Preuves

IHM : Interface Homme-Machine (XAML, WPF)

Je sais décrire le contexte de mon application pour qu'il soit compréhensible par tout le monde.

Contexte application

Le concept de cette application est de créer une bibliothèque de jeux vidéo dans laquelle se trouvent toutes les informations essentielles du jeu: synopsis, date de création, compatibilité avec les consoles…

L'application a pour but de stocker des informations sur ces jeux vidéo pour que l'utilisateur trouve un jeu auquel jouer. De plus, il a accès à d'autres informations comme des théories, des visuels et des musiques associées à ce dernier.

C'est une application tout public. La personne voulant se lancer dans la pratique du jeu vidéo peut avoir les informations qu'elle souhaite sur les différents jeux présents sur le marché. Elle peut donc choisir un jeu en fonction de ses préférences pour débuter. De plus, une personne jouant déjà peut tout aussi bien utiliser l'application pour enrichir sa culture sur une franchise ou un jeu en particulier. C'est aussi une application pour des jeunes utilisateurs puisque nous voulons parler de jeux souvent accessibles à un jeune public. Toutefois dans la description des jeux il y aura bien évidemment l'âge conseillé pour jouer. L'application sera gratuite pour les utilisateurs pour la rendre totalement accessible.

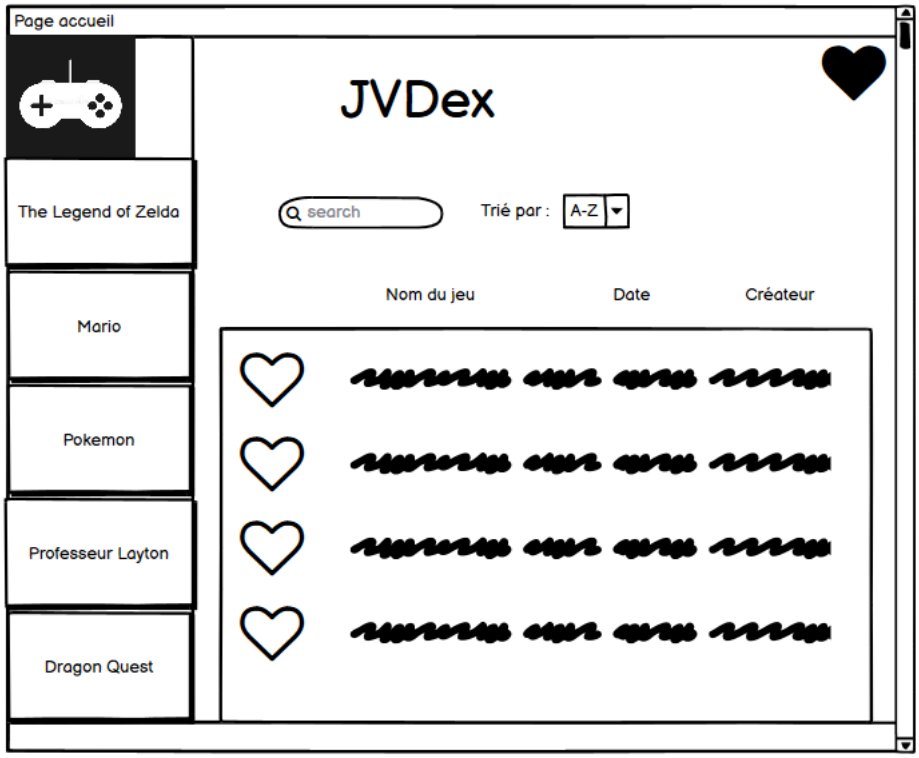
Les jeux seraient triés par franchise. L'utilisateur peut tout d'abord chercher le jeu qu'il souhaite avec une barre de recherche. En outre, les jeux peuvent être triés par ordre alphabétique, date, créateur pour simplifier la recherche de l'utilisateur.

Une liste des différentes franchises serait mise à la disposition de l'utilisateur pour lui permettre de chercher les informations qu'il souhaite plus facilement.

Pour avoir accès aux informations de ces jeux il faut passer par les différentes franchises, par le tri et la recherche. Il est possible de mettre des jeux en favoris pour les retrouver plus tard. Cet onglet est constamment accessible à l'utilisateur.

Voici l'explication du contexte de l'application, dans le fichier *documentation_appliPartie1.pdf*

Je sais dessiner des sketchs pour concevoir les fenêtres de mon application.

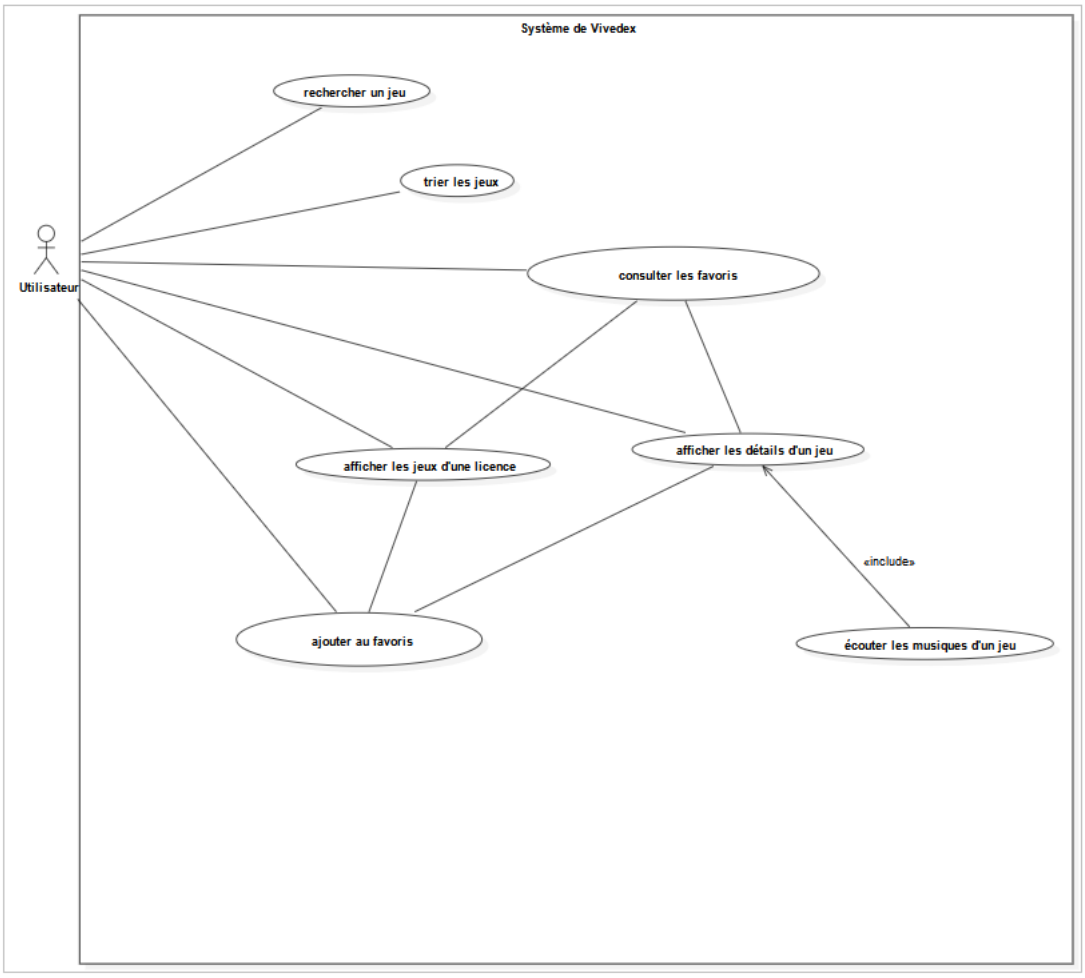


Exemple d'un des sketchs sur notre vue principale.

Je sais enchaîner mes sketchs au sein d'un story-board.

Nous avons fait un storyboard expliqué pour montrer l'enchaînement possible des différents sketcks. Cela se trouve dans *documentation_appliPartie1.pdf*

Je sais concevoir un diagramme de cas d'utilisation qui représente les fonctionnalités de mon application.



Je sais concevoir un diagramme de classes qui représente mon application.

Le diagramme est dans le fichier descriptionDiagramme2.pdf

Je sais réaliser un diagramme de paquetages qui illustre bien l'isolation entre les parties de mon application.

Le diagramme est dans le fichier descriptionDiagramme2.pdf

Je sais décrire mes deux diagrammes en mettant en valeur et en justifiant les éléments essentiels.

Le diagramme est dans le fichier descriptionDiagramme2.pdf

Je sais concevoir une application ergonomique.

Le diagramme est dans le fichier descriptionDiagramme2.pdf

Je sais concevoir une application avec une prise en compte de l'accessibilité.

Le diagramme est dans le fichier descriptionDiagramme2.pdf

Je sais mettre en avant dans mon diagramme de classes la persistance de mon application.

Le diagramme est dans le fichier descriptionPersistance.pdf

Je sais mettre en avant dans mon diagramme de classes ma partie personnelle.

Le diagramme est dans le fichier descriptionPersistance.pdf

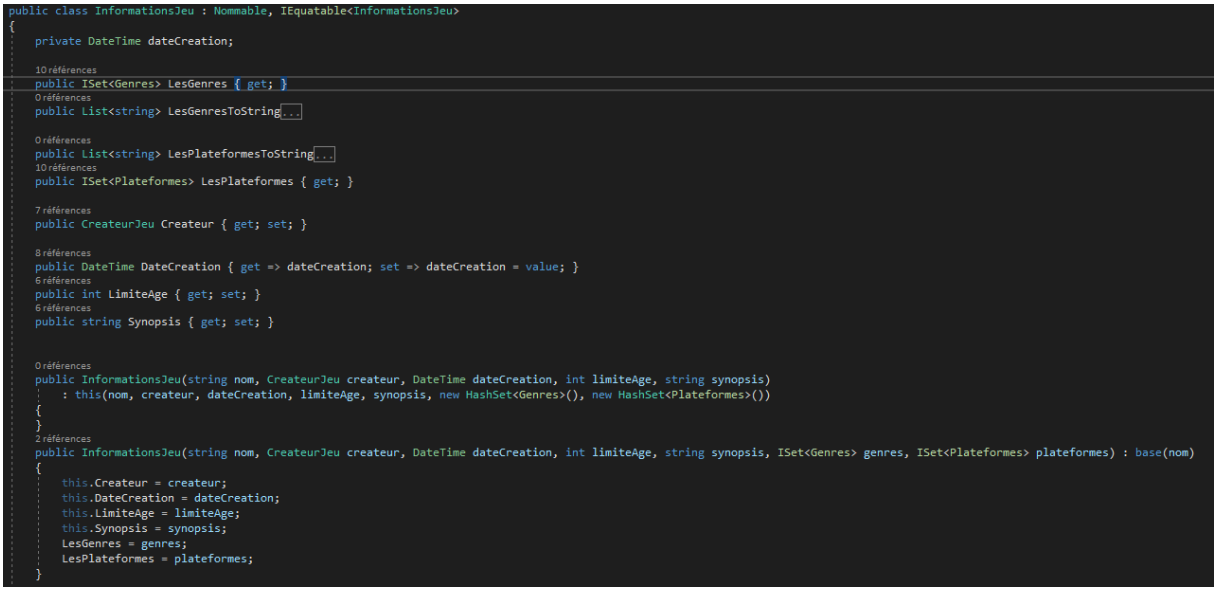
Je sais mettre en avant dans mon diagramme de paquetages la persistance de mon application.

Le diagramme est dans le fichier descriptionPersistance.pdf

Conception et Programmation Orientées Objets (C#, .NET)

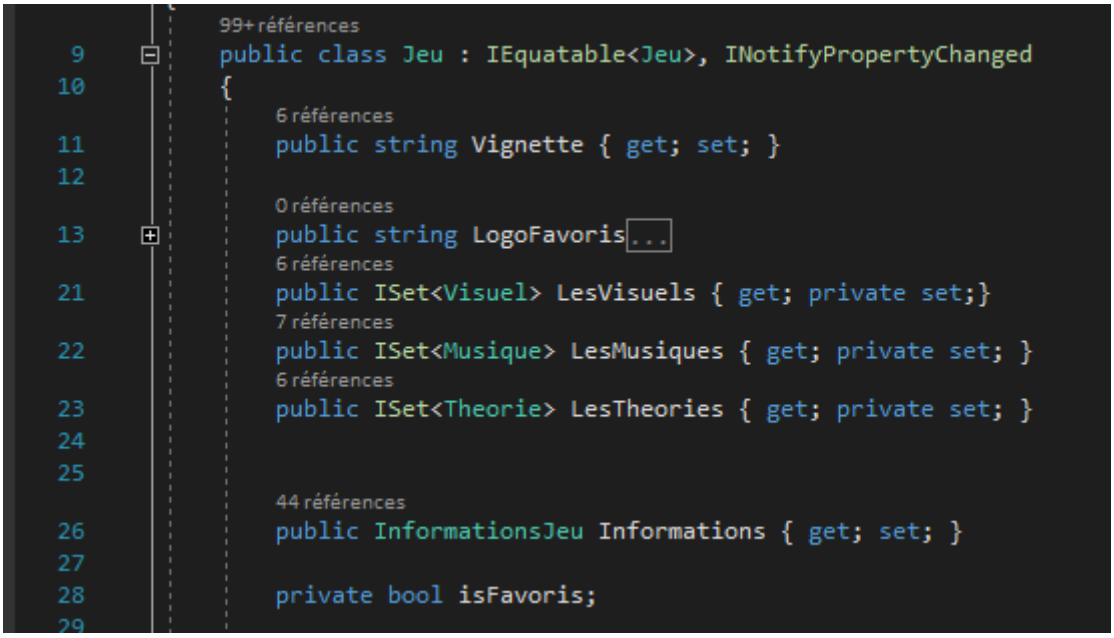
Je maîtrise les bases de la Programmation C# (classes, structures, instances…) :

Nous avons créé de nombreuses classes dans plusieurs projets différents, par exemple *InformationsJeu* que l'on utilise énormément dans notre programme :



```
public class InformationsJeu : Nommable, IEquatable<InformationsJeu>
{
    private DateTime dateCreation;

    10 références
    public ISet<Genres> LesGenres { get; }
    0 références
    public List<string> LesGenresToString...

    0 références
    public List<string> LesPlateformesToString...
    10 références
    public ISet<Plateformes> LesPlateformes { get; }

    7 références
    public CreateurJeu Createur { get; set; }

    8 références
    public DateTime DateCreation { get => dateCreation; set => dateCreation = value; }
    6 références
    public int LimiteAge { get; set; }
    6 références
    public string Synopsis { get; set; }


    0 références
    public InformationsJeu(string nom, CreateurJeu createur, DateTime dateCreation, int limiteAge, string synopsis)
        : this(nom, createur, dateCreation, limiteAge, synopsis, new HashSet<Genres>(), new HashSet<Plateformes>())
    {
    }
    2 références
    public InformationsJeu(string nom, CreateurJeu createur, DateTime dateCreation, int limiteAge, string synopsis, ISet<Genres> genres, ISet<Plateformes> plateformes) : base(nom)
    {
        this.Createur = createur;
        this.DateCreation = dateCreation;
        this.LimiteAge = limiteAge;
        this.Synopsis = synopsis;
        LesGenres = genres;
        LesPlateformes = plateformes;
    }
```

On instancie un *InformationsJeu* dans la classe *Jeu* (ligne 26):



```
    99+ références
 9  public class Jeu : IEquatable<Jeu>, INotifyPropertyChanged
10  {
        6 références
11      public string Vignette { get; set; }
12
        0 références
13      public string LogoFavoris...
        6 références
21      public ISet<Visuel> LesVisuels { get; private set;}
        7 références
22      public ISet<Musique> LesMusiques { get; private set; }
        6 références
23      public ISet<Theorie> LesTheories { get; private set; }
24
25
        44 références
26      public InformationsJeu Informations { get; set; }
27
28      private bool isFavoris;
29
```

Nous avons aussi créé des Enum, par exemple *Plateformes* :

```
public enum Plateformes
{
    PC,
    PS1,
    PS2,
    PS3,
    PS4,
    PS5,
    Gameboy,
    [EnumDescription("Gameboy Color")]
    Gameboy_Color,
    [EnumDescription("Gameboy Advance")]
    Gameboy_Advance,
    Gamecube,
    DS,
    Wii,
    [EnumDescription("Wii U")]
    Wii_U,
    [EnumDescription("3DS")]
    TroisDS,
    [EnumDescription("Nintendo Switch")]
    NintendoSwitch,
    Xbox,
    [EnumDescription("Xbox 360")]
    Xbox360,
    [EnumDescription("Xbox One")]
    XboxOne,
    [EnumDescription("Xbox Serie X")]
    XboxSerieX,
    IOS,
    Android,
    [EnumDescription("Nintendo 64")]
    Nintendo64,
    NES
}
```

Je sais utiliser l'abstraction à bon escient (héritage, interfaces, polymorphisme) :

Pour la persistance, de l'application, nous devions sérialiser la liste de jeux et le dictionnaire de franchise contenues dans *StockApp*. Pour cela, pour avons créé l'interface *IPersistanceStockApp*. Ainsi toutes les classes l'implémentant pourrais être utilisé comme « stockage » de données.

```
namespace BibliothèqueApplication
{
    7 références
    public interface IPersistanceStockApp
    {
        4 références
        (IList<Jeu> jeux, Dictionary<Franchise,List<Jeu>> franchises) ChargeDonnees();

        5 références
        void SauvegardeDonnees(IList<Jeu> jeux, Dictionary<Franchise, List<Jeu>> franchises);
    }
}
```

Pour faire les tests, nous avons donc utiliser la classe *Stub*, qui implémente *IPersistanceStockApp*. La méthode *ChargeDonnees()* possédant des données utilisées seulement pour les tests :

```
1 référence
public class Stub : IPersistanceStockApp
{
    4 références
    public (IList<Jeu> jeux, Dictionary<Franchise, List<Jeu>> franchises) ChargeDonnees()
    {

        Jeu test1 = new Jeu("Test1", new Createur("Matt", "Orillon", new DateTime(2020, 1, 1)), new DateTime(2020, 04, 30), 3, "Ceci est un test de synopsis", "");
        Jeu test1_bis = new Jeu("Test1", new Createur("Matt", "Orillon", new DateTime(2020, 1, 1)), new DateTime(2020, 04, 30), 3, "Ceci est un test de synopsis", "");

        // Jeu zelda1 = new Jeu("Breath of the Wild", new Studio("Nintendo"), new DateTime(2020, 03, 30), 3, "Ceci est un test de synopsis 1", "\\Images;Component\\ima
        Jeu zelda1 = new Jeu("Breath of the Wild", new Studio("Nintendo"), new DateTime(2020, 03, 30), 3, "Ceci est un test de synopsis 1", @"/Ressources;Component/ima
        Jeu zeldaWindwaker = new Jeu("Windwaker", new Studio("Nintendo"), new DateTime(2020, 03, 30), 3, "Ceci est un test de synopsis 1", "\\Images;Component\\images\
        Jeu zeldaWindwaker2 = new Jeu("Windwaker2", new Studio("Nintendo"), new DateTime(2020, 03, 30), 3, "Ceci est un test de synopsis 1", "\\Images;Component\\image
        Jeu zeldaWindwaker3 = new Jeu("Windwaker3", new Studio("Nintendo"), new DateTime(2020, 03, 30), 3, "Ceci est un test de synopsis 1", "\\Images;Component\\image
        Jeu zeldaWindwaker4 = new Jeu("Windwaker4", new Studio("Nintendo"), new DateTime(2020, 03, 30), 3, "Ceci est un test de synopsis 1", "\\Images;Component\\image
        Jeu test2 = new Jeu("Test2", new Createur("Victor", "Gaillard", new DateTime(2020, 10, 14)), new DateTime(2020, 05, 04), 7, "Ceci est un test de synopsis 2", "

        Jeu mario = new Jeu("Super Mario", new Studio("Nintendo"), new DateTime(2012, 05, 04), 7, "Test du synopsis jeu", "");
        Jeu new_mario = new Jeu("New Super Mario Bros", new Studio("Nintendo"), new DateTime(2012, 05, 04), 7, "Test du synopsis jeu", "");

        Jeu dragonQuest = new Jeu("Dragon Quest", new Studio("Enix"), new DateTime(2012, 05, 04), 7, "Test du synopsis jeu", "");

        //Franchise fMario = new Franchise("Mario", "\\Images;Component\\images\\Franchises\\Mario\\background.jpg", "#DE1111");
        Franchise fMario = new Franchise("Mario", @"pack://application:,,,/Images;Componment/images/Franchises/Mario/background.jpg", "#DE1111");
        Franchise fZelda = new Franchise("The Legend of Zelda", "\\Images;Component\\images\\Franchises\\Zelda\\background.jpg", "#1BC213");
        Franchise fDragonQuest = new Franchise("Dragon Quest", "\\Images;Component\\images\\Franchises\\DragonQuest\\background.jpg", "#2092DC");
        Franchise fPokemon = new Franchise("Pokemon", "\\Images;Component\\images\\Franchises\\Pokemon\\background.jpg", "#FFFB00");
        Franchise fProfLayton = new Franchise("Professeur Layton", "\\Images;Component\\images\\Franchises\\Layton\\background.jpg", "#896335");
        Theorie thZelda1 = new Theorie("vie de Link", "link est un farfadet");
        Theorie thZelda2 = new Theorie("vie de Link2", "link est un farfadet");
        Visuel V1 = new Visuel("\\Images;Component\\images\\Franchises\\Zelda\\Jeux\\BOTW\\vignette.png", "Image du jeu");
        Visuel V2 = new Visuel("\\Images;Component\\images\\Franchises\\Zelda\\Jeux\\BOTW\\vignette.png", "Image du jeu 2");
```

Mais pour le déploiement, nous avions besoin d'une persistance dans un fichier, c'est pourquoi nous avons fait la classe *PersistanceFichierTexte*, qui comme son nom l'indique, utilise un fichier texte pour charger et sauvegarder les données :

```
public class PersistanceFichierTexte : IPersistanceStockApp
{
    4 références
    public (IList<Jeu> jeux, Dictionary<Franchise, List<Jeu>> franchises) ChargeDonnees()
    {
        List<Jeu> lJeux = new List<Jeu>();
        Dictionary<Franchise, List<Jeu>> dFranchise = new Dictionary<Franchise, List<Jeu>>();

        using(FileStream fs = File.OpenRead("infoJVDex.txt"))
        {
            using (TextReader reader = new StreamReader(fs))
            {
                int.TryParse(reader.ReadLine(), out int nbFranchises); //TryParse = convertit string to int => valeur récup avec out
                for(int i = 0; i<nbFranchises; i++)
                {
                    Franchise f = LireFranchise(reader);
                    dFranchise.Add(f, new List<Jeu>());
                    int.TryParse(reader.ReadLine(), out int nbJeux);
                    for(int k = 0; k<nbJeux; k++)
                    {
                        Jeu j = LireJeu(reader);
                        bool.TryParse(reader.ReadLine(), out bool isFavori);
                        j.IsFavoris = isFavori;
                        lJeux.Add(j);
                        dFranchise.TryGetValue(f, out var lFranchise);
                        lFranchise.Add(j);
                    }
                }
            }
        }
        return (lJeux, dFranchise);
    }
```

(ici on ne voit que la méthode ChargeDonnees(), mais il y en d'autres, cf Data/PersistanceFichierTexte)

Ainsi, dans app, on peut changer facilement de « mode » de persistance (on pourrait en rajouter d'autre dans le futur : xml, bases de données…) tout en ne changeant pas le type statique de la propriété *Pers* d'*App.* Il suffit d'instancier la classe de persistance que l'on souhaite :

```
public partial class App : Application
{
    /// <summary> Instance de PersistanceFichierTexte qui est la persistance utilisé ...
    2 références
    public IPersistanceStockApp Pers { get; } = new PersistanceFichierTexte();
    private static StockApp sa;
    /// <summary> Propriété qui stocke le Manager utilisé dans l'application
    14 références
    public Manager LeManager { get; set; }
    /// <summary> Propriété qui stocke le Navigator utilisé dans l'application
    5 références
    public Navigator LeNavigateur { get; set; } = new Navigator();
    /// <summary> Constructeur de App, appelé au lancement de l'application. Il perm ...
    1 référence
```

Je sais gérer des collections simples (tableaux, listes…) :

Dans la classe *Jeu*, on a utilisé des collections très simples pour stocker les *Theories, les Musiques et les Visuels*. Nous avons préféré utiliser des *Set* car on ne veut pas de doublons et on n'a pas besoins qu'elles soient triées :

```
6 références
public ISet<Visuel> LesVisuels { get; private set;}
7 références
public ISet<Musique> LesMusiques { get; private set; }
6 références
public ISet<Theorie> LesTheories { get; private set; }
```

Je sais gérer des collections avancées(dictionnaires) :

Pour stocker les *Franchise*, nous avons utilisé un dictionnaire. Chaque clé est donc une *Franchise* et chaque valeur associée est une *List<Jeu>*, qui sont donc les jeux d'une franchise.

```
public Dictionary<Franchise, List<Jeu>> ToutesLesFranchises { get => toutesLesFranchises; set => toutesLesFranchises = value; }
```

Je sais contrôler l'encapsulation au sein de mon application :

Pour encapsuler nos données, nous avons utilisé des Propriété, qui sont donc publiques, d'où on peut récupérer les valeurs stockées dans les champs privés associé.

Exemple : Propriété publique *TousLesJeux* => Champ privé *tousLesJeux*

```
private IList<Jeu> tousLesJeux;
private Dictionary<Franchise, List<Jeu>> toutesLesFranchises;
private string jeuRecherche;
private TypeTri typeTriJeuSelected;
private Franchise franchiseSelected;
private Jeu jeuSelected;

public event PropertyChangedEventHandler PropertyChanged;


/// <summary> Propriété qui rend la liste des jeux, recherchés ou non, trié. Ell ...
10 références
public IList<Jeu> TousLesJeux...

/// <summary> permet de savoir quel est le type de tri sélectionné
6 références
public Dictionary<Franchise, List<Jeu>> ToutesLesFranchises { get => toutesLesFranchises; set => toutesLesFranchises = value; }
5 références
public TypeTri TypeTriJeuSelected...

0 références
public string TypeTriJeuSelectedToString ...

0 références
public string DescriptionTypeTri => typeTriJeuSelected.ToDescription();


/// <summary> propriété
4 références
public string JeuRecherche...
0 références
public IList<TypeTri> ListTypeTriJeu => new List<TypeTri>(Enum.GetValues(typeof(TypeTri)).Cast<TypeTri>());

9 références
public Jeu JeuSelected...
0 références
public IList<TypeTri> ListTypeTriJeux => new List<TypeTri>(Enum.GetValues(typeof(TypeTri)).Cast<TypeTri>());

7 références
public Franchise FranchiseSelected...

1 référence
public IList<Jeu> JeuxDeLaFranchiseSelected...
```

Je sais tester mon application :

Nous avons un projet entier consacré aux tests de notre application, *ConsoleTest.* Cependant nous n'avons fait que des tests fonctionnels :

```
private static void Test_Trier_Jeux(Manager app)
{



    Console.WriteLine("Comment voulez vous trié? ");
    Console.WriteLine("1) A-Z");
    Console.WriteLine("2) Z-A");
    Console.WriteLine("3) Les plus récents d'abord");
    Console.WriteLine("4) Les plus anciens d'abord");
    Console.WriteLine("5) Nom des créateur (A-Z)");

    var choix = Convert.ToInt32(Console.ReadLine());
    app.TypeTriJeuSelected = (TypeTri)choix;
    Test_Afficher_Jeux(app);


}

3 références
public static void Test_Afficher_Jeux(Manager mgr)
{
    foreach(Jeu j in mgr.TousLesJeux)
    {
        Console.WriteLine($"- {j.Informations.Nom}");
    }
}

2 références
private static void Test_Afficher_Favoris(Manager app)
{
    app.TypeTriJeuSelected = TypeTri.A_Z;
    foreach (var jeu in app.TousLesJeux.Where(jeu => jeu.IsFavoris))
    {
        Console.WriteLine($"- {jeu.Informations.Nom}");
    }
}
```

(ici il n'y en a que 3, pour les autres : cf ConsoleTest/Program)

Je sais utiliser LINQ :

Nous avons, dans notre *Manager*, la liste de tous les jeux disponibles dans l'application. Cette liste est reliée à une propriété calculée qui permet d'effectuer la recherche de jeux dans la liste, ainsi que le tri. Pour faire cela on utilise LINQ, ce qui facilite grandement les choses :

```
public IList<Jeu> TousLesJeux
{
    get
    {
        IList<Jeu> JeuxRecherchés = new List<Jeu>();
        if (String.IsNullOrEmpty(JeuRecherche))
        {
            JeuxRecherchés = tousLesJeux;
        }
        else
        {
            JeuxRecherchés = tousLesJeux.Where(j =>j.Informations.Nom.ToLower().Contains(JeuRecherche.ToLower())).ToList();
        }

        switch (TypeTriJeuSelected)
        {
            case TypeTri.Z_A:
                return JeuxRecherchés.OrderByDescending(jeu => jeu.Informations.Nom).ToList();

            case TypeTri.Premier_Anciens:
                return JeuxRecherchés
                   .OrderBy(jeu => jeu.Informations.DateCreation.Date)
                   .ThenBy(jeu => jeu.Informations.Nom)
                   .ToList();

            case TypeTri.Premier_Récents:
                return JeuxRecherchés
                    .OrderByDescending(jeu => jeu.Informations.DateCreation.Date)
                    .ThenBy(jeu => jeu.Informations.Nom)
                    .ToList();
            case TypeTri.NomCréateur:
                return JeuxRecherchés
                    .OrderBy(jeu => jeu.Informations.Createur.Nom)
                    .ThenBy(jeu => jeu.Informations.Nom)
                    .ToList();
            default:
                return JeuxRecherchés.OrderBy(jeu => jeu.Informations.Nom).ToList();

        }

    }

    set => tousLesJeux = value;
```

Je sais gérer les évènements :

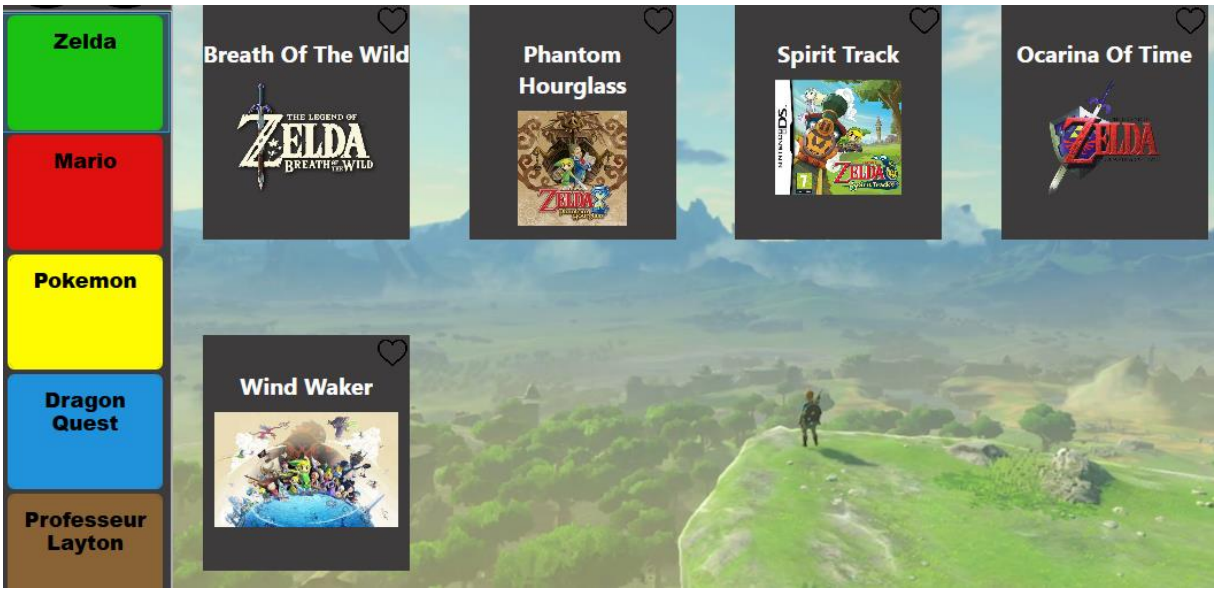
Nous avons utilisé l'évènement *PropertyChanged* pour notifier lorsqu'une propriété avait changé, surtout dans le *Manager :*

```
5 références
public TypeTri TypeTriJeuSelected
{
    get => typeTriJeuSelected;
    set
    {
        if(typeTriJeuSelected != value)
        {
            typeTriJeuSelected = value;
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("TousLesJeux"));
        }
    }
}
4 références
public string JeuRecherche
{
    get => jeuRecherche;
    set
    {
        if (jeuRecherche != value)
        {
            jeuRecherche = value;
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("TousLesJeux"));
        }
    }
}
```

Je sais choisir mes layouts à bon escient :

Nous avons dû choisir nos layouts précisément pour que notre application ressemble à ce que nous avions prévu. C'est pourquoi, dans notre fenêtre *MainWindow* nous avons une *Grid.* En effet, nous voulions deux colonnes pour afficher le master (les franchises) à gauche de la fenêtre et le détail à droite (les jeux) :

```
<Grid x:Name="MainGrid" Background="#3D3C3C">

    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="150"/>
        <ColumnDefinition Width="930"/>
    </Grid.ColumnDefinitions>
```



Je sais choisir mes composants à bon escient :

Nous avons utilisé de nombreux composants dans nos UserControl, TextBox pour la recherche, ComboBox pour le tri…

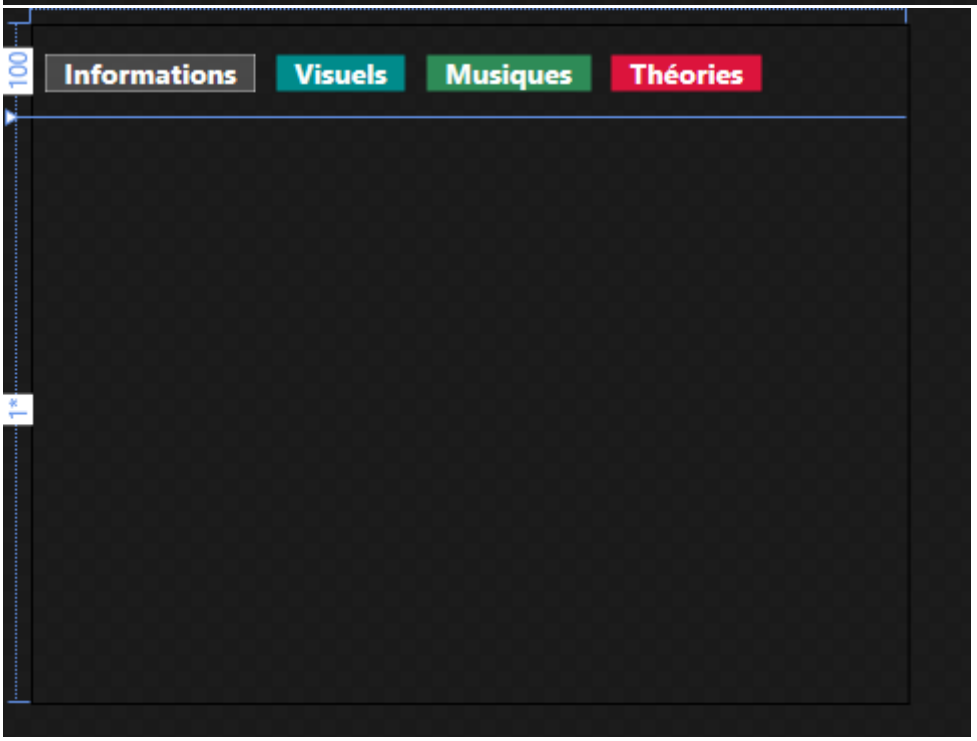
```
<TextBox DockPanel.Dock="Left" VerticalAlignment="Center"  Width="200" Height="45"
            Background="Transparent" Foreground="White" BorderBrush="#FFD4DAEA"
            FontWeight="Bold" FontSize="30"
            Text="{Binding JeuRecherche}"
            Name="TextBoxRecherche"
            KeyDown="TextBoxRecherche_KeyDown"/>
```

```
<ComboBox DockPanel.Dock="Left" Width="200" VerticalAlignment="Center" HorizontalAlignment="Left"
            Background="#3D3C3C"
            SelectedIndex="0"
            Name="ComboBoxTri"
            ItemsSource="{Binding ListTypeTriJeu}"
            SelectionChanged="ComboBoxTri_SelectionChanged">
    <ComboBox.ItemTemplate>
        <DataTemplate>
            <TextBlock Text="{Binding}" FontWeight="Bold" Name="TextComboBox"/>
        </DataTemplate>
    </ComboBox.ItemTemplate>
</ComboBox>
```

(il y en a beaucoup d'autre dans tout le projet ApplicationGraphique)

Je sais créer mon propre composant :

Nous avons dû créer énormément de UserControl comme nous n'avions qu'une vue, les principaux sont *UserControlMain, UserControlFranchise et UserControlVueJeu :*



(Tous les autres se trouvent aussi dans ApplicationGraphique/Nos_UC/)

Je sais intercepter les évènements de la vue :

Que ce soit pour la navigation entre les fragments, la recherche de jeu, le tri, l'écoute d'une musique, etc., nous avons besoin d'intercepter les évènements de la vue presque partout dans *ApplicationGraphique*, en voici plusieurs exemples :

```
1 référence
private void ListBoxListeJeux_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    mgr.JeuSelected = (sender as ListBox).SelectedItem as BibliothèqueApplication.Jeu;
    mgr.FranchiseSelected = null;
    nav.NavigateTo("Jeu");
}

1 référence
private void ButtonRecherche_Click(object sender, RoutedEventArgs e)
{
    mgr.JeuRecherche = TextBoxRecherche.Text;
}

1 référence
private void TextBoxRecherche_KeyDown(object sender, KeyEventArgs e)
{
    if (e.Key == Key.Enter)
    {
        (sender as TextBox).MoveFocus(new TraversalRequest(FocusNavigationDirection.Previous));
    }
}

1 référence
private void ComboBoxTri_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    mgr.TypeTriJeuSelected = (TypeTri)ComboBoxTri.SelectedItem;
}
```

Code behind UserControlMainWindow

```
private void ButtonPlay_Click(object sender, RoutedEventArgs e)
{
    player.Play();
}

1 référence
private void ButtonPause_Click(object sender, RoutedEventArgs e)
{
    player.Pause();
}

1 référence
private void ButtonStop_Click(object sender, RoutedEventArgs e)
{
    player.Stop();
}
```

Code Behind UserControlMusique4ItemsControlMusique

```
private void ListBoxFranchises_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    if(ListBoxFranchises.SelectedItem != null)
        mgr.FranchiseSelected = ListBoxFranchises.SelectedItem as Franchise;
    mgr.JeuSelected = null;
    nav.NavigateTo("Franchise");
}

/// <summary> Permet de revenir à la page principale. On affiche UserControlMain ...
1 référence
private void Button_RetourMain_Click(object sender, RoutedEventArgs e)
{
    mgr.FranchiseSelected = null;
    mgr.JeuSelected = null;
    nav.NavigateTo("Main");
}

/// <summary> Permet d'afficher la page des favoris. On affiche UserControlFranc ...
1 référence
private void Button_Favoris_Click(object sender, RoutedEventArgs e)
{
    mgr.FranchiseSelected = null;
    mgr.JeuSelected = null;
    nav.NavigateTo("Favoris");
}
1 référence
private void ListBoxFranchises_LostFocus(object sender, RoutedEventArgs e)
{
    (sender as ListBox).SelectedItem = null;
}
/// <summary> Permet de sauvegarder les favoris ajoutés dans le fichier texte
1 référence
private void Window_Closing(object sender, System.ComponentModel.CancelEventArgs e)
{
    new PersistanceFichierTexte().SauvegardeDonnees(mgr.TousLesJeux, mgr.ToutesLesFranchises);
}
```

Code Behind MainWindow

Je sais gérer le DataBinding sur mon master :

Pour afficher les franchises à gauche de l'application, il nous a fallu bind notre *ListBoxFranchises* au dictionnaire *ToutesLesFranchises* du *Manager*. Pour récupérer ses franchises on se bind donc sur ToutesLesFranchises.Keys (étant la liste des clés du dictionnaire) :

```
<ListBox Grid.Row="1" Background="Transparent" MaxHeight="560"
    ScrollViewer.HorizontalScrollBarVisibility="Disabled"
    ScrollViewer.VerticalScrollBarVisibility="Hidden"
    SelectionChanged="ListBoxFranchises_SelectionChanged"
    LostFocus="ListBoxFranchises_LostFocus"
    Name="ListBoxFranchises"
    ItemsSource="{Binding ToutesLesFranchises.Keys}">
   <ListBox.ItemTemplate>
      <DataTemplate>
         <Border HorizontalAlignment="Center" BorderThickness="2" Padding="10"
                 CornerRadius="5" Height="100" Width="135">
            <Border.BorderBrush>
               <SolidColorBrush Color="{Binding Couleur}"/>
            </Border.BorderBrush>
            <Border.Background>
               <SolidColorBrush Color="{Binding Couleur}"/>
            </Border.Background>
            <TextBlock  HorizontalAlignment="Center" TextAlignment="Center"
                        FontSize="18" FontWeight="Bold" FontFamily="Arial Black"
                        TextWrapping="Wrap" Text="{Binding Nom}"/>
         </Border>
      </DataTemplate>
   </ListBox.ItemTemplate>
</ListBox>
```

Ce qui donne ça avec nos données :

(Voir aussi le binding de ApplicationGraphique/UserControlFranchise)

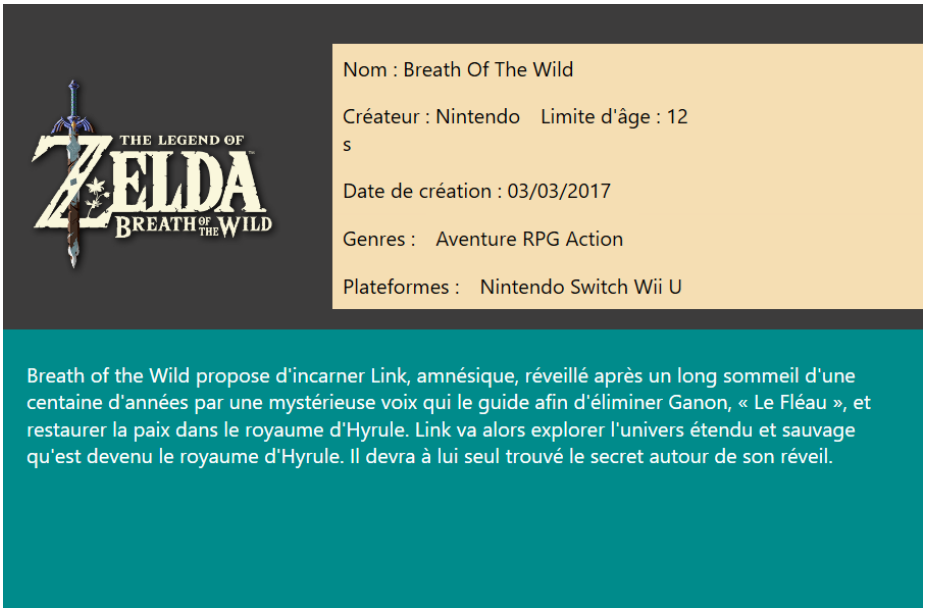
Je sais gérer le DataBinding sur mon detail :

Le detail de notre application est l'affichage du jeu sélectionné. Ainsi la vue d'un *Jeu* est bind sur *JeuSelected* de *Manager*. Cependant, cette vue est séparée en onglets, que sont *UserControlInformations, UserControlVisuels, UserControlMusiques et UserControlTheories.*

Exemple : *UserControlInformations*

```
<Image Margin="30" Source= "{Binding Vignette}"/>

<StackPanel Grid.Column="1" Margin="20" Name="StackPanel_Jeu"  Background="Wheat">
    <TextBlock Background="Wheat" Padding="10" FontSize="20" Name="TextBlockNom"
               Text="{Binding Informations.Nom, StringFormat=Nom : {0:#}}"></TextBlock>
    <StackPanel Orientation="Horizontal" >
        <TextBlock Background="Wheat" Padding="10" FontSize="20" Name="TextBlockCreateur"
                   Text="{Binding Informations.Createur.Nom, StringFormat=Créateur : {0:#}}"/>
        <TextBlock Background="Wheat" Padding="10" FontSize="20" Name="TextBlockLimiteAge"
                   Text="{Binding Informations.LimiteAge, StringFormat=Limite d\'âge : {0:#}}"/>

    </StackPanel>
    <TextBlock Background="Wheat" Padding="10" FontSize="20" Name="TextBlockDateCreation"
               Text="{Binding Informations.DateCreation, StringFormat=Date de création : {0:dd/MM/yyyy}}"/>
    <StackPanel Orientation="Horizontal">
        <TextBlock FontSize="20" Padding="10">Genres : </TextBlock>
        <ItemsControl Background="Wheat" Padding="10" FontSize="20" Name="TextBlockGenres"
                      ItemsSource="{Binding Informations.LesGenresToString}">
            <ItemsControl.ItemsPanel>
                <ItemsPanelTemplate>
                    <WrapPanel />
                </ItemsPanelTemplate>
            </ItemsControl.ItemsPanel>
        </ItemsControl>
    </StackPanel>
    <StackPanel Orientation="Horizontal">
        <TextBlock FontSize="20" Padding="10">Plateformes : </TextBlock>
        <ItemsControl Background="Wheat" Padding="10" FontSize="20" Name="TextBlockPlateformes"
                      ItemsSource="{Binding Informations.LesPlateformesToString}">
            <ItemsControl.ItemsPanel>
                <ItemsPanelTemplate>
                    <WrapPanel />
                </ItemsPanelTemplate>
            </ItemsControl.ItemsPanel>
        </ItemsControl>
    </StackPanel>
```

Design avec un jeu :



Je sais gérer le DataBinding et les Dependency Property sur mes UserControl :

Le DataBinding sur les *UserControl* a été effectué beaucoup de fois, puisque nous en possédons un grand nombre, cependant, un seul *UserControl* possède des *Dependency Property : UserControlJeu,* qui est utilisé dans *UserControlFranchise* pour afficher chaque *Jeu*.

Ainsi le binding de fais sur les *Dependency Property,* et non sur celles du *Jeu.*

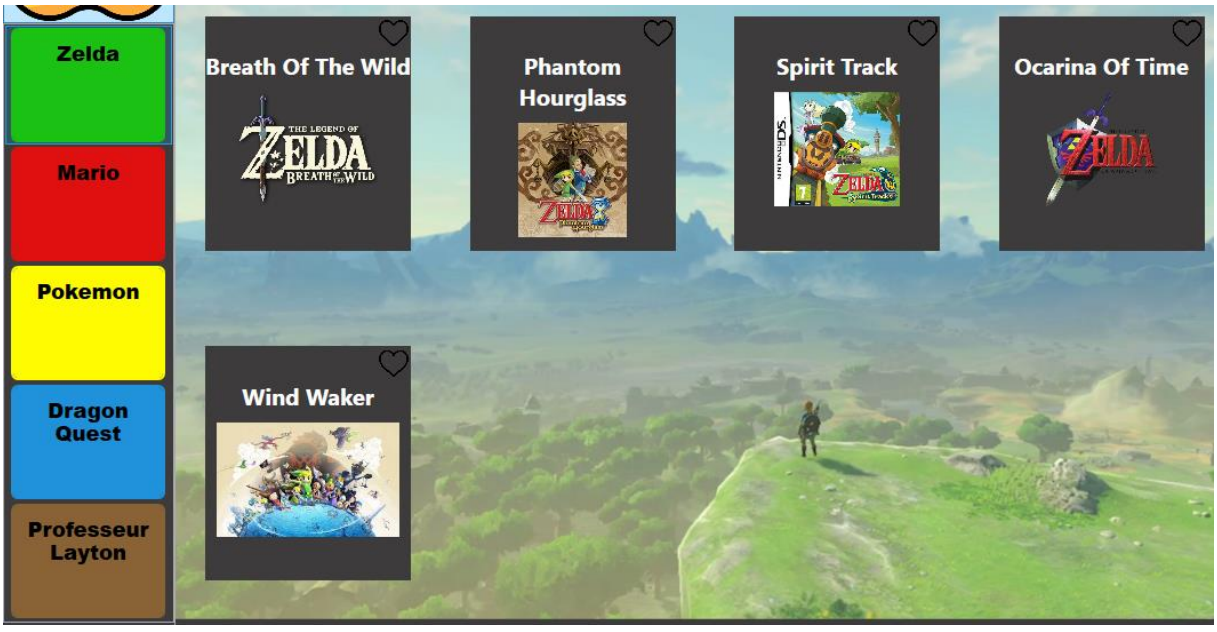
```
<StackPanel Background="#3D3C3C" >
    <Button HorizontalAlignment="Right" Background="Transparent" BorderBrush="Transparent" Width="30" Height="30"
            Name="Button_JeuFavori"
            Click="Button_JeuFavori_Click">
        <Image Source="{Binding LogoFavori, ElementName=root}"/>
    </Button>
    <TextBlock Foreground="White" HorizontalAlignment="Center" TextAlignment="Center" FontWeight="Bold" FontSize="20" TextWrapping="Wrap"
               Text="{Binding Nom, ElementName=root}"/>
    <Image Margin="10" HorizontalAlignment="Center" Width="auto" Height="auto" MaxHeight="100"
           Source="{Binding Image, ElementName=root}"/>
</StackPanel>
```

```
public string Image
{
    get { return (string)GetValue(ImageProperty); }
    set { SetValue(ImageProperty, value); }
}

// Using a DependencyProperty as the backing store for Image.  This enables animation, styling, binding, etc...
public static readonly DependencyProperty ImageProperty =
    DependencyProperty.Register("Image", typeof(string), typeof(UserControlJeu));

0 références
public string Nom
{
    get { return (string)GetValue(NomProperty); }
    set { SetValue(NomProperty, value); }
}

// Using a DependencyProperty as the backing store for Nom.  This enables animation, styling, binding, etc...
public static readonly DependencyProperty NomProperty =
    DependencyProperty.Register("Nom", typeof(string), typeof(UserControlJeu));



0 références
public string LogoFavori
{
    get { return (string)GetValue(LogoFavoriProperty); }
    set { SetValue(LogoFavoriProperty, value); }
}

// Using a DependencyProperty as the backing store for LogoFavori.  This enables animation, styling, binding, etc...
public static readonly DependencyProperty LogoFavoriProperty =
    DependencyProperty.Register("LogoFavori", typeof(string), typeof(UserControlJeu));
```

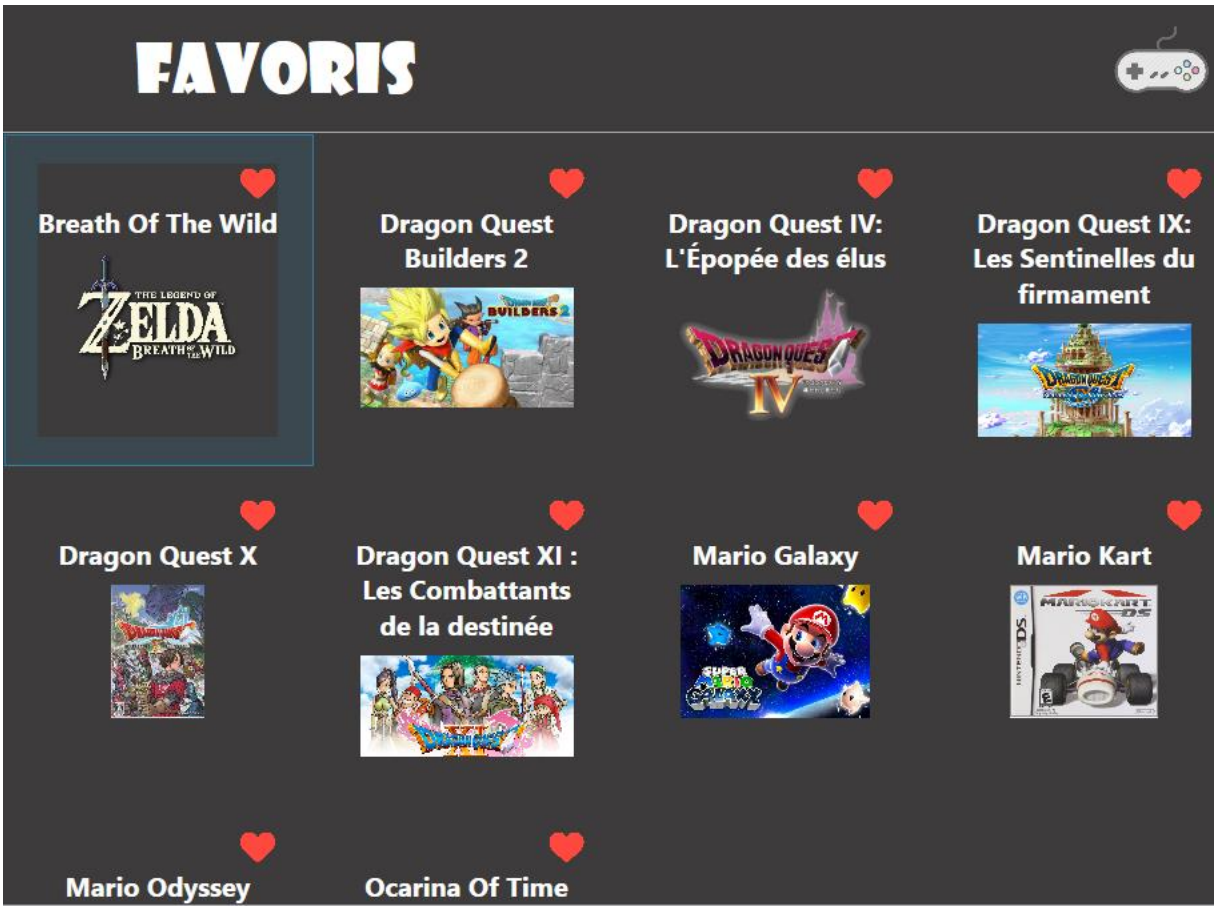
Je sais développer un Master/Detail :

Nous avons tout fait dans le but de produire un master détail. Même s' il y a plusieurs masters, in fine le detail reste le jeu sélectionné. Pour y arriver on peut donc passer par la liste de jeux, grâce, par exemple, à la recherche, ou alors depuis une franchise, ou encore depuis les favoris.

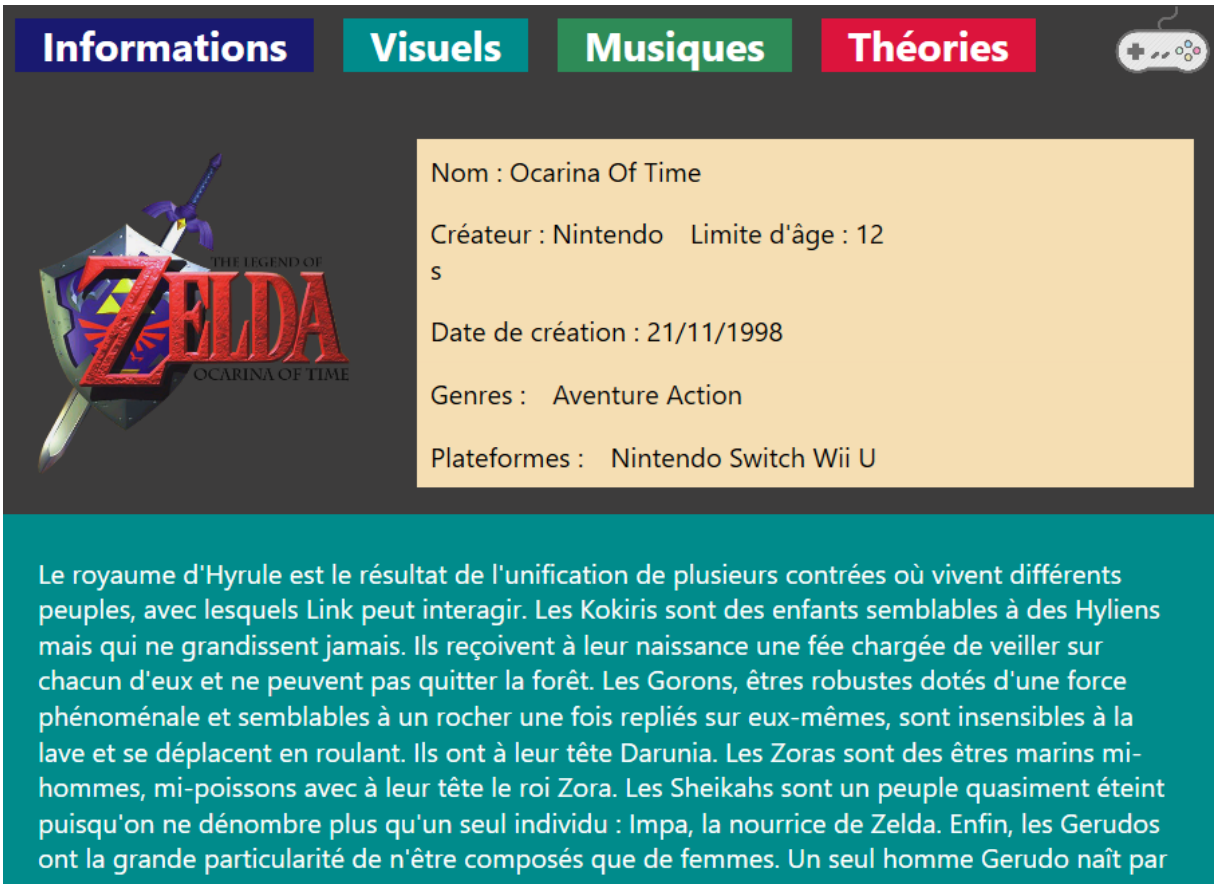
<u>Master 1 :</u> Liste des jeux de UserControlMainWindow



<u>Master 2 :</u> Franchise sélectionnée (ici => Zelda)



<u>Master 3 :</u> Onglet des favoris



<u>Detail :</u> Jeu sélectionné (ici => Ocarina Of Time)

Je sais réaliser une vidéo de 1 à 3 minutes qui montre la démo de mon application.

Notre vidéo se trouve dans le répertoire document.

Je sais coder la persistance au sein de mon application

Dans notre application seulement les jeux en favoris sont persistés. Quand l'utilisateur quitte l'application, il retrouve ses favoris. On utilise des méthodes *ChargerDonnees()* et *SauvegardeDonnees().*

Je sais coder une fonctionnalité qui m'est personnelle

Nous avons codé un lecteur de musique. En plus de cela nous avons une méthode de tri et de recherche personnelle.

Je sais coder la persistance au sein de mon application.

Nous avons codé un lecteur de musique. En plus de cela nous avons une méthode de tri et de recherche personnelle.

Je sais utiliser SVN.

#			Date	Auteur	Commentaire
163	◉		14/06/2021 22:15	Victor GAILLARD	Début des preuves.docx
162	○	◉	14/06/2021 21:04	Victor GAILLARD	Documentation de App/MainWindow/Navigator + vidéo de présentation
161	○	○	14/06/2021 18:30	Mathilde ORILLON	finalisation de la doc, présence des 3 pdf
160	○	○	14/06/2021 15:18	Victor GAILLARD	Création Projet JVDex_Setup + installateur de l'appli JVDex_Setup.msi
159	○	○	13/06/2021 22:47	Mathilde ORILLON	dernière modif musique + ajout des données
158	○	○	13/06/2021 21:12	Victor GAILLARD	Persistance
157	○	○	13/06/2021 19:59	Mathilde ORILLON	ajout des jeux pokemon dans la liste
156	○	○	13/06/2021 19:22	Mathilde ORILLON	essai musique
155	○	○	13/06/2021 17:04	Mathilde ORILLON	franchise pokemon finie
154		○	13/06/2021 15:08	Victor GAILLARD	Modifs vues Theorie + Musique

Voir les différences

Je sais développer une application qui compile

Notre application compile, nous n'avons aucune erreur.

Je sais développer une application fonctionnelle

Notre application compile, nous n'avons aucune erreur.

Je sais mettre à disposition un outil pour déployer mon application

Le fichier est JVDexbin/JVDex_Setup.msi