TEST TECHNIQUE UGLA

Julien Vasseur

Etape 1 et 2 : Connexion et requêtes à la BDD

Après avoir mis en place WampServer64 sur mon ordinateur et avoir installé la base de données, ma première problématique était de faire communiquer Unity avec la BDD.

Mon idée était de récupérer la base de données et de la convertir en fichier Json sur mon ordinateur, afin de pouvoir la lire simplement avec Unity, sans avoir à faire des requêtes constantes.
J'ai principalement utilisé comme référence la documentation de Php ainsi que ce cours de W3Schools pour l'utilisation de la méthode json_encode.
Voici quelques extraits de code du fichier index.php:

```php
$catalogue = new Catalogue();

// Fetch series data from the database
$sql = "SELECT id, title, genre, note, episodes FROM series";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
    // Create an array to store series data
    $series = array();

    while ($row = $result->fetch_assoc()) {
        // Add series data to the array
        $series[] = $row;
    }

    $catalogue->series = $series;

    echo "Series data found.<br>";
} else {
    echo "No series data found.<br>";
}
```

```php
// Fetch genre data from the database
$sql = "SELECT id, Genre FROM genre";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
    // Create an array to store genre data
    $genre = array();

    while ($row = $result->fetch_assoc()) {
        // Add genre data to the array
        $genre[] = $row;
    }

    $catalogue->genre = $genre;

    echo "Genre data found.<br>";
} else {
    echo "No genre data found.<br>";
}
```

```php
// Convert the array to JSON
$json = json_encode($catalogue, JSON_PRETTY_PRINT);

// Save the JSON data to a file
file_put_contents('series.json', $json);
echo "JSON file generated successfully.";

// Close the database connection
$conn->close();
```

Pour la WebRequest nécessaire pour accéder au lien et lancer le script Php, j'ai suivi l'exemple dans la documentation de Unity sur les UnityWebRequests, disponible ici.

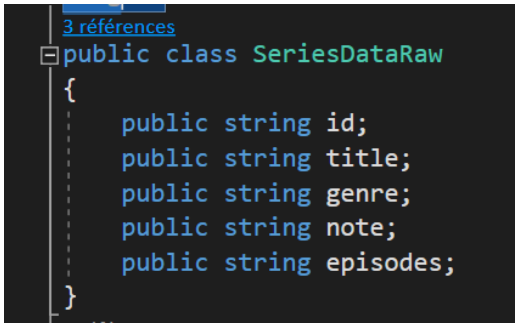
Grâce à cela, j'avais à présent à ma disposition un fichier Json comportant toutes les données de séries et de genre de la base de données.

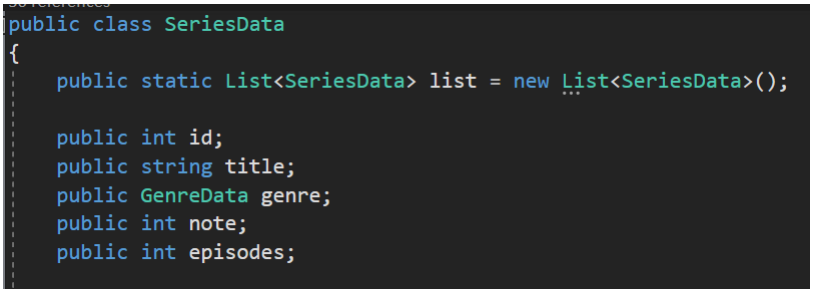
Etape 3 : Classe stockant le résultat de la requête

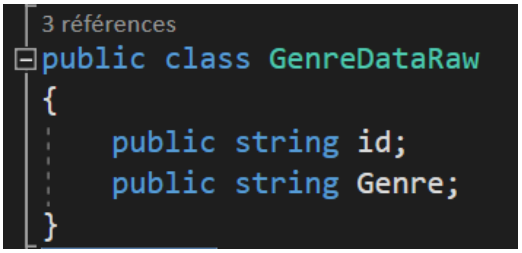
Le principal problème de cette approche, c'est que toutes les données récupérées sont de type string (chaîne de caractères). Il me fallait donc un moyen de convertir ces chaînes de caractère en leur valeur d'origine.

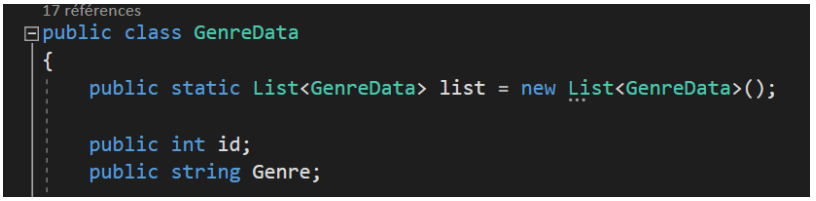
L'objectif est simple : stocker les données de la table "series" dans une classe "SeriesData", et celles de la table "genre" dans une classe "GenreData".

Pour ce faire, j'ai opté pour la création de classes intermédiaires, afin de pouvoir en premier stocker les données sous formes de chaînes puis les convertir vers leur type d'origine.
Voici les différentes classes utilisées:









Il est donc par exemple possible d'écrire *MySeries.genre.Genre* pour récupérer "Action" ou "Sci-fi".

Pour la lecture des données du fichier Json, j'ai utilisé la librairie NewtonSoft, qui propose des méthodes simples pour exploiter le format. Voici un lien vers leur documentation.

La conversion se déroule au sein de la méthode ReadJson de la classe SeriesLoader:

```
1 référence
private void ReadJSON()
{
    // Clear old lists
    GenreData.list.Clear();
    SeriesData.list.Clear();

    string jsonString = File.ReadAllText(_JsonFilePath);

    // Get genres
    List<GenreDataRaw> lGenreListRaw = JsonConvert.DeserializeObject<RootObject>(jsonString).genre;

    foreach (GenreDataRaw lRaw in lGenreListRaw)
    {
        GenreData lData = new GenreData();

        lData.id = StringToInt(lRaw.id);
        lData.Genre = lRaw.Genre;

        GenreData.list.Add(lData);
    }
```

```
    // Get series
    List<SeriesDataRaw> lSeriesListRaw = JsonConvert.DeserializeObject<RootObject>(jsonString).series;

    foreach (SeriesDataRaw lRaw in lSeriesListRaw)
    {
        SeriesData lData = new SeriesData();

        lData.id = StringToInt(lRaw.id);
        lData.title = lRaw.title;
        lData.genre = GenreData.GetGenreByName(lRaw.genre);
        lData.note = StringToInt(lRaw.note);
        lData.episodes = StringToInt(lRaw.episodes);

        SeriesData.list.Add(lData);
    }

    ON_SeriesUpdated.Invoke();
}
```

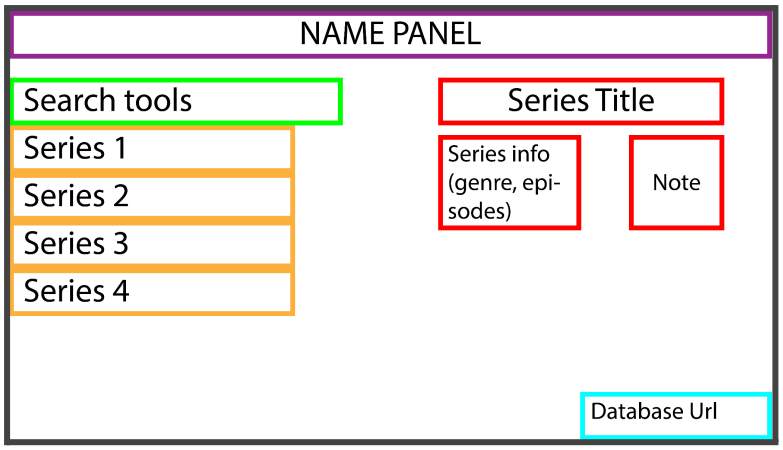
Etape 4 : Utilisation des données en affichant dans une UI le contenu de la base de données

Pour l'organisation de l'interface, j'ai voulu m'inspirer de l'écran de sélection de map du jeu de rythme "Osu!". En effet, celui-ci propose des options de tri, mais aussi de regroupement : par exemple, pouvoir mettre côte à côte les maps du même artiste, ou de la même difficulté, indépendamment des options de tris.
Dans cet exemple, les maps sont triées par titre, et groupées par difficulté. Cela permet de chercher des maps de son niveau, et de les trier par ordre alphabétique.

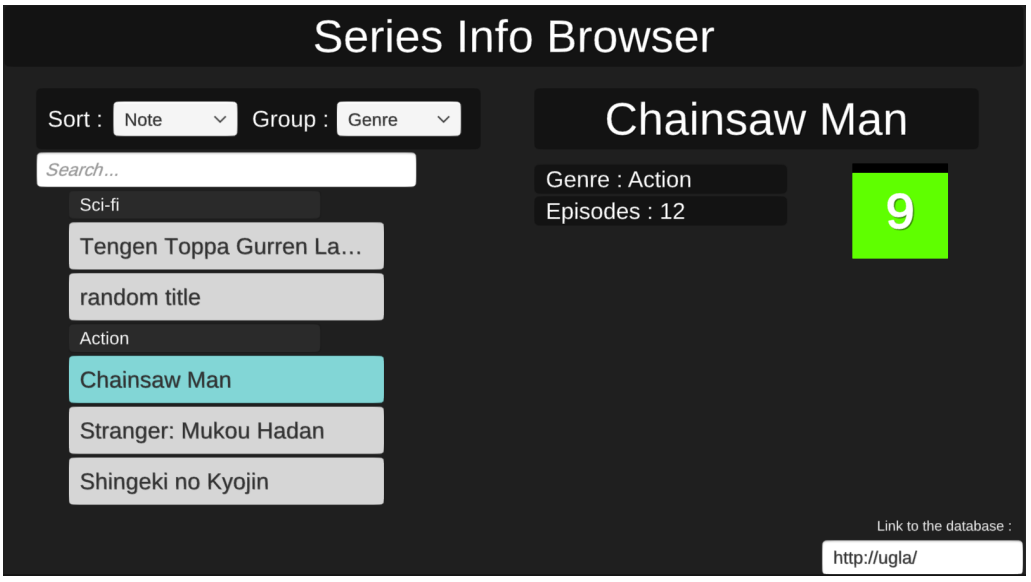
Retranscrit dans le contexte d'une base de données de séries, cela permettrait par exemple de grouper par genre et de trier par note pour facilement trouver la meilleure série d'action.

Autre exemple, si l'on cherchait à faire une base de données de meubles, cela permettrait de facilement trouver le meuble de type "lit" le moins cher ou le plus spacieux…

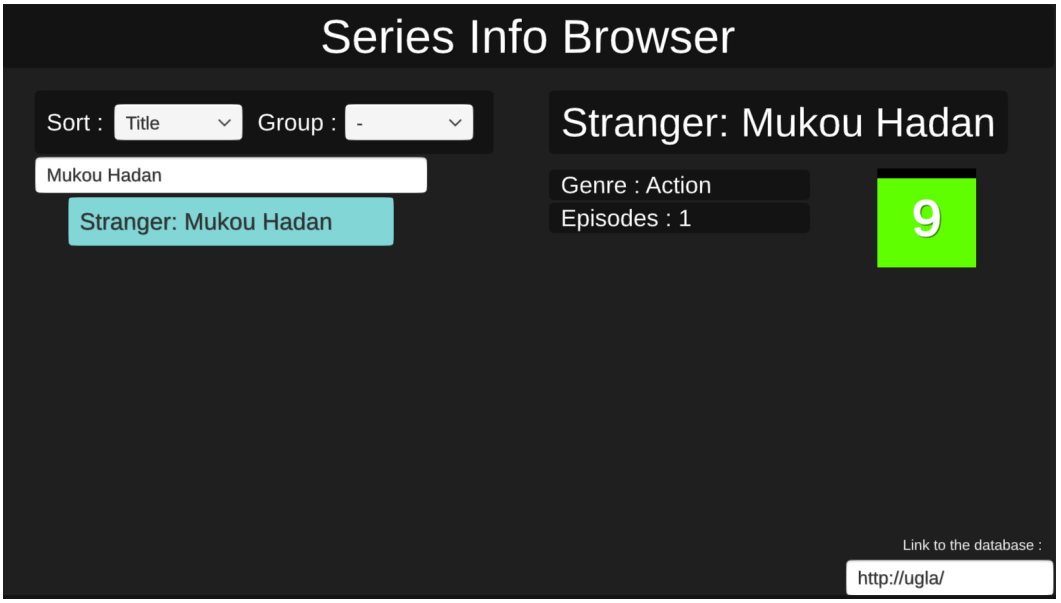
Voici ma première ébauche d'interface réalisée sur Illustrator:



Et voici le résultat final sur Unity:
1 : Grouper par genre + trier par note



2 : Recherche par titre (peu importe l'emplacement des caractères dans le résultat de la recherche)



3 : Recherche par genre

Conclusion : Fonctionnement détaillé du produit final

Lors du lancement, le script SeriesLoader envoie une WebRequest à l'url affiché en bas à droite de l'interface. Celle-ci peut être modifiée en fonction de la mise en place de l'import de la base de données. Un fichier Json est ensuite généré, comportant toutes les données de la BDD.

Ce fichier Json est ensuite lu et interprété dans la classe SeriesLoader, et les données sont stockées respectivement dans les listes statiques des classes SeriesData et GenreData.

Après cela, la liste des séries affichée à gauche de l'écran est générée grâce aux données de la classe SeriesData. Cliquer sur l'une des séries invoque un événement qui est récupéré par la classe SeriesInfo pour afficher les informations sur la droite de l'interface.

Les différentes options de recherches (*Sort* et *Group*) réarrangent la liste des résultats.
La barre de recherche masque les résultats qui ne comportent pas la chaîne de caractère dans le titre ou le genre de la série.