



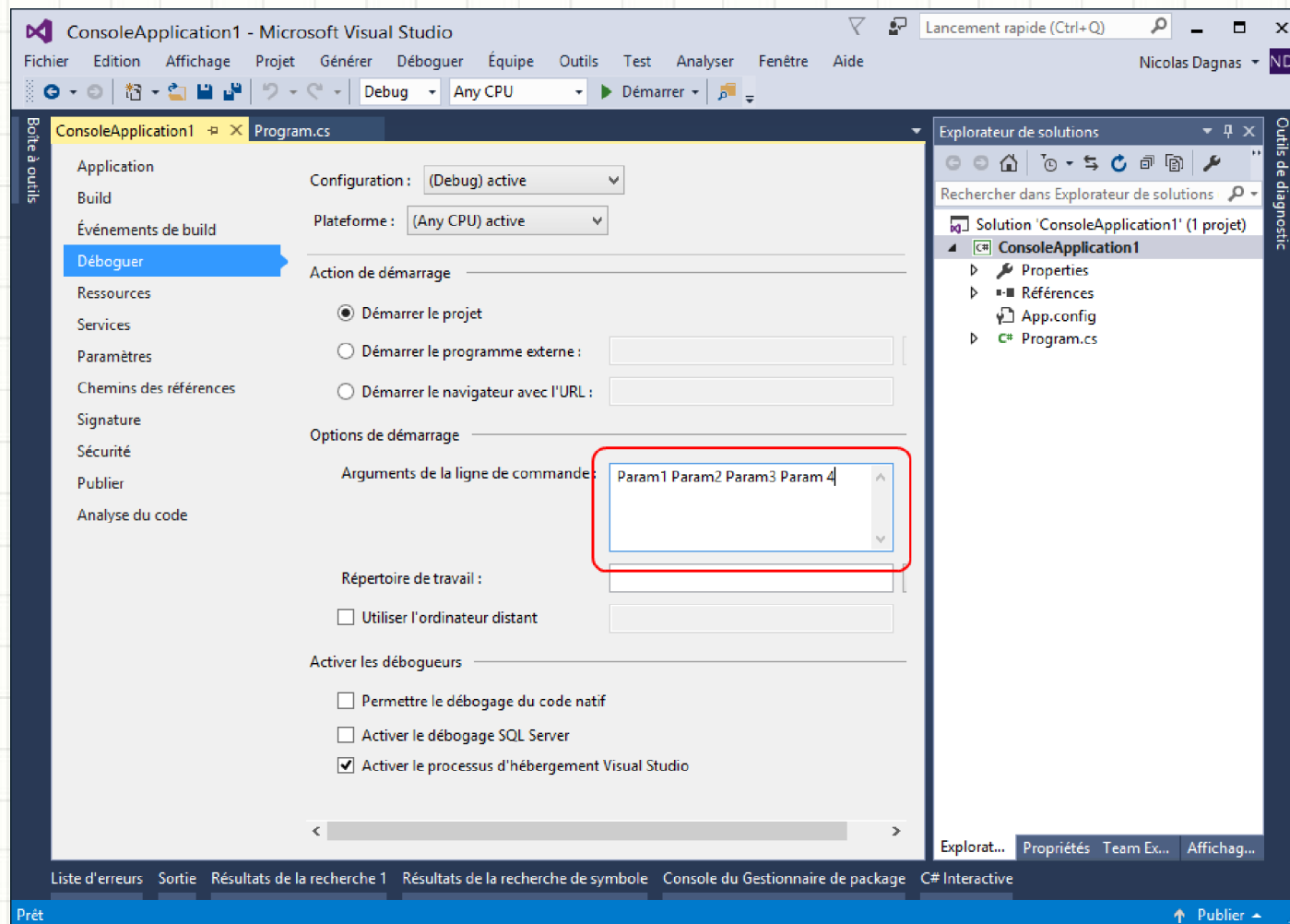
VS – Propriétés - Application

- Les différents onglets des propriétés de notre projet.



VS – Propriétés - Déboguer

- Arguments de la ligne de commande.



VS – Ligne de commande

- Contenue de la méthode « Main » :

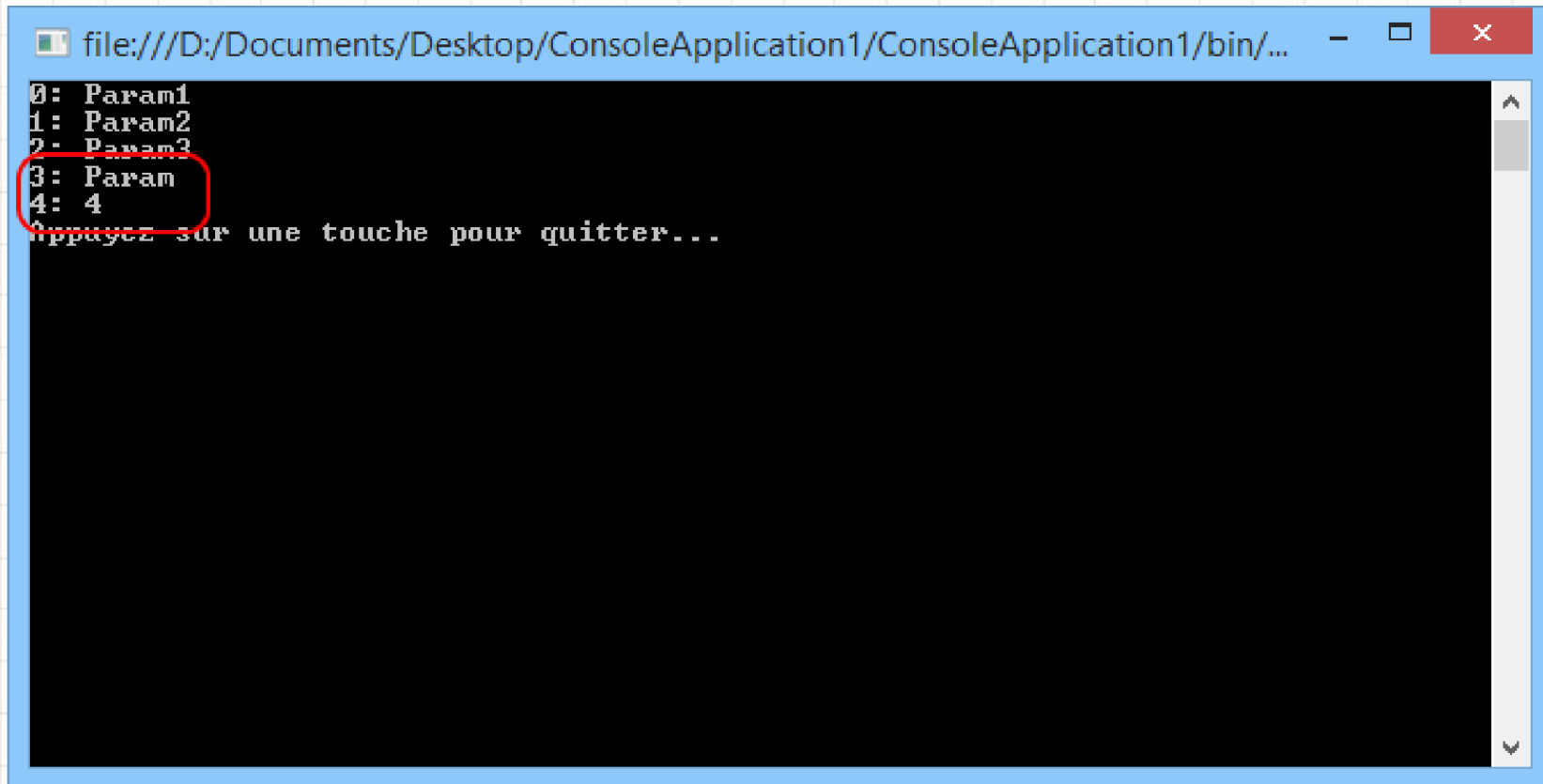
```
class Program
{
    static void Main(string[] args)
    {
        for ( int Index = 0 ; Index < args.Length ; Index ++ )
        {
            Console.Out.WriteLine ( string.Format ( "{0}: {1}" , Index, args[Index]) );
        }

        Console.Out.WriteLine ( "Appuyez sur une touche pour quitter..." );

        Console.In.Read ();
    }
}
```

VS – Ligne de commande

- Exécutons du projet.



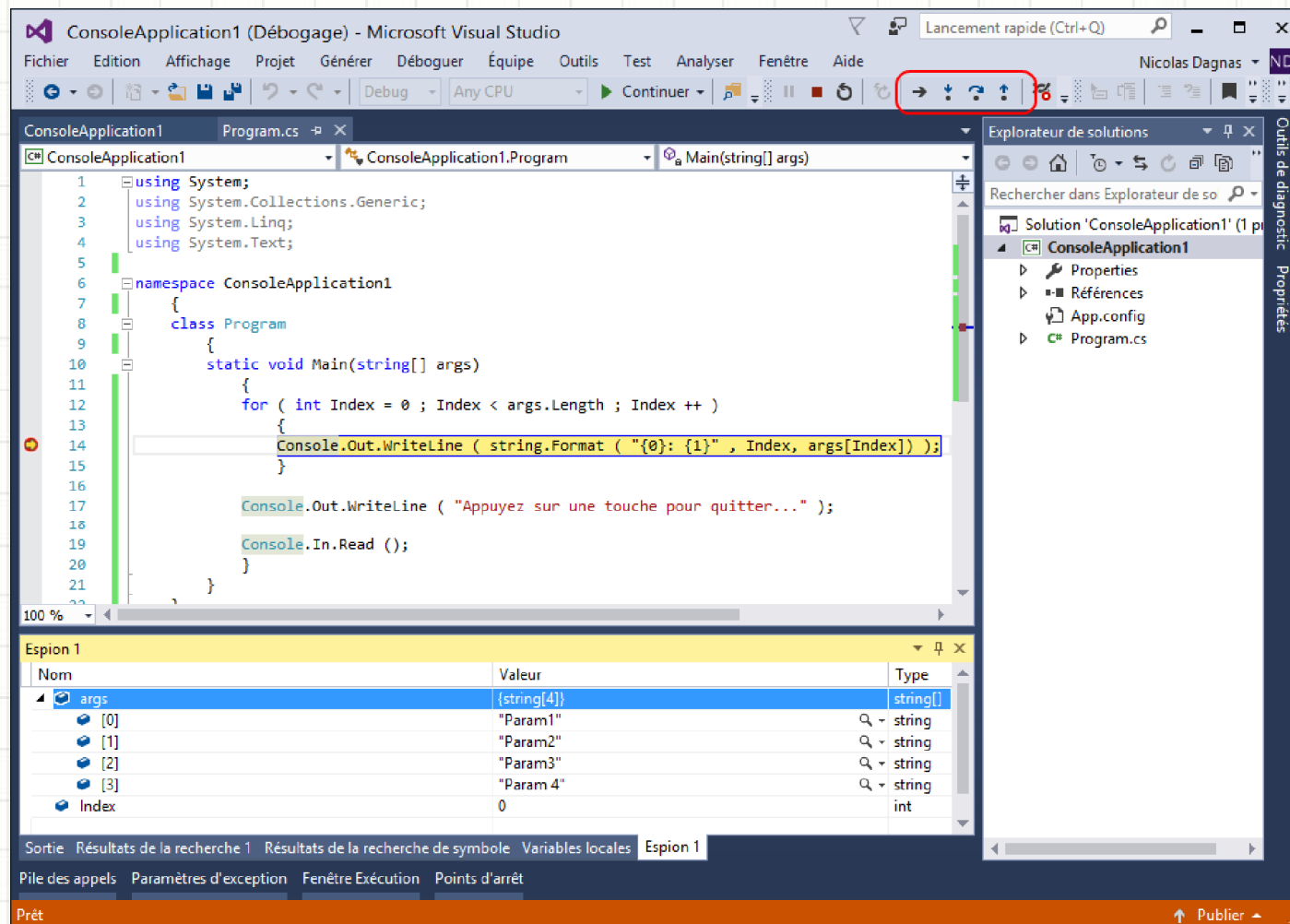
```
file:///D:/Documents/Desktop/ConsoleApplication1/ConsoleApplication1/bin/...  
0: Param1  
1: Param2  
2: Param3  
3: Param  
4: 4  
Appuyez sur une touche pour quitter...
```

VS – L'exécution Pas à Pas

- Les points d'arrêts.
- La fenêtre « Espion ».
 - Déboguer > Fenêtres > Espion (menu et fenêtre disponible uniquement à l'exécution).
- Ajouter un espion, comment ?

VS – L'exécution Pas à Pas

- L'exécution et le débogage pas à pas.



VS – Namespace

- Qu'est ce qu'un « namespace » ?
- Trouver le « namespace » d'un objet ?

VS – L'objet « Debug »

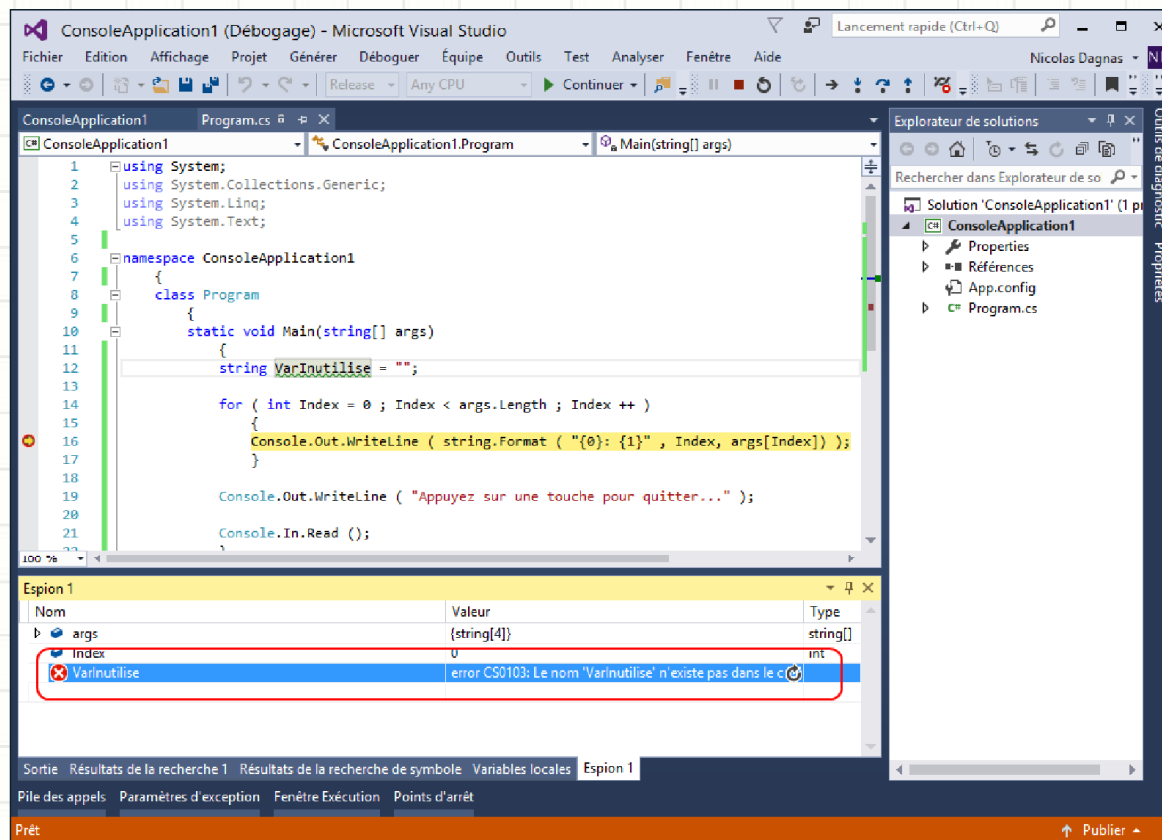
- L'objet « Debug » et son « namespace ».
- La fenêtre « Sortie ».
 - Déboguer > Fenêtres > Sortie (menu et fenêtre disponible uniquement à l'exécution).
- Des limitations ?

VS – Modes d'exécutions

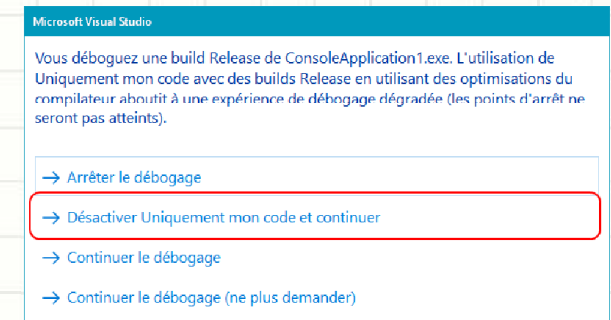
- Le mode d'exécution « Debug »
- Le mode d'exécution « Release »

VS – Le Mode Release !

- Le mode « release » optimise le code. Qu'est ce que cela implique ?



Si Visual Studio vous indique que vous déboguez une build Release et vous propose 4 options, sélectionnez : « Désactiver Uniquement mon code et continuer ».

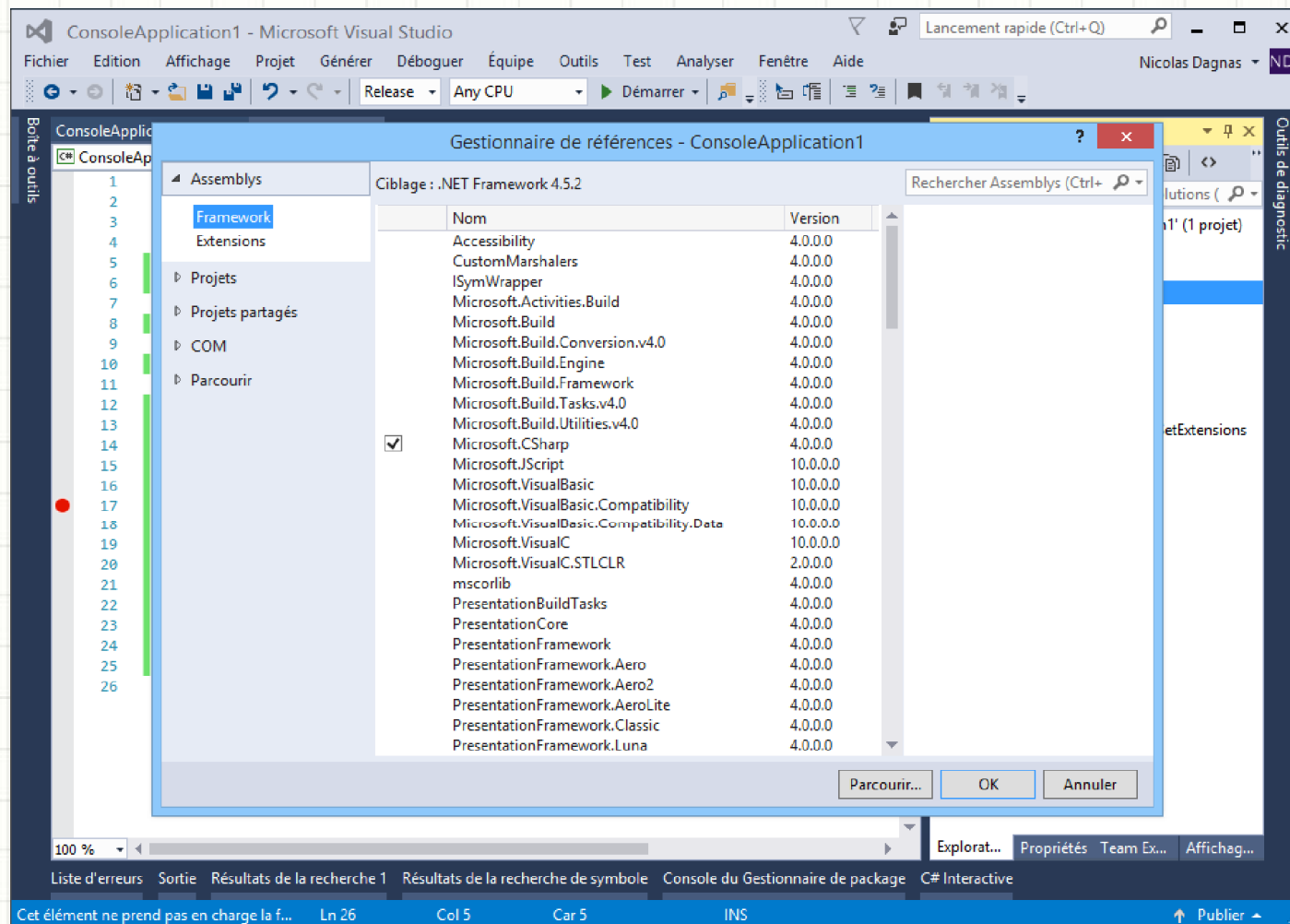


VS – Références

- Qu'est ce qu'une « référence » ?
- Où les trouver ?
- Il y a 3 sources principales de références :
 - Les références systèmes, les plus classiques bien sûr.
 - Les références fournies par un tiers.
 - Nuget/GitHub qui sont des bibliothèques de références sur Internet (Nous y reviendrons dans le TP).

VS – Références

- Ajouter une référence.



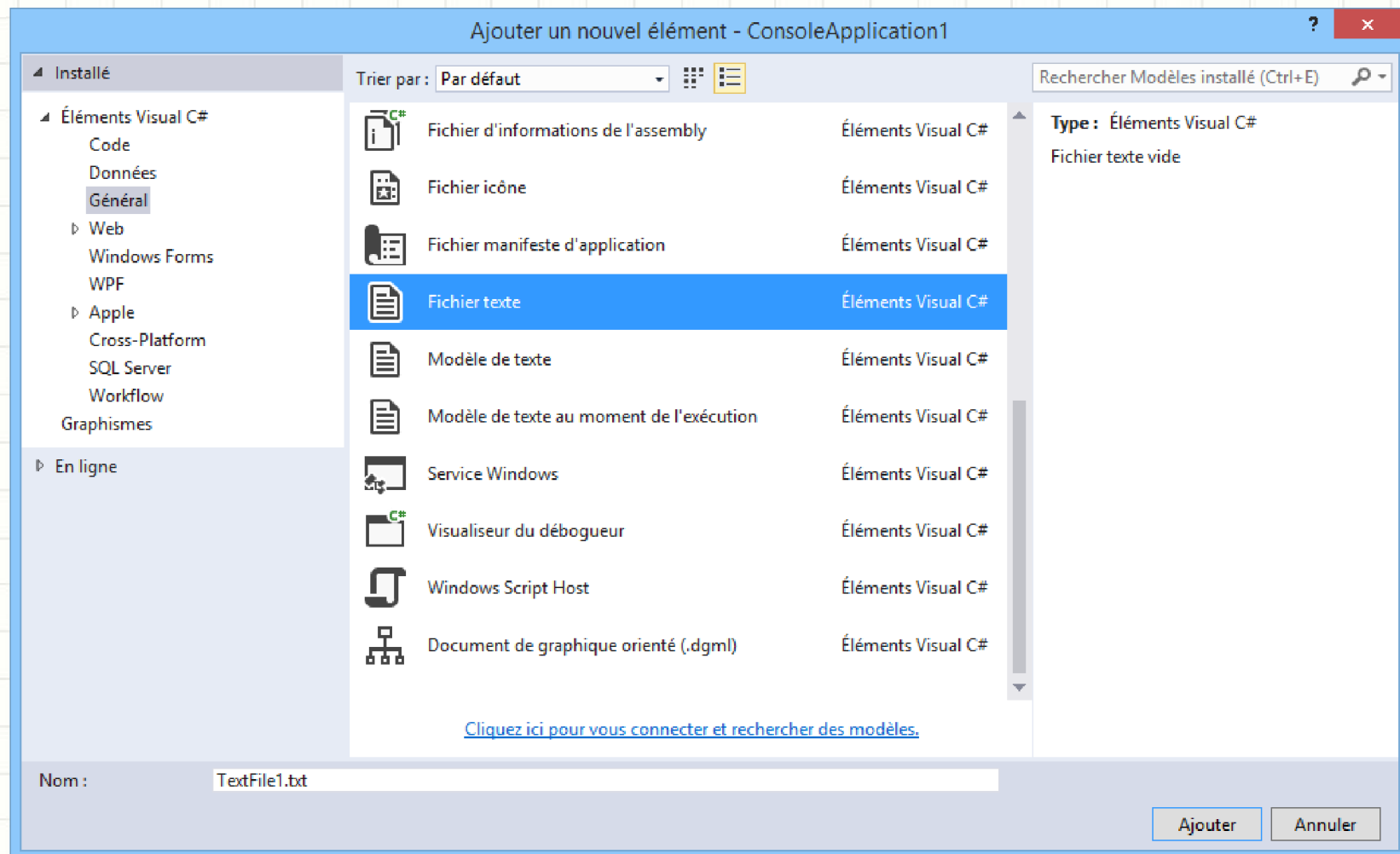


VS – Les Flux

- Lire un fichier texte
- Écrire dans un fichier texte

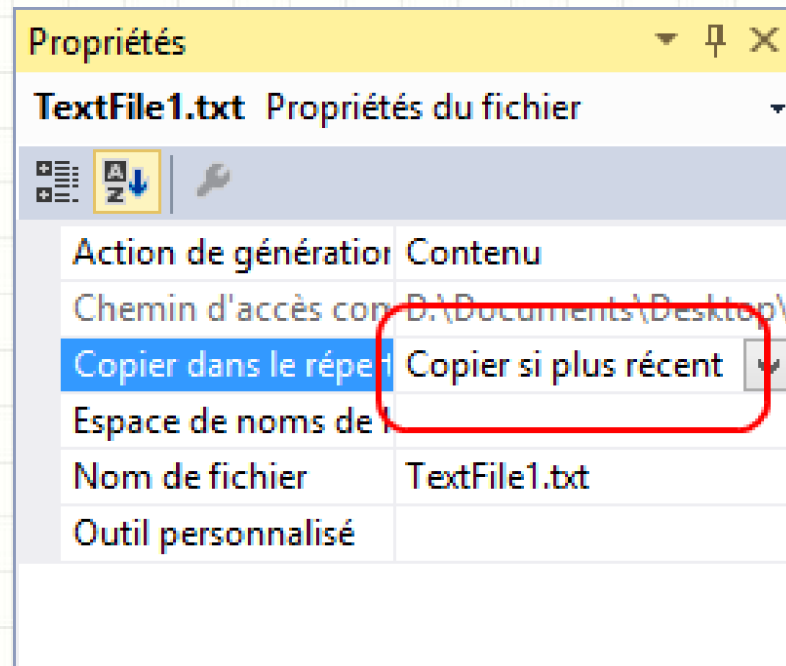
VS – Les Flux (lecture)

- Ajouter un « Fichier texte ».



VS – Les Flux (lecture)

- « Propriétés » du fichier.



VS – Les Flux (lecture)

- Code à saisir :

```
class Program
{
    static void Main(string[] args)
    {
        // Est ce qu'un paramètre a été passé à l'application ?

        if ( args.Length > 0 && File.Exists ( args[0] ) )
        {
            // Ouvrons le fichier

            StreamReader Sr = File.OpenText ( args[0] );

            string Ligne;

            while ( ( Ligne = Sr.ReadLine () ) != null )
            {
                // Affichons dans notre console, le contenu du fichier

                Console.Out.WriteLine ( Ligne );
            }

            Sr.Close (); // <= Fermons le fichier

            Sr.Dispose (); // Assurons nous que toutes les ressources sont libérées
        }

        Console.Out.WriteLine ( "Appuyez sur une touche pour quitter..." );

        Console.In.Read ();
    }
}
```

VS – Le using « disposer »

- L'interface « IDisposable » et la méthode « Dispose () ».

```
using ( StreamReader Sr = File.OpenText ( args[0] ) )  
{  
    string Ligne;  
  
    while ( ( Ligne = Sr.ReadLine () ) != null )  
    {  
        // Affichons dans notre console, le contenu du fichier  
  
        Console.Out.WriteLine ( Ligne );  
    }  
}
```

VS – Les Flux (écriture)

- Modification de la méthode « Main » :

```
if ( args.Length > 0 && File.Exists ( args[0] ) )
{
    // Ouvrons le fichier

    using ( StreamReader Sr = File.OpenText ( args[0] ) )
    {
        string Ligne;

        while ( ( Ligne = Sr.ReadLine () ) != null )
        {
            // Affichons dans notre console, le contenu du fichier

            Console.WriteLine ( Ligne );
        }
    }

    using ( StreamWriter Sw = File.AppendText ( args[0] ) )
    {
        // Petit exercice de découverte ...
    }
}
```

VS – Les Flux (écriture)

- Et si on faisait un petit exercice ?
 - Ajoutez une méthode statique (comme « Main ») que vous nommerez « Demander_Et_Ecrire » prenant en paramètre d'entrée un objet « StreamWriter » et sans valeur de retour.
 - Le début de cette méthode affichera sur la console le texte :
 - Saisissez votre texte puis appuyez sur 'ENTER' ('exit' pour sortir).
 - Ajouter ensuite une boucle permettant à un utilisateur de saisir du texte.
 - Utilisez Console.In...
 - Lorsque l'utilisateur appuiera sur 'ENTER', ajoutez le texte saisi dans le fichier texte à l'aide de l'objet passé en paramètre, retour chariot compris. Indiquez par un texte dans la console que le texte saisi a été ajouté au fichier.
 - Lorsque l'utilisateur saisira 'exit' (ou même 'Exit' pour complexifier un peu) puis appuiera sur 'ENTER', vous sortirez de la boucle pour continuer normalement l'exécution de votre projet.
- Cette méthode est à appeler dans la section « using » que vous avez précédemment ajoutée.

VS – Les Flux (écriture)

- Une fois que vous avez codé votre méthode, testez là sans hésiter à recourir au déboguage comme vu précédemment.
- Pour voir le fichier texte modifié, allez dans le dossier « bin » puis « debug » de votre projet, passez par l'explorateur.
- Ensuite, rappelez-vous, quand nous avons ajouté notre fichier texte à notre projet, nous avons modifié une propriété : « Copier dans le répertoire de sortie ». Modifiez votre fichier encore ouvert dans l'E.D.I., enregistrez le, puis exécutez votre projet.
- Le contenu du fichier se trouvant dans « bin/debug » s'en retrouve écrasé par le contenu du fichier que vous venez de modifier.

VS – Intellisense

- Et la documentation dans tout ça ?

```
/// <summary>  
/// |  
/// </summary>  
/// <param name="Sw"></param>  
private static void Demander_Et_Ecrire ( StreamWriter Sw )  
{  
    // ...  
}
```

VS – Intellisense

- Renseignez les éléments « summary » et « param » comme suit :

```
/// <summary>
/// Demande à l'utilisateur de saisir du texte pour l'écrire dans le fichier.
/// </summary>
/// <param name="Sw">Objet <b>StreamWriter</b> permettant l'écriture.</param>
private static void Demander_Et_Ecrire ( StreamWriter Sw )
{
    // ...
}
```

- Puis en laissant la souris sur l'appel à cette méthode, nous voyons apparaître notre résumé :

```
using ( StreamWriter Sw = File.AppendText ( args[0] ) )
{
    // Petit exercice de découverte ...

    Demander_Et_Ecrire ( Sw );
}

void Program.Demander_Et_Ecrire(StreamWriter Sw)
    Demande à l'utilisateur de saisir du texte pour l'écrire dans le fichier.
```

VS – Les Exceptions

- Les exceptions très [voir trop] présentes en « C# ».

```
private static void Demander_Et_Ecrire ( StreamWriter Sw )  
{  
    // Est ce que Sw est renseigné ?  
  
    if ( Sw == null )  
        throw new ArgumentNullException ( "Sw", "Sw vaut null !" );  
  
    // ...  
}
```

VS – Les Exceptions

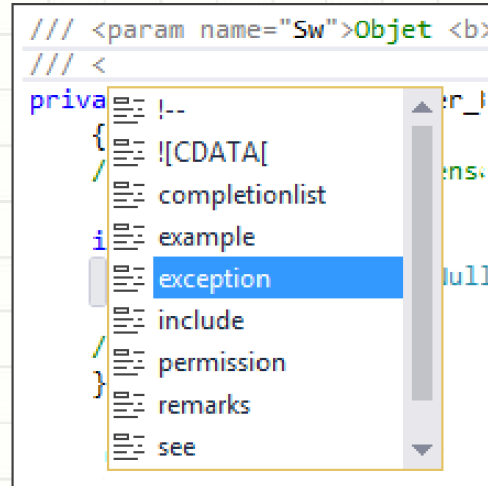
- Intercepter une exception, un grand classique : « Try ... Catch ».

```
try
{
    using ( StreamWriter Sw = File.AppendText ( args[0] ) )
    {
        // Petit exercice de découverte ...

        Demander_Et_Ecrire ( Sw );
    }
}
catch ( ArgumentNullException )
{
    // Si on arrive ici, c'est qu'un argument vaut NULL
    // et que cela a généré une exception !
}
catch ( Exception Err )
{
    // Ici une autre exception a été générée :/
}
}
```

VS – Intellisense (Suite)

- Documenter les exceptions :



```
/// <summary>  
/// Demande à l'utilisateur de saisir du texte pour l'écrire dans le fichier.  
/// </summary>  
/// <param name="Sw">Objet <b>StreamWriter</b> permettant l'écriture.</param>  
/// <exception cref="ArgumentNullException">...</exception>
```


VS – Raccourcis clavier

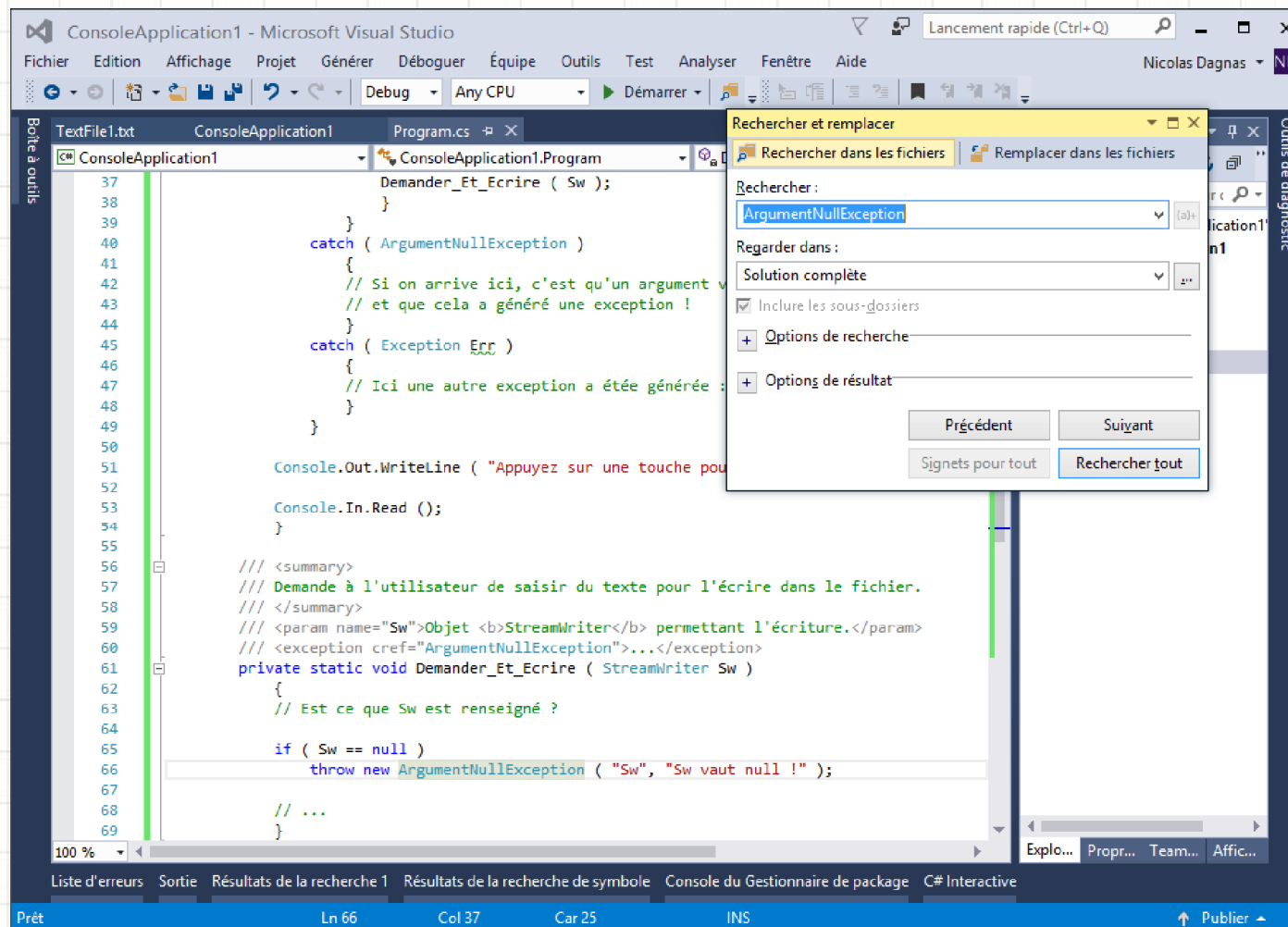
- Quelques raccourcis claviers dans Visual Studio :
 - F5 : Lancer l'application
 - F10 : Exécuter jusqu'à la ligne suivante
 - F11 : Exécuter en entrant dans les fonctions
 - F12 : Permet d'aller à la déclaration de l'objet sélectionné
 - Ctrl+C : Copier la ligne entière avec le retour chariot
 - Ctrl+X : Couper la ligne entière avec le retour chariot
 - F6 : Compiler le projet (Sans l'exécuter)
 - Ctrl+F : Effectuer une recherche dans la page en cours

 - Ctrl+Alt+F : Effectuer une recherche globale au projet
 - Ctrl+K > Ctrl+C : Placer la ligne en commentaire

 - Etc...

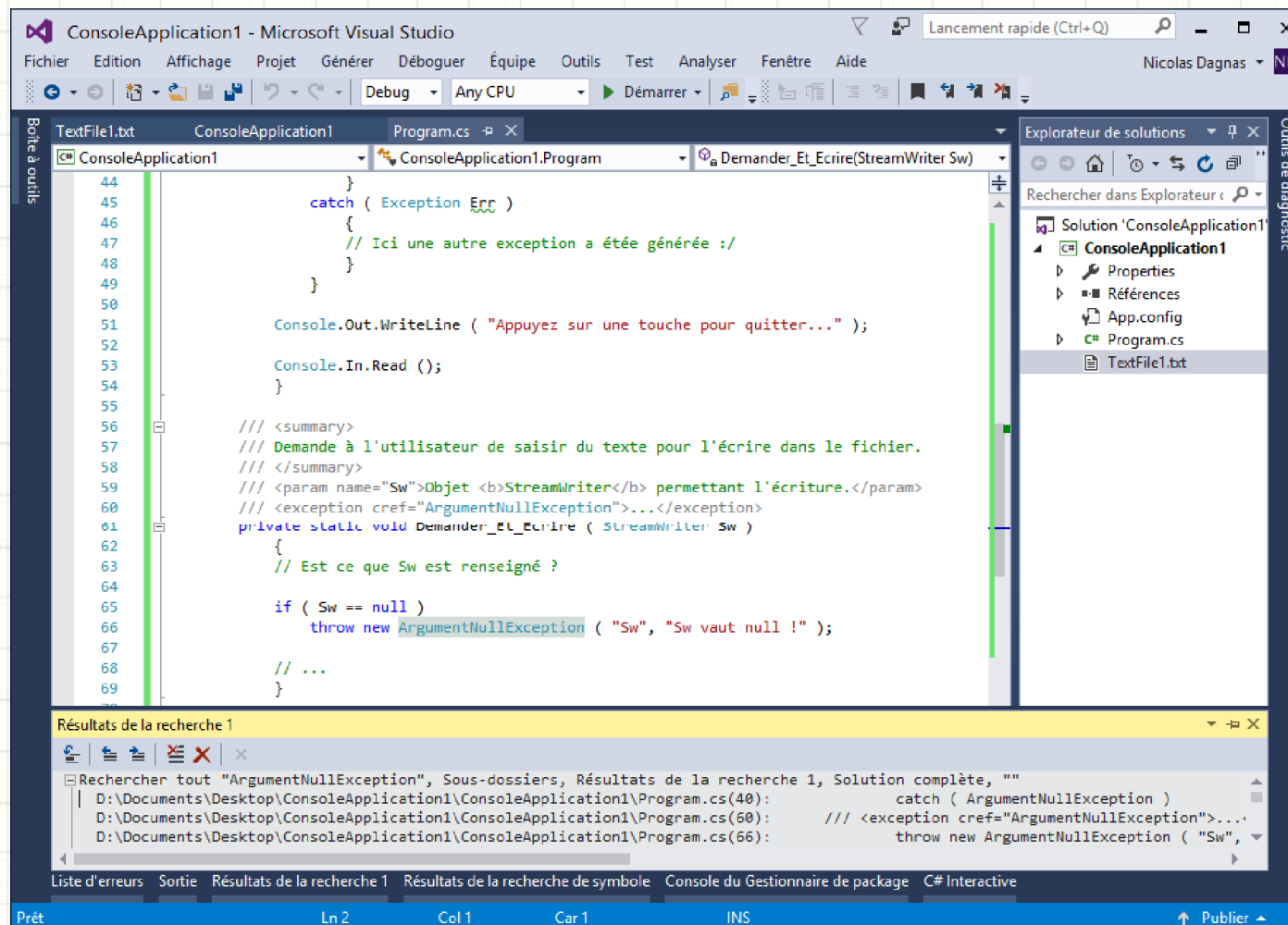
VS – La Recherche classique

- La recherche locale « Ctrl+F » ou la recherche dans tout le projet « Ctrl+Maj+F ».



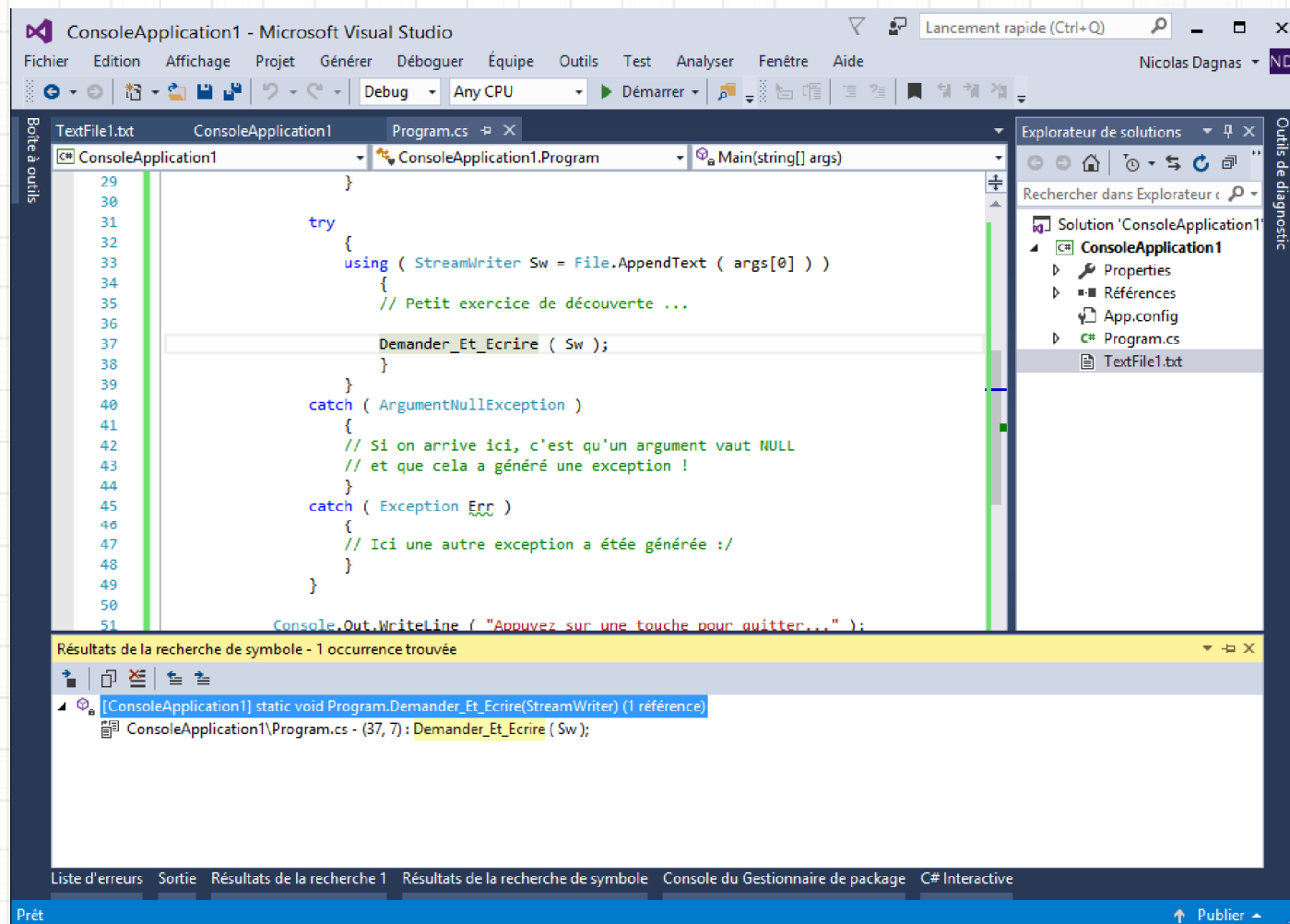
VS – Résultat de la recherche

- Le résultat d'une recherche dans tout le projet.



VS – La Recherche de référence

- La recherche de toutes les références à un objet.

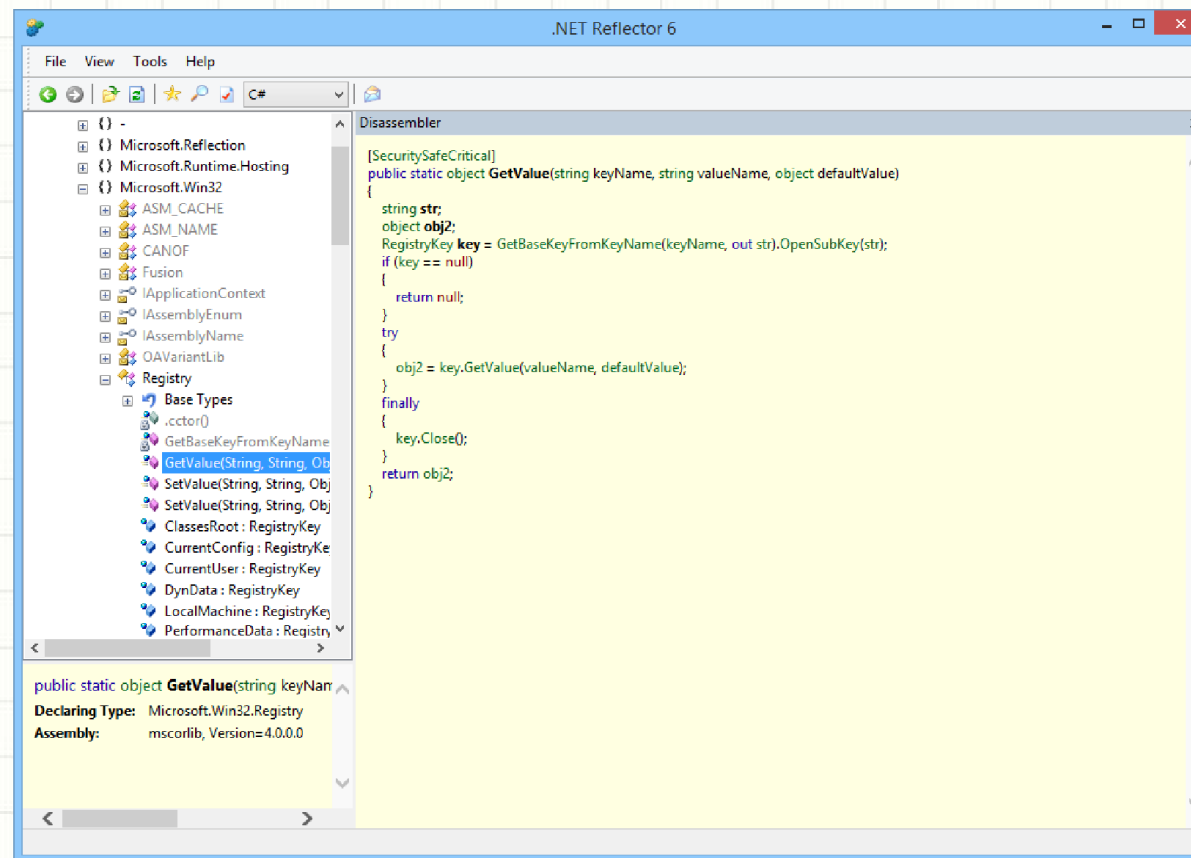


VS – Mes Fichiers !

- Petit rappel des fichiers composant le projet :
 - Le fichier « Solution » => .sln, pouvant référencer plusieurs projets.
 - Le fichier « Projet » => .csproj, permet de constituer notre projet.
 - Le fichier « User » => .user, contient les paramètres d’affichage du projet en cours, comme les éléments ouverts.
 - Les fichiers « Sources » => .cs, ...
 - Le dossier « Properties » => en général, il n’est jamais modifié.
 - Les dossiers « bin » et « obj » => contient les fichiers compilés.

VS – Dé-compilateurs

- Les dé-compilateurs.



- Depuis Vs 2019, on dispose d'un outil intégré équivalent : Outils - Options - C# - Avancé.

VS – Fin de la première partie

- Les types principaux :
 - « .Net Framework » : destiné aux applications et services basiques windows.
 - « .Net Core » : se destine plus à l'environnement UWP, c'est-à-dire le Store Windows par exemple et la création d'applications pour Windows IOT.
 - « Xamarin » : pour tout ce qui sera cross-platform « Android », « iOS » et « OS-X ».
- Mais tous se basent sur « .Net Standard » qui en est la base.



VS – Partie 2

« Projet Winform »

Un éditeur de texte minimaliste