TP 10

Les objectifs de ce dixième TP sont :
- l'écriture de feuilles de style XSLT.

Ce TP est l'examen que j'ai donné le 16 janvier 2012 aux étudiants de GI.
N'oubliez pas que pour vous entrainer, vous trouverez sur l'ENT les annales des années passées.

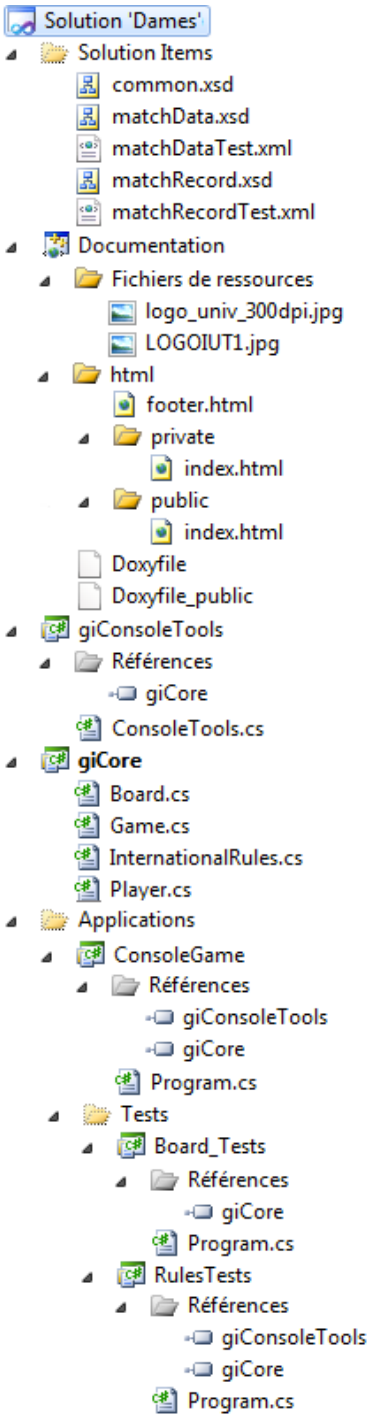
Sujet - Arborescence Visual

CONTEXTE ET OBJECTIFS

Le schéma XML visual.xsd, permet d'écrire des instances XML représentant des solutions de Visual Studio (avec leurs projets, leurs fichiers...), et le fichier visual.xml est une instance particulière d'une solution.

L'objectif de cette épreuve est de transformer l'instance XML en une page html ressemblant (presque) à la capture d'écran ci-contre. Pour cela, nous utiliserons un pattern composé de deux patrons principaux : un `template` dit «`display`» qui affichera les informations (image, nom) et un `template` «`traverse`» qui propagera l'information aux sous-éléments et notamment la tabulation pour donner cet effet escalier.

Pour atteindre ce résultat, il vous est proposé de suivre le plan proposé par les questions suivantes. Vous pourrez également rendre un fichier XSLT par question, pour éviter de détruire vos réponses au fur et à mesure des questions, par exemple `visualQ1.xsl`, `visualQ2.xsl`, `visualQ3.xsl`...



capture d'écran de Visual Studio

QUESTION 1 : MISE EN PLACE DU PATTERN

Écrivez une feuille de style XSLT permettant de transformer le fichier `visual.xml` en une table html (cf. `visualQ1.html`) affichant les résultats comme dans l'image ci-contre.

Pour cela, vous pourrez utiliser (au moins) deux templates :

- le premier, «`display`», affichera le nom de l'élément représenté par le noeud courant (dossier de solution, dossier, fichier, projet...) quel qu'il soit (sauf la solution qui affichera entre parenthèses le nombre de projets qu'elle contient), et appellera le `template` «`traverse`» sur le noeud courant ;
- le deuxième, «`traverse`», permettra d'appeler le `template` «`display`» sur tous les enfants du noeud courant.

On mettra ainsi en place une récursivité. Dans le cas du fichier `visual.xml`, son utilisation entraine : l'appel du `template` «`display`» sur la solution, affiche son nom puis, via «`traverse`», appelle «`display`» sur ses enfants «*Applications*», «*Solution Items*», «*Documentation*», «*giConsoleTools*», «*giCore*» ; l'appel du `template` «`display`» sur «*Applications*» entraine alors, via «`traverse`», l'appel du `template` «`display`» sur «*Tests*» et «*ConsoleGame*», ainsi de suite...

Il vous est également demandé de trier les enfants d'un noeud dans l'ordre suivant :

- les dossiers de solutions,
- les projets,
- les références,
- les dossiers,
- les fichiers.

S'il y a plusieurs enfants du même type, ils sont alors triés par ordre alphabétique.

Note : la fonction XPath `local-name(node)` rend la partie locale du noeud `node`. Ainsi, `local-name(node)!='project'` rend `false` si `node` est un *project*, et `true` sinon. De plus, alphabétiquement parlant, `false` arrive avant `true`...

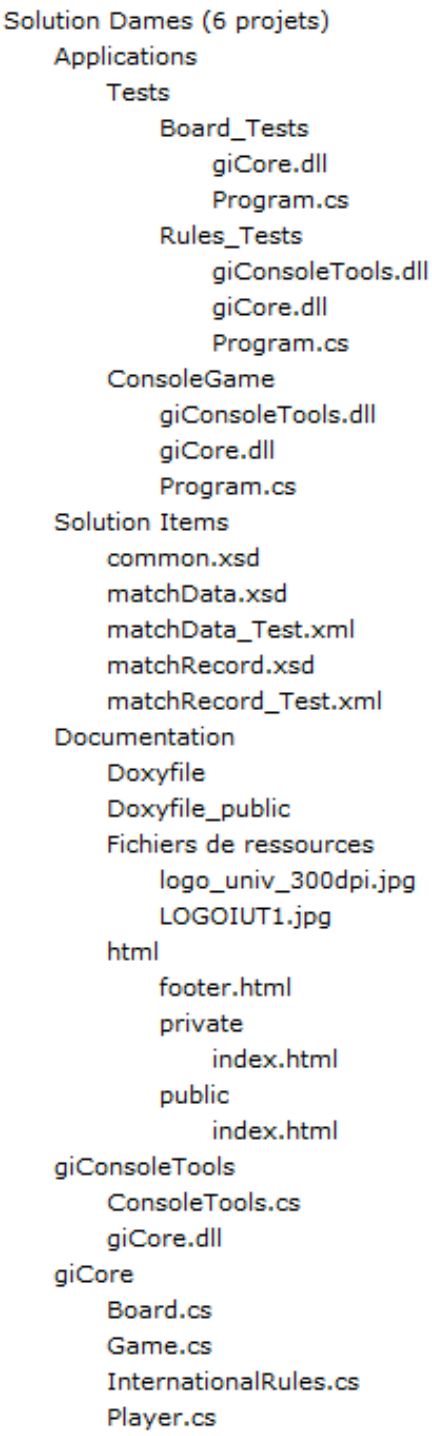
Solution Dames (6 projets)
Applications
Tests
Board_Tests
Références
giCore.dll
Program.cs
Rules_Tests
giCore.dll
giConsoleTools.dll
Program.cs
ConsoleGame
giCore.dll
giConsoleTools.dll
Program.cs
Solution Items
common.xsd
matchData.xsd
matchData_Test.xml
matchRecord.xsd
matchRecord_Test.xml
Documentation
Fichiers de ressources
logo_univ_300dpi.jpg
LOGOIUT1.jpg
html
private
index.html
public
index.html
footer.html
Doxyfile
Doxyfile_public
giConsoleTools
giCore.dll
ConsoleTools.cs
giCore
Board.cs
Game.cs
InternationalRules.cs
Player.cs

QUESTION 2 : TABULATION

Nous allons maintenant ajouter les tabulations pour mieux représenter la hiérarchie.

À l'aide de passage de paramètre(s) dans vos `templates`, ajoutez un `padding` de 25 pixels par niveau de profondeur dans la hiérarchie.

Par exemple, vous pouvez créer un paramètre «`padding`» dans vos `templates` et une variable «`TABULATION`» globale. Lors de l'application du `template` «`traverse`», vous passez alors en paramètre (aux cas particuliers près...) `«padding» = «padding + TABULATION»`, ce qui entrainera le décalage voulu (cf. `visualQ2.html`)



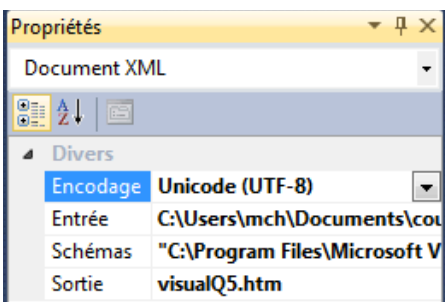
QUESTION 3 : IMAGES

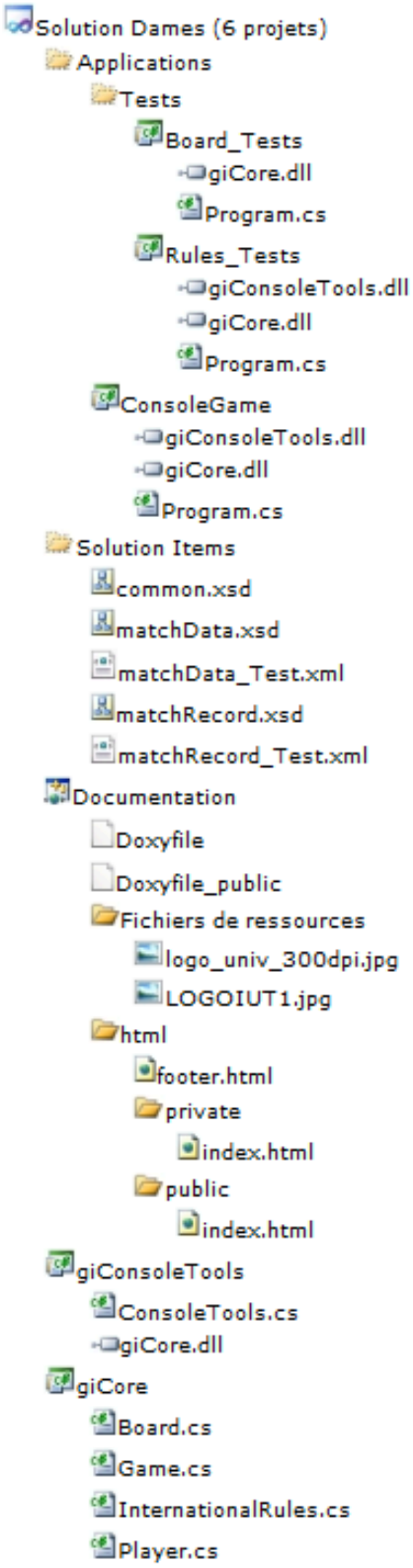
Écrivez un `template` sans attribut `match`, qui transforme un noeud en une image en fonction de son type. Puis modifiez le `template` «`display`» pour ajouter l'image adaptée devant chaque nom de noeud (cf. `visualQ3.html`).

type de noeud	image
solution	solution.jpg
dossier de solution	solutionFolder.jpg
dossier	folder.jpg
projet de type vcproj	vcproj.jpg
projet de type csproj	csproj.jpg
fichier .cs	cs.jpg
fichier .html	html.jpg
fichier .xml	xml.jpg
fichier .xsd	xsd.jpg
fichier .jpg	jpg.jpg
fichier sans extension	sansExtension.jpg
référence	ref.jpg

Note : vous pouvez tester la partie locale d'un noeud `node` avec la fonction XPath `local-name(node)`

Note 2 : pour voir les images depuis la transformation visual studio, affichez les propriétés de la feuille xslt (F_4), puis changez la Sortie en donnant un chemin local (relatif depuis la feuille XSLT). Vous pourrez pour cela placer les images dans le même dossier que vos feuilles XSLT.

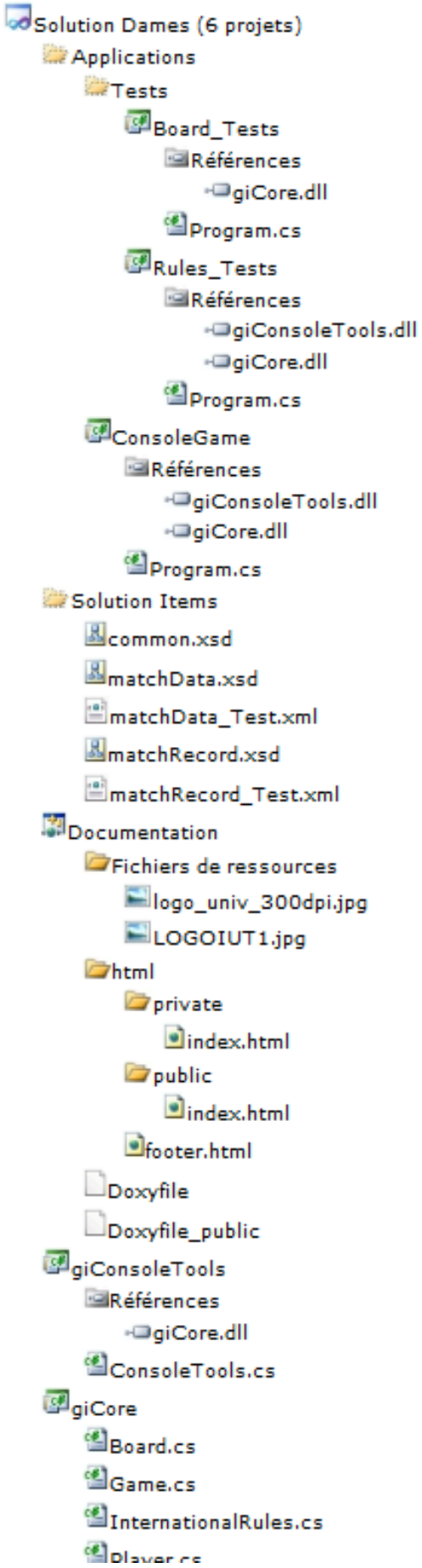




QUESTION 4 : CAS PARTICULIER DES RÉFÉRENCES

Modifiez votre feuille de style pour que les références soient précédées d'un dossier `Références` avec l'image `references.jpg` (cf. `visualQ4.html`). Attention aux tabulations.

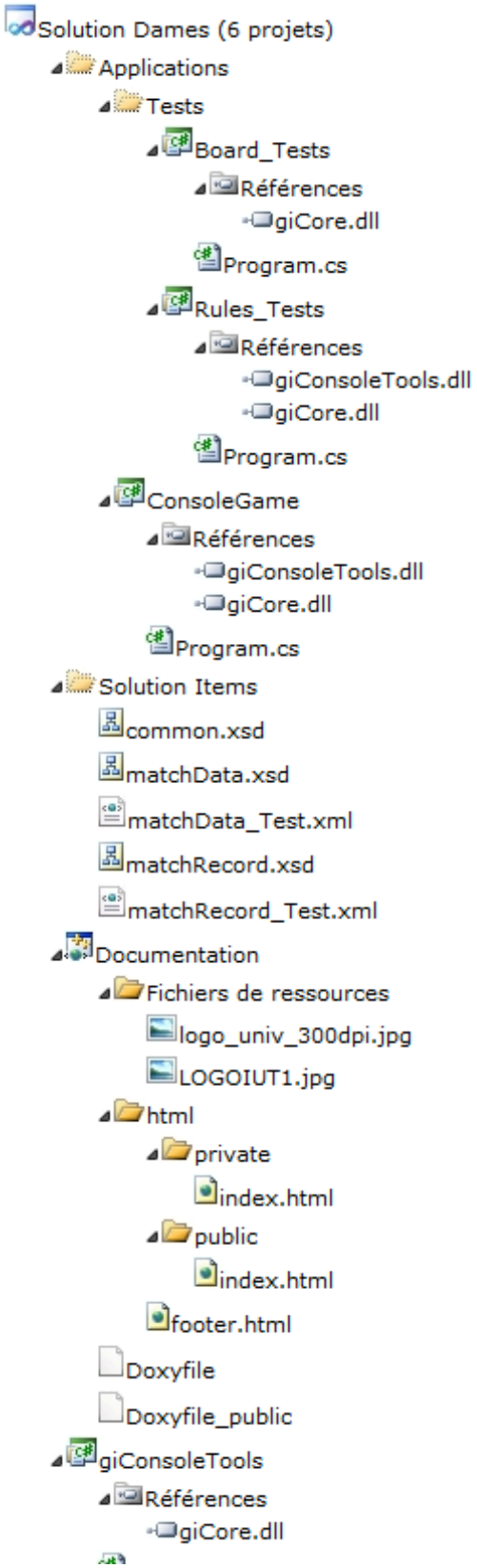
Vous pouvez par exemple, faire une espèce de 'spécialisation' de `template` similaire à une spécialisation d'une classe générique en C++ ou en C#.



(image incomplète)

QUESTION 5 : DIFFÉRENCIER LES NOEUDS POSSÉDANT DES ENFANTS

Enfin pour finir, on souhaite afficher une petite flèche indiquant que le dossier est «développé», seulement devant les éléments possédant des noeuds fils. Pour cela, ne testez pas le type de noeuds, mais testez plutôt si le noeud a des enfants. Si c'est le cas, ajoutez l'image, sauf pour la solution (cf. `visualQ5.html`).



(image incomplète)